Available online at www.sciencedirect.com
ScienceDirect
Omega 37 (2009) 272–281
omega
The International Journal of Management Science
www.elsevier.com/locate/omega

Some heuristic algorithms for total tardiness minimization in a flowshop with blocking☆

Débora P. Ronconi*,[1], Luís R.S. Henriques

Departamento de Engenharia de Produção, Escola Politécnica, Universidade de São Paulo, Av. Prof. Almeida Prado, 128, Cidade Universitária, 05508-900, São Paulo SP, Brazil

Received 9 August 2005; accepted 19 January 2007
Available online 14 February 2007

Abstract

The flowshop scheduling problem with blocking in-process is addressed in this paper. In this environment, there are no buffers between successive machines; therefore intermediate queues of jobs waiting in the system for their next operations are not allowed. Heuristic approaches are proposed to minimize the total tardiness criterion. A constructive heuristic that explores specific characteristics of the problem is presented. Moreover, a GRASP-based heuristic is proposed and coupled with a path relinking strategy to search for better outcomes. Computational tests are presented and the comparisons made with an adaptation of the NEH algorithm and with a branch-and-bound algorithm indicate that the new approaches are promising.

Keywords: Flowshop scheduling; Blocking in-process; Tardiness; Heuristic; GRASP

1. Introduction

This paper analyzes the flowshop scheduling problem with blocking between machines. This environment is characterized by processing n jobs on m machines in the same order, that is, the jth operation of every job must always be conducted on machine j. The processing times of each job on each machine are known. Since there is no buffer storage between machines, queues of jobs waiting in the system for their next operation are not allowed. A job completed on one machine blocks it until the next machine is available for processing. Note that this environment is different from the no-wait flowshop environment. In the latter, there is no machine blocking: once a job is started on the first machine, it must be continuously processed (without interruption) until its completion on the last machine.

According to Hall and Sriskandarajah [1], blocking can be related to the production process itself. Some examples of blocking can be found in concrete block manufacturing, which does not allow stock in some stages of the manufacturing process [2], and 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 [3].

The scheduling performance measure considered in this paper is the minimization of the total tardiness of jobs. The tardiness criterion is of great importance in manufacturing systems because certain costs are incurred when a job is not completed by its due date.

☆ This manuscript was processed by Associate Editor Kubiak.

* Corresponding author. Tel.: +55 11 30915363.
E-mail address: dronconi@usp.br (D.P. Ronconi).

[1] Partially supported by CNPq (PROSUL 490333/2004-4 and Grant 305663/2003-8) and FAPESP (Grant 05/60452-6).

0305-0483/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2007.01.003

These costs include: penalty clauses in the contract, loss of goodwill resulting in an increased probability of losing the customer for some or all future jobs, and a damaged reputation which will turn other customers away [4].

The tardiness of a job can be computed using the recursive equations presenteded by Pinedo [5]. Let $t_1, t_2, \ldots, t_i, \ldots, t_n$ be the sequence to be evaluated, where t_i represents the job that occupies the ith position in the considered sequence. Let $M_1, M_2, \ldots, M_j, \ldots, M_m$ be the ordered sequence of machines, p_{kj} the processing time of job k on machine M_j, and d_k the due date of job k. Let $D_{ti,0}$ denote the start time of job t_i on the first machine and $D_{ti,j}$ the departure time of job t_i on machine M_j. The departure times of each job on each machine are given by the following expressions:

$$D_{t1,0} = 0, \tag{1}$$

$$D_{t1,j} = \sum_{q=1}^{j} p_{t1,q}, \quad j = 1, \ldots, m-1, \tag{2}$$

$$D_{ti,0} = D_{t(i-1),1} \quad i = 2, \ldots, n, \tag{3}$$

$$D_{ti,j} = \max(D_{ti,j-1} + p_{ti,j}, D_{t(i-1),j+1})$$
$$i = 2, \ldots, n, \quad j = 1, \ldots, m-1, \tag{4}$$

$$D_{ti,m} = D_{ti,m-1} + p_{ti,m} \quad i = 1, \ldots, n. \tag{5}$$

Since the completion times of the jobs are known ($D_{ti,j}$), the total tardiness is given by

$$T = \sum_{i=1}^{n} \max(D_{ti,m} - d_{ti}, 0). \tag{6}$$

The level of difficulty of the problem can be assessed using the particular case of a single machine, which is NP-hard for the tardiness criterion [6]. Moreover, it can be shown that the decision version of the flowshop problem with blocking and three machines minimizing the makespan criterion, which is NP-hard in the strong sense, is reducible to the decision version of this problem considering the minimization of the total tardiness (to know whether the makespan is less than or equal to a threshold value y, one needs to define $d_k = y$ for all $k = 1 \ldots n$ and to solve the corresponding tardiness decision problem). See Lenstra et al. [7] for similar cases.

The research on flowshop with blocking is not extensive. An overview of the literature on the makespan criterion follows. A good review can be found in Hall and Sriskandarajah [1], who proved that this problem with three machines is NP-hard in the strong sense. In a previous work, Leisten [8] presented a more complex proof that the equivalent decision version of this problem is NP-complete. This author reduced the three-dimensional matching problem to a special case of the flowshop problem with blocking and three machines.

Among heuristic approaches, McCormick et al. [9] developed an algorithm, known as Profile Fitting, which tries to initially sequence jobs that lead to the minimum sum of idle times and blocking times on machines. A more comprehensive approach is presented by Leisten [10], who compares heuristics adapted from cases of no-wait, unlimited buffers, limited buffers, and two specially designed heuristics which attempt to optimize the utilization of the available buffer storage. The author concludes that the heuristics allowing job passing did not lead to good solutions and that the NEH algorithm proposed by Nawaz et al. [11] performs better. Ronconi [12] suggests three constructive heuristics, including a combination of the Profile Fitting heuristic and the enumeration procedure used by the NEH algorithm. The proposed methods outperform the NEH algorithm in problems involving up to 500 jobs and 20 machines.

Abadi et al. [13] propose a heuristic for minimizing the steady state cycle time to repetitively produce a minimal part set in an m-machine blocking flowshop. The key idea is slowing down operations in order to make a connection between the no-wait flowshop, in which jobs do not wait between operations, and the blocking flowshop. This method can also be applied to minimize the makespan. Caraffa et al. [14] used this concept to develop a genetic algorithm (GA) to minimize this criterion. Computational results indicate that the proposed method shows a better performance than the heuristic developed by Abadi et al. [13]. More recently, Grabowski and Pempera [15] develop a tabu search algorithm that utilizes multimoves to accelerate the convergence of the method. The proposed strategy achieved better results than the GA proposed by Caraffa et al. [14]. Furthermore, this methodology was able to improve the reference makespans provided by the branch-and-bound algorithm (B&B) presented by Ronconi [16].

As far as we know, few studies deal with the total tardiness criterion in the flowshop environment with blocking. Armentano and Ronconi [17] suggest a tabu search-based heuristic with an initial solution given by the algorithm LBNEH [18], which exploits characteristics of the tardiness criterion. Diversification, intensification and neighborhood restriction strategies were also evaluated. The authors report computational tests with problems up to 50 jobs. In another work, Ronconi and Armentano [19] present a B&B algorithm to minimize total tardiness where fixed jobs are placed at the end of the complete sequence. They propose a lower bound for this criterion and, as a by-product, lower bounds for

the makespan and for the sum of job completion times.

In this paper we propose heuristic approaches for the minimization of the total tardiness of jobs in a flowshop with blocking in-process. We present a constructive heuristic that explores the non-existence of intermediate buffers, as well as the characteristics related to the tardiness criterion. The small computational effort of such strategy, which is valuable in some practical applications, is one of the reasons that motivated this study. A GRASP-based (greedy randomized adaptive search procedure) heuristic is also proposed. Furthermore, the metaheuristic is coupled with a path relinking strategy to search for better outcomes. We assessed the performance of the proposed methods through a comparative study with other methods described in the literature.

This paper is organized as follows. Section 2 presents the constructive heuristic, while Section 3 explains how the GRASP metaheuristic was adapted for the focused problem. An additional strategy based on path relinking is also described. Section 4 shows computational experiments with the constructive heuristic, the basic implementation of GRASP and the inclusion of path relinking. The last section summarizes the main results.

2. Constructive heuristic

The proposed heuristic achieves the solution in two steps. First a dispatch rule (FPD—*fitting processing times and due dates*) generates a list of priorities among the jobs. Then, guided by this list, the insertion procedure of the NEH algorithm [11] is applied to generate the final scheduling. This heuristic is henceforth called FPDNEH.

The FPD algorithm works dynamically, i.e., after the selection of a job for a position on the fixed sequence (σ), the list of priorities is reorganized considering all jobs that are not yet fixed. The selection of the first job of this list is of great importance for the performance of the method, because while the first job is been processed on the first machine, the remaining jobs are waiting out of the considered environment, which leads to low performance levels on the whole system. Several alternatives were assessed to select this job. Choosing the job with the smallest value of the sum of its due date and processing time on the first machine (I_k) is the strategy that presents the best results.

For the remaining positions, we use a priority measure F_k composed of two terms. This measure considers the processing time profile of the last fixed job (t_i) in sequence σ, as well as the due dates of candidate jobs. The first term (fit'_k) benefits the jobs that have processing times near the "windows" generated by job t_i on each machine. The main idea is to choose a job that causes the minimum blocking or idle time on machines. The second term ($\text{dynslack}'_k$) aims at increasing the priority of jobs that have a short remaining period of time to be processed before their due date. A description of the FPD algorithm is presented below.

Step 1: Set $\sigma = \varnothing$.

Step 2: Compute I_k for all jobs $k \notin \sigma$:

$$I_k = d_k + p_{k1}. \tag{7}$$

Choose the job with the smallest I_k for the first position on the sequence σ. Make $next\_pos = 2$.

Step 3: Compute LB_k [18] for all jobs $k \notin \sigma$:

$$LB_k = d_k - \sum_{j=1}^{m} p_{kj}. \tag{8}$$

Step 4: Compute the departure time of the last job on σ, t_i, from each machine $D_{ti,j}$, $j = 1, 2, \ldots, m$.

Step 5: Compute the available period of time (b_j) on each machine:

$$b_j = D_{ti,j+1} - D_{ti,j}, \quad j = 1, 2, \ldots, m-1. \tag{9}$$

Step 6: Compute for all jobs $k \notin \sigma$:

$$F_k = \rho\, \text{fit}'_k + (1-\rho)\, \text{dynslack}'_k, \tag{10}$$

where

$$\text{fit}'_k = \frac{\text{fit}_k - \min_{k\notin\sigma}(\text{fit}_k)}{\max_{k\notin\sigma}(\text{fit}_k) - \min_{k\notin\sigma}(\text{fit}_k)},$$
$$\text{dynslack}'_k = \frac{\text{dynslack}_k - \min_{k\notin\sigma}(\text{dynslack}_k)}{\max_{k\notin\sigma}(\text{dynslack}_k) - \min_{k\notin\sigma}(\text{dynslack}_k)}, \tag{11}$$

$$\text{fit}_k = \sum_{j=1}^{m-1} |b_j - p_{kj}| \quad \text{and} \quad \text{dynslack}_k = LB_k - D_{ti,1}. \tag{12}$$

Step 7: Place the job with the smallest value of F_k on position $next\_pos$ of σ.

Step 8: If $next\_pos = n$, stop. Otherwise, make $next\_pos = next\_pos + 1$ and go back to Step 4.

Note that both terms of F_k, fit' (perfect fit) and $\text{dynslack}'$ (dynamic slack) take values between 0 and 1. The objective of this normalization is to avoid the possible distortion caused by the magnitude of the term values. Parameter ρ is used to weigh the terms.

The sequence generated by the FPD algorithm is used as a priority list for the insertion procedure of the NEH algorithm. The two jobs with the highest priority are selected from the list and the two possible sequences

for these jobs are generated. Sequences are evaluated according to the total tardiness of the jobs and the best partial sequence is chosen. The relative positions of those two jobs do not change during the whole algorithm. Then, the job with the third highest priority is selected and three sequences are generated by positioning this job at the beginning, middle and end of the sequence generated in the previous step. The best partial sequence will set the relative positions of these three jobs. Should there be a tie, the original order in the list of priorities is followed. This process is repeated until all jobs are scheduled and the final sequence is found. An interesting analysis about the good performance of the NEH algorithm for flowshops with unlimited buffer can be found in Kalczynski and Kamburowski [20].

3. GRASP metaheuristic

In this section a GRASP-based search heuristic is proposed as a method to explore the solution space. The GRASP metaheuristic, developed by Feo and Resende [21,22], was chosen for this study because it has successfully solved several scheduling problems (see for example [23–25]). Furthermore, GRASP presents special characteristics, such as the simplicity of implementation and a small number of parameters.

The GRASP metaheuristic is a multi-start or iterative process, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. A pseudo-code of this metaheuristic can be found in Resende and Ribeiro [26].

3.1. GRASP basic version

Initially, GRASP was used only with its basic components, which will be henceforth called basic version. The detailed information on how the two GRASP phases were adapted to the focused problem is given below.

3.1.1. Construction phase

GRASP requires a good constructive heuristic to generate multiple initial solutions. A feasible solution is built, adding one element at a time. The element is randomly chosen among the components of a *restricted candidate list* (RCL). Once the element is added to the partial solution, the RCL is updated. This scheme is repeated until a complete feasible solution is found. In this phase, GRASP combines greedy and random construction. The existence of the RCL provides the greedy feature, while the probabilistic feature is given by the randomness to select an element from this list. This process allows the method to build different initial solutions at each time.

For the construction of the RCL, a set of candidate elements C is built with elements that maintain the feasibility of the partial solution. The cost $c(e)$ is calculated for each element $e \in C$, but only the best elements are selected to be part of the RCL. Function $c(e)$ measures the local benefit of including an element in the partial solution. The RCL length can be a function of the desired number of elements or their quality. Let $c^{\min} = \min_{e \in C} c(e)$ and $c^{\max} = \max_{e \in C} c(e)$. Three strategies described by Pitsoulis and Resende [27] were analyzed for the selection of elements:

(i) cardinality_based_scheme: $\lfloor \alpha \cdot n \rfloor$ top-ranked candidates are placed in the RCL, where α is a parameter in [0, 1],
(ii) value_based_scheme$_1$: $RCL = \{e \in C | c(e) \leqslant c^{\min}(1 + p)\}$, where p is a parameter in $[0, \infty)$, and
(iii) value_based_scheme$_2$: $RCL = \{e \in C | c(e) \leqslant c^{\min} + \alpha(c^{\max} - c^{\min})\}$, where α is a parameter in [0, 1].

In the first case, the RCL length is previously defined, while in the other cases it can vary from iteration to iteration. The description of the construction phase follows:

Step 1: Let $x =$ *empty partial sequence*.
Step 2: Let $C =$ {all jobs}.
Step 3: Compute $c(e) = I_e$ (7) for the candidate jobs $e \in C$.
Step 4: Build the RCL using the selected scheme (cardinality_based_scheme, value_based_scheme$_1$ or value_based_scheme$_2$).
Step 5: Choose e from the RCL at random.
Step 6: Allocate job e at the end of the partial sequence x.
Step 7: Remove the selected job e from C.
Step 8: Compute $c(e) = F_e$ (10) for the candidate jobs $e \in C$.
Step 9: If x is not complete, go back to Step 4.

3.1.2. Local search phase

The efficiency of a local search depends on several factors such as the neighborhood definition, the search strategy and the initial solution. In this study, the neighborhood is defined by the insertion move, which consists of removing a job from its original position and inserting it on the $n - 1$ remaining positions. This move generates a neighborhood of size $(n - 1)^2$.

Two commonly used search techniques were considered: *best improving* and *first improving*. In the *best improving* strategy, the entire neighborhood is analyzed and the current solution is replaced by the best one, while in the *first improving* strategy, the current solution is replaced by the first solution which improves its value. Resende and Ribeiro [28] report that both strategies lead to the same final solution in many applications, but the *first improving* strategy is faster. Furthermore, according to these authors, a premature convergence to a non-global local minimum is more likely to occur with the *best improving* strategy. For these reasons, we selected the *first improving* technique for this implementation.

3.2. GRASP with path relinking

Path relinking was originally proposed by Glover [29] as an intensification strategy that explores paths that connect high-quality solutions (elite solutions) obtained by tabu search. Laguna and Martí [30] first presented the use of path relinking associated with GRASP as an intensification strategy. Other recent successful applications of this association can be found in Canuto et al. [31], Aiex et al. [25] and Festa et al. [32].

According to Resende and Ribeiro [28] the application of path relinking as an intensification procedure in each local optimum seems to be more effective than using it only as a post-optimization step. Therefore, for this implementation, a path is evaluated at each local optimum using an elite solution as an initial solution and the local optimum as the guide solution. This strategy is known as *backward relinking*. This strategy was chosen because, according to Ribeiro et al. [33], if only one path is to be investigated, the best solutions are found when the relinking procedure starts from the best solution. They also observed that exploring two different paths for each pair of solutions takes approximately twice the time required to explore only one of them, with hardly any improvement in the quality of the solution. The elite solution used is randomly selected from the set S, which is stored during the application of GRASP. Every local optimum is considered to be part of S and is included in this set if its objective function value is better than any element of the set.

In order to build the path connecting the initial solution x_i and the guide solution x_g, the symmetric difference between them, given by $\Delta(x_i, x_g)$, must be calculated. In this problem, this difference is defined as the number of jobs that are not at the same absolute position in the initial solution and in the guide solution. First, the procedure examines all possible insertion moves $m \in \Delta(x_i, x_g)$ from the initial solution and selects the lowest cost solution. This move is made and a new intermediate solution (x) is found. If necessary, the best solution x^* is updated. The set of available moves $\Delta(x, x_g)$ is updated and a new iteration of the procedure begins with the current solution x. A similar search strategy is described by Resende and Ribeiro [26].

It should be noted that it is possible to connect x_i to x_g in up to m moves using insertion moves if each move is made in an ordered way. However, if the moves are not ordered, it can be shown that more than m moves could be necessary because one move can corrupt a previous one. The steps below show the proposed path relinking strategy:

Step 1: When a local optimal x_{ls} is reached make $x_g = x_{ls}$.

Step 2: Choose the initial solution x_i from S and let iter $= 1$.

Step 3: Compute $\Delta(x_i, x_g)$ and make iter_max $=\Delta(x_i, x_g)$, where iter_max is the maximum number of iterations

Step 4: Evaluate all moves $m \in \Delta(x_i, x_g)$.

Step 5: Choose the move m with lowest cost solution.

Step 6: Set $x_i = x_i \oplus m$, where $\oplus$ defines the insertion move execution.

Step 7: Let iter $=$ iter $+ 1$.

Step 8: Let $x^* =$ Best_solution(x_i, x^*), where Best_solution() compares two feasible solutions and returns the best one.

Step 9: Compute $\Delta(x_i, x_g)$.

Step 10: If $\Delta(x_i, x_g) > 0$ and iter $\leqslant$ iter_max, go back to Step 4.

4. Numerical experiments

The codes were written in C++ and the tests were conducted on a Pentium IV with a 2.3 GHz processor and 512 Mb RAM. Test problems were generated according to Taillard [34]. These instances are commonly used in the literature (see, e.g. [15,35]). Ten different matrices of processing times were generated for each of the 12 sizes in Table 1. For each of those matrices, four scenarios were built. Processing times were uniformly distributed between 1 and 99. All instances are available at http://www.prd.usp.br/docentes/debora/.

Due dates are uniformly distributed between $P(1-T-R/2)$ and $P(1-T+R/2)$ [36], where T and R are the tardiness factor of jobs and dispersion range of due dates, respectively, while P is a lower bound of the makespan on the flowshop with unlimited buffer [34]

Table 1
Improvement percentage of the FPDNEH compared to LBNEH

Size ($n \times m$)	Scenario				Average
	1	2	3	4	
20×5	18.12	−13.96	0.35	7.64	3.04
20×10	3.26	3.03	0.38	1.69	2.09
20×20	−1.38	−3.50	−0.44	−0.52	−1.46
50×5	24.25	62.70	10.58	2.23	24.94
50×10	7.63	11.78	−0.61	1.86	5.17
50×20	2.76	1.62	−0.80	−2.38	0.30
100×5	39.56	83.94	15.86	−3.48	33.97
100×10	17.49	34.35	4.93	1.19	14.49
100×20	9.56	6.84	1.92	0.64	4.74
200×10	25.85	56.07	10.29	−1.31	22.73
200×20	13.88	18.39	4.04	−0.94	8.84
500×20	19.04	31.86	7.91	−0.79	14.51
Average	15.00	24.43	4.53	0.49	11.11

defined as

$$P = \max\left\{\max_{1\leqslant v\leqslant m}\left[\sum_{k=1}^{n} p_{kv} + \min_{k}\sum_{q=1}^{v-1} p_{kq} + \min_{k}\sum_{q=v+1}^{m} p_{kq}\right], \max_{k}\sum_{v=1}^{m} p_{kv}\right\}. \quad (13)$$

The scenarios represent different configurations by varying T and R, as follows:

- *scenario 1*: low tardiness factor (T=0.2) and small due date range ($R = 0.6$),
- *scenario 2*: low tardiness factor (T=0.2) and wide due date range ($R = 1.2$),
- *scenario 3*: high tardiness factor ($T = 0.4$) and small due date range ($R = 0.6$),
- *scenario 4*: high tardiness factor ($T = 0.4$) and wide due date range ($R = 1.2$).

4.1. Performance of the FPDNEH algorithm

The FPDNEH heuristic is evaluated by the relative percentage improvement with respect to the LBNEH algorithm proposed by Armentano and Ronconi [16,17]. According to the computational tests conducted by the authors, LBNEH is the best constructive procedure to minimize the total tardiness on a flowshop with blocking and unlimited buffer. The improvement is calculated using the following expression:

$$\text{Improvement} = \frac{T_{\text{LBNEH}} - T_{\text{FPDNEH}}}{T_{\text{LBNEH}}} 100,$$

where T_{LBNEH} and T_{FPDNEH} are the values of total tardiness obtained by the LBNEH and FPDNEH algorithm, respectively. When $T_{\text{LBNEH}} = T_{\text{FPDNEH}} = 0$ the improvement obtained by FPDNEH in relation to LBNEH is defined as zero (as in all other cases in which $T_{\text{LBNEH}} = T_{\text{FPDNEH}}$). When $T_{\text{LBNEH}} = 0$ and $T_{\text{FPDNEH}} > 0$, the above expression suggests an improvement of $-\infty$, however, this case did not occur in the presented computational experiments.

First, several tests were conducted to select parameter ρ of expression (10). The performance of FPDNEH was analyzed in terms of average improvement and the number of superior solutions for the values $\{0, 0.1, 0.2 \ldots 1\}$. The FPDNEH heuristic presents its best performance in both measures when the parameter $\rho = 0.3$, i.e., when the *dynslack* term has a higher weight. Table 1 shows the average percentage improvement of each class with 10 problems with the application of the FPDNEH algorithm.

By analyzing Table 1, it can be seen that the FPDNEH algorithm presents better average results in all scenarios, especially in the scenarios with low tardiness factor. The proposed algorithm outperforms the LBNEH algorithm in 75% of the classes and the average improvement was superior to 10% in 35.4% of the classes. Considering the number of superior results, the FPDNEH algorithm outperformed the LBNEH algorithm in 336 of the 480 test-problems and presented the same total tardiness value in eight problems. This behavior is probably due to the fact that, on its first stage, the FPDNEH algorithm considers a higher number of characteristics that are inherent to the problem, such as the blocking situation.

In the largest case, 500 jobs and 20 machines, the FPDNEH heuristic presented a running time of 15.1 s. The CPU time was less than 1 s for the all the remaining problems.

4.2. Performance of GRASP

4.2.1. Basic version

Several tests were conducted to select the components of GRASP in its basic version and the following parameters were chosen. The construction phase is composed by the FPD dispatching rule associated with the RCL constructed by the *value_based_scheme*$_1$ ($p = 0.35$) (see the basic steps in Section 3.1). Before using a local search, the solution is improved by the insertion procedure of the NEH algorithm. The local search utilizes the insertion move and the *first improving* strategy. The algorithm is executed until n local searches are completed or the maximum CPU time of 1800 s is reached.

Table 2
Improvement percentage of the GRASP basic version compared to FPDNEH heuristic

Size ($n \times m$)	Improvement percentage				Improvement by size	CPU time (s)
	Scenario 1	Scenario 2	Scenario 3	Scenario 4		
20 × 5	42.24	27.52	18.29	11.97	25.00	0.10
20 × 10	16.78	19.21	9.76	7.07	13.20	0.19
20 × 20	9.68	9.66	5.91	5.07	7.58	0.31
50 × 5	57.32	97.67	25.87	28.93	52.45	9.48
50 × 10	39.85	61.91	21.22	15.75	34.68	23.71
50 × 20	29.94	27.17	11.67	10.78	19.89	38.11
100 × 5	59.56	68.81	26.21	37.76	48.09	308.84
100 × 10	45.94	87.50	21.84	26.61	45.47	730.46
100 × 20	33.90	69.22	15.51	11.91	32.64	1517.56
200 × 10	40.44	99.28	20.09	21.51	45.33	1705.82
200 × 20	28.76	69.32	14.23	13.57	31.47	1800.02
500 × 20	11.38	36.84	6.09	9.51	15.95	1800.02
Average	34.65	56.18	16.39	16.70	30.98	661.22

Table 3
Improvement percentage of the GRASP with path relinking compared with FPDNEH heuristic

Size ($n \times m$)	Improvement percentage				Improvement by size	CPU time (s)
	Scenario 1	Scenario 2	Scenario 3	Scenario 4		
20 × 5	**44.42**	27.00	**18.80**	**12.08**	**25.57**	0.11
20 × 10	**17.48**	**19.71**	**9.94**	6.76	**13.47**	0.22
20 × 20	9.09	**9.78**	5.81	5.06	7.44	0.34
50 × 5	**60.14**	**98.00**	**25.90**	**29.62**	**53.42**	9.72
50 × 10	**40.15**	**63.19**	**21.38**	**16.79**	**35.37**	24.51
50 × 20	**30.70**	27.16	**12.33**	**10.90**	**20.27**	39.55
100 × 5	**59.95**	68.73	**28.07**	**37.77**	**48.63**	317.87
100 × 10	**46.66**	**88.02**	**22.40**	26.21	**45.82**	755.73
100 × 20	**34.24**	**71.04**	15.16	**12.25**	**33.17**	1576.46
200 × 10	39.75	98.73	19.67	**21.53**	44.92	1676.97
200 × 20	27.81	69.12	14.18	12.97	31.02	1803.66
500 × 20	9.57	32.92	5.45	8.68	14.15	1800.03
Average	**35.00**	56.11	**16.59**	**16.72**	**31.11**	667.10

Table 4
Comparison of the GRASP with path relinking and the basic version

Number of successes of the path relinking	Max (%)	Number of successes of the basic version	Max (%)	Even
213	100.0	180	95.5	87

The GRASP metaheuristic is evaluated by the relative percentage improvement with respect to the FPDNEH algorithm. Table 2 shows the average percentage improvement of each class with 10 problems. An overall average improvement of 30.98% was achieved by GRASP. The *Improvement by size* column also shows that, in most cases, the increase in the number of jobs with the same number of machines benefits the performance of the GRASP algorithm. The main exception is the case of 500 jobs and 20 machines, probably due the short CPU time allowed to run the algorithm. The last column of Table 2 shows the average computational

CPU time of GRASP. Note that the FPDNEH algorithm is faster, although better solutions can be found by GRASP.

4.2.2. Path relinking

Preliminary tests were conducted to select the dimension of S, the number of elite solutions, on the GRASP version with path relinking. Best results were obtained with $|S| = 1$. The improved GRASP strategy was run with the same stopping criterion of the basic version. Table 3 shows the improvement percentage obtained by the association of GRASP and path relinking in comparison with the FPDNEH heuristic. The numbers in bold indicate the classes of problems where this version outperformed the average improvement obtained by the basic version.

Note that the application of this strategy leads to better solutions in 58.3% of the classes and moderately improves the average improvement in scenarios 1, 3 and 4. It can also be observed that better results were achieved in medium-size problems with $n \leqslant 100$, where the running time was not a critical factor. In larger problems, within the maximum allowed running time, this strategy presents a slightly lower performance when compared with the basic version. Although a small overall gain was achieved, it should be stressed that this strategy is easy to implement and has almost the same running time as the basic version.

Table 4 presents an additional analysis. The number of successes is used to evaluate the basic version of GRASP against its association with path relinking for the 480 problems tested. The number of successes is defined as the total number of times that the analyzed strategy provided the best solution. "Max", in columns 2 and 4, represents the largest gain of one version compared with the other. "Even", in the last column, denotes the number of problems in which the best solutions of the versions have the same tardiness values. It can be observed that the application of GRASP with path relinking results in a larger number of successes when compared to the basic version of GRASP.

4.3. Comparison with B&B

To conduct a better evaluation of the proposed heuristics, a comparative study is presented with the optimal solutions provided by the B&B algorithm proposed by Ronconi and Armentano [19]. Given the difficulty in finding optimal solutions for problems with the sizes used in the original set of tests, a new set with smaller sizes was created for the tests. Similar approaches have been considered in Chen and Wu [37], Yamashita et al. [38] and Armentano and Scrich [39].

The B&B algorithm was applied to a sample of five processing time matrices for each size ($n \leqslant 20$ and $m \leqslant 10$) and four scenarios for each matrix, that is, in $27 \times 5 \times 4 = 540$ problems. The degree of suboptimality (tolerance) allowed was 0.5%. Respecting the maximum CPU time allowed, the B&B algorithm achieved the optimal solution in 486 problems.

The proposed methods—FPDNEH algorithm, GRASP basic version and its association with path relinking—were applied to the problems with known optimal solutions. Both GRASP versions were executed until n local searches were completed. The running time for each problem was less than 0.14 s.

In these size-reduced problems, the basic version of GRASP has found optimal values in 287 problems, while the association of GRASP with path relinking has found optimal values in 297 problems. The FPDNEH algorithm has found optimal solutions in 61 problems. In average, the basic version of GRASP has provided solution values that were 4.7% higher than the B&B value, while its association with path relinking was, in average, 4.3% higher than the B&B value.

5. Final remarks

In this paper we studied the minimization of the total tardiness in a flowshop with blocking scheduling. First, we proposed a constructive heuristic called FPDNEH. In a comparison of the FPDNEH against the LBNEH algorithm, known as the best constructive heuristic for this problem, the proposed algorithm presented an average improvement of 11.11% and achieved better results in 336 of the 480 tested problems. This behavior is probably due to the fact that the dispatch rule FPD explores the non-existence of intermediate buffers, as well as the characteristics related to the tardiness criterion.

Then, we developed a GRASP-based heuristic to obtain better solutions within a reasonable period of time. The basic version presented an overall average improvement of 30.98% in comparison with the constructive heuristic, and better results in 475 of the 480 test-problems. Furthermore, the association of GRASP with a path relinking strategy was able to improve the performance of the metaheuristic for some problems, with approximately the same running time. Finally, the comparison with optimal solutions for small problems showed that all versions can yield good results, especially GRASP with path relinking. As a future research effort, it would be interesting to extend the ideas pre-

sented in this paper to the flowshop with blocking, aiming to minimize the total earliness and tardiness cost.

Acknowledgments

The author would like to thank Wieslaw Kubiak and two anonymous referees whose comments helped a lot to improve this paper. This research was funded by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq) and *Fundação de Amparo à Pesquisa do Estado de São Paulo* (FAPESP).

References

[1] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 1996;44:510–25.

[2] Grabowski J, Pempera J. Sequencing of jobs in some production system. European Journal of Operational Research 2000;125:535–50.

[3] Sethi SP, Sriskandarajah C, Sorger G, Blazewicz J, Kubiak W. Sequencing of parts and robot moves in a robotic cell. International Journal of Flexible Manufacturing Systems 1992;4:331–58.

[4] Sen T, Gupta SK. A state-of-art survey of static scheduling research involving due date. Omega 1984;12:63–76.

[5] Pinedo M. Scheduling: theory, algorithms, and systems. New Jersey: Prentice-Hall; 1995.

[6] Du J, Leung JY-T. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 1990;15: 483–95.

[7] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.

[8] Leisten R. Die Einbeziehung beschränkter Zwischenlager in die Auftragsreihenfolgeplanung bei Reihenfertigung. Düsseldorf: VDI-Verlag; 1985.

[9] McCormick ST, Pinedo ML, Shenker S, Wolf B. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 1989;37:925–36.

[10] Leisten R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 1990;28:2085–100.

[11] Nawaz M, Enscore EE, Ham I. A heuristic algorithm for the M-machine, N-job flow-shop sequencing problem. Omega 1983;11:91–5.

[12] Ronconi DP. A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics 2004;87:39–48.

[13] Abadi INK, Hall NG, Sriskandarajah C. Minimizing cycle time in a blocking flowshop. Operations Research 2000;48:177–80.

[14] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. Minimizing makespan in a flowshop using genetic algorithms. International Journal of Production Economics 2001;70:101–15.

[15] Grabowski J, Pempera J. The permutation flow shop problem with blocking. A tabu search approach. Omega 2007;35: 302–11.

[16] Ronconi DP. A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Annals of Operations Research 2005;138:53–65.

[17] Armentano VA, Ronconi DP. Minimização do Tempo Total de Atraso no Problema de Flowshop com Buffer Zero através de Busca Tabu. Gestão & Produção 2000;7:352–63.

[18] Armentano VA, Ronconi DP. Tabu search for total tardiness minimization in flowshop scheduling problems. Computers and Operations Research 1999;26:219–35.

[19] Ronconi DP, Armentano VA. Lower bounding schemes for flowshops with blocking in-process. Journal of the Operational Research Society 2001;52:1289–97.

[20] Kalczynski PJ, Kamburowski J. On the NEH heuristic for minimizing the makespan in permutation flow shops. Omega 2007;35:53–60.

[21] Feo T, Resende MGC. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 1989;8:67–71.

[22] Feo T, Resende MGC. Greedy randomized adaptative search procedures. Journal of Global Optimization 1995;6:109–33.

[23] Rios-Mercado RZ, Bard JF. Heuristics for the flow line problem with setup costs. European Journal of Operational Research 1998;110:76–98.

[24] Akturk MS, Ozdemir D. A new dominance rule to minimize total weighted tardiness with unequal release dates. European Journal of Operational Research 2001;135:394–412.

[25] Aiex RM, Binato S, Resende MGC. Parallel grasp with path-relinking for job shop scheduling. Parallel Computing 2003;29:393–430.

[26] Resende MGC, Ribeiro CC. Grasp with path-relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M, editors. Metaheuristics: progress as real problem solvers. Berlin: Springer; 2005. p. 29–63.

[27] Pitsoulis LS, Resende MGC. Greedy randomized adaptative search procedures. In: Pardalos PM, Resende MGC, editors. Handbook of applied optimization. Oxford: Oxford University Press; 2002. p. 168–83.

[28] Resende MGC, Ribeiro CC. Greedy randomized adaptative search procedures. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. Drodrecht: Kluwer Academic Publishers; 2003. p. 219–49.

[29] Glover F. Tabu search and adaptative memory programing—advances, applications and challenges. In: Barr R, Helgason R, Kennington J, editors. Interfaces in computer science and operations research. Dordrecht: Kluwer Academic Publishers; 1996. p. 1–75.

[30] Laguna M, Martí R. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 1999;11:44–52.

[31] Canuto SA, Resende MGC, Ribeiro CC. Local search with perturbations for the prize-collecting steiner tree problem in graphs. Networks 2001;38:50–8.

[32] Festa P, Pardalos PM, Pitsoulis LS, Resende MGC. Grasp with path-relinking for the weighted maximum satisfiability problem. In: Proceedings of IV workshop on efficient and experimental algorithms (WEA2005), Lecture notes in computer science, vol. 3503, 2005. p. 367–379.

[33] Ribeiro CC, Uchoa E, Werneck RF. A hybrid grasp with pertubations for the steiner problem in graphs. INFORMS Journal on Computing 2002;14:228–46.

[34] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.

[35] Li X, Wang Q, Wu C. Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega 2006; doi: 10.1016/j.omega.2006.11.003.

[36] Potts CN, Van Wassenhove LN. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters 1982;1:177–81.

[37] Chen J-F, Wu T-H. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. Omega 2006;34:81–9.

[38] Yamashita DS, Armentano VA, Laguna M. Scatter search for project scheduling with resource availability cost. European Journal of Operational Research 2006;169:623–37.

[39] Armentano VA, Scrich CR. Tabu search for minimizing total tardiness in a job shop. International Journal of Production Economics 2000;63:131–40.