Engineering Optimization

ISSN: 0305-215X (Print) 1029-0273 (Online) Journal homepage: https://www.tandfonline.com/loi/geno20

High-performing heuristics to minimize flowtime in no-idle permutation flowshop

Marcelo Seido Nagano, Fernando Luis Rossi & Nádia Junqueira Martarelli

ENGINEERING OPTIMIZATION
2019, VOL. 51, NO. 2, 185–198
https://doi.org/10.1080/0305215X.2018.1444163

High-performing heuristics to minimize flowtime in no-idle permutation flowshop

Marcelo Seido Nagano [a], Fernando Luis Rossi[b] and Nádia Junqueira Martarelli[a]

[a]Department of Industrial Engineering, São Carlos School of Engineering, University of São Paulo, Brazil; [b]Management Department, Federal Institute of São Paulo, Brazil

ABSTRACT
In this article, the issue of production scheduling in a no-idle flowshop environment is addressed. An extensive literature review has shown that there are no heuristics specifically proposed for this problem, especially when it comes to constructive heuristic methods. In this context, this article proposes a highly efficient simple constructive heuristic to minimize the total flowtime criterion. The proposed heuristic was embedded in the high performance iterated greedy algorithm. Computational results and statistical analysis show that the proposed heuristic overperformed the main constructive methods found up to now. In addition, it is observed that the integration of the proposed heuristic with the iterated greedy algorithm provides the most efficient metaheuristic for the problem.

ARTICLE HISTORY
Received 21 August 2017
Accepted 17 February 2018

KEYWORDS
Scheduling; no-idle flowshop; heuristics; total flowtime

1. Introduction

Production environments, which are conditioned to high setup times or upraised operation costs, do not allow the existence of idle machines after a job sequence begins. There are also cases in which the no-idle restriction appears due to technological restrictions, for instance the production of integrated circuits by means of photolithography or in ceramic production (Pan and Ruiz 2014), fibreglass processing (Kalczynski and Kamburowski 2005) and foundry operations (Saadani, Guinet, and Moalla 2005).

In this sense, research has been done to find a good production sequence which results in the minimization or maximization of an objective function (Pan and Ruiz 2014). One of these is the Total FlowTime minimization (TFT), which is a significant criterion in current production environments due to the fact that it is directly related to the minimization of an in-process inventory (Liu and Reeves 2001).

The no-idle flowshop with a TFT criterion is technically denoted as $F_m|no-idle|\sum C_j$ (Graham *et al.* 1979), in which F_m means the number of machines in flowshop, in this case, m; no-idle is a restriction that does not allow the machines to be stopped after beginning the jobs and the $\sum C_j$ means the sum of job j's completion time. A sequence (π) of n jobs that must be processed by a set of m different machines is also defined, in the same order. A job j in a given sequence, in the position k, may be denoted by π_k. The processing time of jobs j in the machine i is represented by $p_{i,j}$, where $i = \{1, 2, \dots, m\}$ and $j = \{1, 2, \dots, n\}$. Let $C_{i,j}$ be the completion time of job j in machine i, the TFT criterion is the sum of the last machine job completion times, $\sum_{j=1}^{n} C_{m,j}$.

CONTACT Marcelo Seido Nagano drnagano@usp.br

Supplemental data for this article can be accessed at https://doi.org/10.1080/0305215X.2018.1444163

Although the afore-stated issue is of practical importance, it has not been attracting attention in the literature, whereas most studies currently aim to minimize the makespan (total time from start to finish) $F_m|no-idle|C_{max}$ or the total tardiness $Fm|no-idle|\sum T_j$, as described in the work by Rad, Ruiz, and Boroojerdian (2009), Tasgetiren *et al.* (2011), Deng and Gu (2012), Pan and Ruiz (2014), Zhou, Chen, and Zhou (2014), Nagano, Rossi, and Tomazella (2017), and Shao, Pi, and Shao (2017).

In addition, the no-idle problem has attracted attention in some other problem variations. Lu (2016) considers no-idle flowshop scheduling with a time-dependent learning effect and deteriorating jobs with makespan and total flowtime minimization. Ying *et al.* (2017) proposes an Iterated Reference Greedy (IRG) algorithm for the distributed no-idle permutation flowshop scheduling problem with a makespan objective.

One of the first no-idle studies has been proposed by Adiri and Pohoryles (1982), who developed a polynomial-time algorithm for $F_2|prmu, no-idle|\sum C_j$, where $prmu$ means that the same job sequence is maintained in each machine. Observing some limitations in the work by Adiri and Pohoryles (1982), Čepek, Okada, and Vlach (2000) improved it.

Recently, Narain and Bagga (2005) contributed to this area by proposing a branch and bound method for the most general problem, $F_m|no-idle|\sum C_j$. However, only small cases could be efficiently and optimally solved since it is an NP-complete problem, to date.

This problem type requires an extreme central processing unit (CPU) time if resulted by exact algorithms, even for moderate-scale problems. Therefore, metaheuristics, or heuristics, are used to seek optimal or near-optimal solutions in a reasonable time, especially for large-scale production problems in industry.

Regarding metaheuristics, Tasgetiren *et al.* (2013) proposed an approach called Iterated Greedy (IG), with Differential Evolution (vIG DE), to solve the $F_m|no-idle|C_{max}$ and $F_m|no-idle|\sum C_j$ problems. The proposed algorithm outperformed the metaheuristics from Pan and Wang (2008a), Pan and Wang (2008b), and Deng and Gu (2012).

Framinan, Gupta, and Leisten (2004) divides a heuristic into three steps, namely index development, solution construction, and solution improvement. A heuristic may use one or all of these phases. From this concept, Framinan, Leisten, and Ruiz-Usano (2005) classifies a heuristic as simple and composite. Composite heuristics use a simple heuristic in one of the three above mentioned phases. A simple heuristic does not use another heuristic within any of the three phases.

There is another way to classify a heuristic. It is known that a constructive heuristic obtains a solution directly and uses a procedure to construct the solution sequence for each job by priority index. It is also known that an improvement heuristic works with a provided initial solution, making this kind of method search for a better solution, for example, using neighbourhood search procedures (Nagano and Moccellin 2002).

Even with these advances, no constructive heuristics have been proposed for the problem so far, which is noteworthy, as heuristics are widely applied to other problems to obtain high-quality solutions in large size problems in short times (see Ponnambalam, Aravindan, and Chandrasekaran 2001; Liu and Reeves 2001; Nagano and Moccellin 2002; Framinan, Leisten, and Ruiz-Usano 2002; Framinan and Leisten 2003; Kalczynski and Kamburowski 2007; Li, Wang, and Wu 2008; Kalczynski and Kamburowski 2008; Dong, Huang, and Chen 2008; Rad, Ruiz, and Boroojerdian 2009; Kalczynski and Kamburowski 2009; Laha and Sarin 2009; Ribas, Companys, and Tort-Martorell 2010; Kalczynski and Kamburowski 2011; Pan and Ruiz 2013; Fernandez-Viagas and Framinan 2014 2015; Benavides and Ritt 2016; Zheng *et al.* 2016; Fernandez-Viagas, Leisten, and Framinan 2016; Rossi, Nagano, and Tavares Neto 2016; Liu, Jin, and Price 2016; Rossi, Nagano, and Sagawa 2017; Ribas, Companys, and Tort-Martorell 2017; Liu, Jin, and Price 2017; Huang *et al.* 2017).

Aware of the area gap, this article proposes a new simple constructive heuristic for no-idle flowshop environment $F_m|no-idle|\sum C_j$, besides an adaptation from some main simple heuristics identified in the literature, which were found from an extensive literature review. This search was conducted to identify the best simple heuristic in term of performance proposed to the $F_m|prmu|\sum C_j$

problem. Aiming for a deeper contribution, a modification in the vIG DE algorithm from Tasgetiren *et al.* (2013) has also been proposed using the proposed heuristic.

This article is organized as follows. Section 2 presents the main simple heuristics for the $F_m|prmu|\sum C_j$ problem, selected to be adapted to the $F_m|no-idle|\sum C_j$ problem. Section 3 shows a new simple heuristic method. Section 4 brings the computational results and statistical analysis. Finally, Section 5 displays the conclusions.

2. Simple heuristics

Some main simple heuristics available for the problem $F_m|prmu|\sum C_j$ are presented. They were extracted from works by Pan and Ruiz (2013) and Fernandez-Viagas and Framinan (2015), who conducted an extensive comparison between simple and composite methods for minimizing total flowtime.

Woo and Yim (1998) developed a simple heuristic, called WY, which did not apply an initial order in the first step but allows jobs that were not scheduled to be inserted in every possible position of the partial sequence, selecting the sequence with the best total flowtime.

The simple heuristic LR(x), developed by Liu and Reeves (2001), initially orders jobs j according to the index $\xi_{j,k}(k=0)$, which considers the machines idle time and the effects of job insertion in the conclusion times of the subsequent jobs, resulting in a jobs list $\alpha = \alpha_1, \ldots, \alpha_n$. The lowest value job $\xi_{j,k}(k=0)$ is placed in the first position of the sequence and the value of $\xi_{j,k}(k=0)$ (Equation (1)) is calculated for each unscheduled (U) job j, where U is the unscheduled jobs set. The lowest value job, $\xi_{i,k}(k=0)$, is placed in the last position of the sequence. It ends when every job has been sequenced.

The LR(x) heuristic generates x different sequences. It follows the job α_2, instead of the α_1 job, in the first sequence position as far as x sequences are generated:

$$\xi_{j,k} = (n-k-2) * IT_{j,k} + AT_{j,k} \tag{1}$$

The $IT_{j,k}$ and $AT_{j,k}$ values are obtained by

$$IT_{j,k} = \sum_{i=2}^{m} \left(\frac{m * \max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i + \frac{k*(m-i)}{n-2}} \right) \tag{2}$$

and

$$AT_{j,k} = C_{m,j} + C_{m,u} \tag{3}$$

respectively.

As the no-idle issue does not allow idleness in the machines, Equation (2) results in $\max\{C_{i-1} - C_{i,[k]}, 0\} = 0$, consequently $IT_{j,k} = 0$, transforming Equation (1) into

$$\xi_{j,k} = AT_{j,k} = C_{m,j} + C_{m,u} \tag{4}$$

where $C_{m,j}$ means the completion time of job j to be added in the last position of the current sequence and $C_{m,u}$ is the completion time of the artificial job u in the last machine. The artificial job is defined as an average calculated by the processing time of all the jobs in $U - \{j\}$ on each machine, where j is the job added in the last position. Therefore, this average is considered the artificial processing time from the artificial job.

The artificial times p'_i are calculated as follows:

$$p'_i = \frac{\sum_{k=1}^{U-\{j\}} p_{i,k}}{|U - \{j\}|} \qquad (i = 1, 2, \ldots, m) \tag{5}$$

The relationship illustration between the completion times, $C_{m,j}$ and $C_{m,u}$, is in Figure 1.

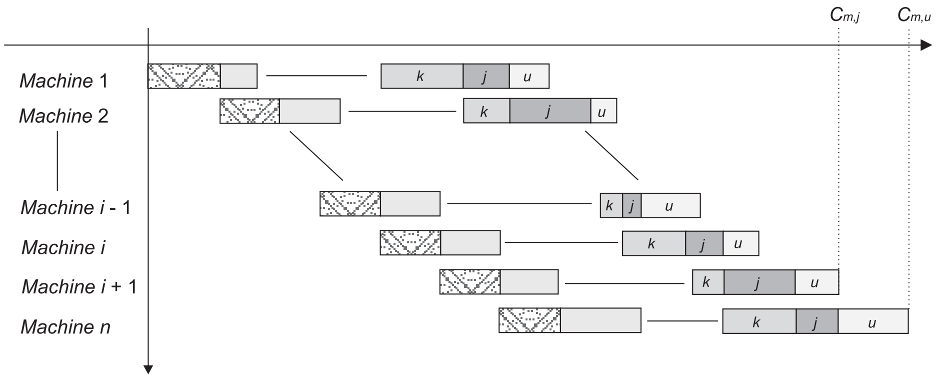

Figure 1. Relationship between completion times $C_{m,j}$ and $C_{m,u}$.

Algorithm 1 LR-NEH(x) heuristic. Adapted from Pan and Ruiz (2013)

```
Calculate the values of $\xi_{j,0}$ (Equation (4))
Order jobs in non-descending order of $\xi_{j,0}$, obtaining $\alpha = (\alpha_1, \dots, \alpha_n)$
$d = \frac{3n}{4}$
for $l = 1$ to $x$ do
    $\pi^l = \alpha_l$
    $U = U - \alpha_l$
    for $k = 2$ to $d$ do
        Select the job $j$ with the lowest $\xi_{j,k} \in U$ value. Place it at the end of $\pi^l$
        $U = U - j$
    end for
    Calculate $P_j = \sum_{i=1}^{m} p_{i,j} \qquad \forall j \in U$
    Order the jobs in non-descending order of $P_j$, obtaining $\beta = \beta_1, \dots, \beta_{n-d}$
    for $k = 1$ to $n - d$ do
        Test job $\beta_k$ in all positions of $\pi^l$
        Insert job $\beta_k$, searching for a position which minimizes the $\sum C_i$ value
    end for
end for
return sequence $\pi \in \{\pi^1, \pi^2, \dots, \pi^x\}$ with the lowest value of $\sum C_i$
```

The FL-LS heuristic, developed by Laha and Sarin (2009), is a modification of the Framinan and Leisten (2003) heuristic. It uses an iterative job insertion, similar to the NEH Nawaz, Enscore, and Ham (1983) heuristic, where the job, after it is inserted in the best possible position, is reinserted in the current sequence. The authors proved that this modification improves the performance of the Framinan and Leisten (2003) heuristic, significantly. The FL-LS heuristic also outperformed the WY method.

LR-NEH(x), developed by Pan and Ruiz (2013), minimizes the total flowtime problem, using the LR(x) heuristic to generate a partial sequence of d jobs and the NEH heuristic to insert $n{-}d$ jobs in the partial sequence. The relative position of the jobs does not change while the sequence is being constructed.

The authors affirm that the FL-LS heuristic, from Laha and Sarin (2009), is the best regarding solution quality but not in terms of CPU time. They concluded that the LR-NEH(x) heuristic offers a good balance between CPU time and solution quality. LR-NEH(x) pseudocode is presented in Algorithm 1.

FF(x), developed by Fernandez-Viagas and Framinan (2015), is similar to LR(x), with regards to addition jobs. The jobs are inserted one by one at the end of the sequence according to the index $\xi'_{j,k}$, which is based on the inserted job idle time and the completion time. This method proved to be more efficient than LR(x). Fernandez-Viagas and Framinan (2015) also integrated the procedure to several other heuristics, replacing the LR(x) with the new proposal of jobs prioritization. The index $\xi'_{j,k}$ is calculated as follows:

$$\xi'_{j,k} = \left(\frac{n-k-2}{\alpha}\right) * IT'_{j,k} + AT'_{j,k} \tag{6}$$

The values of $AT'_{j,k}$ and $IT'_{j,k}$ are calculated in

$$AT'_{j,k} = C_{m,j} \tag{7}$$

and

$$IT'_{j,k} = \sum_{i=2}^{m} \left(\frac{m * \max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i - b + k * \left(\frac{m-i+b}{n-2}\right)} \right) \tag{8}$$

where IT' is the weighted total machine idle-time between the processing of the kth position job of the sequence and job j, and AT' is the job j completion time in the last machine.

Afresh, $IT'_{j,k} = 0$. Hence, only the element $AT'_{j,k}$ will be presented in index $\xi'_{j,k}$, resulting in

$$\xi'_{j,k} = C_{m,j} \tag{9}$$

The index in Equation (9) chooses the shortest completion time job j in the last machine m. Therefore, the difference between Equations (4) and (9) is the artificial job, which is not considered in Equation (9).

3. A new simple heuristic method

The proposed heuristic, namely Proposed(x,y), was based on the LR-NEH(x) heuristic from the work by Pan and Ruiz (2013), where x and y are the number of generated sequences by the heuristic and the maximum number of reinsertions to be performed, respectively.

To solve the $F_m|no-idle|\sum C_j$ problem, a partial solution is generated with d jobs using LR(x) and the remaining $n-d$ jobs are inserted in the sequence using a NEH heuristic. The proposal in this article uses a modified version of the FL-LS heuristic instead of the NEH method. Other changes were made, such as the reinsertion of y selected jobs rather than reinserting all of them in the sequence after each iteration of NEH. The aim is to optimize the partial sequences generated by NEH efficiently. While reinsertion improves the partial sequences, which results in better final solutions, the selection of a limited number of jobs for reinsertion ensures the heuristic maintains its efficiency.

The jobs are selected in line with a selection procedure, which is based on two main components. The first consists of defining a weight function that prioritizes jobs in ascending order regarding the index W_j,

$$W_j = TFT(\pi - \{j\}) \qquad \text{for } j = 1, \ldots, k \tag{10}$$

where j is the job removed from a sequence π with k jobs. TFT is the sum of the last machine job completion times, $\sum_{j=1}^{n} C_{mj}$. The job to be prioritized for reinsertion is the one that results in the lowest total flow time when removed from the current sequence. Thus, the insertion procedure avoids jobs that have little relevance in the total flowtime solution.

The second main component is comprised of the selection procedure to prevent the jobs that were already selected in previous reinsertion iterations to be selected again. This is necessary because the method only selects the job by the W_j index, therefore, the reinsertion neighbourhood will be similar since the reinsertion will select similar job sets.

Algorithm 2 Proposed(x,y) heuristic

```
Calculate the values of $\xi_{j,0}$, (Equation 4)
Order the jobs in non-descending order of $\xi_{j,0}$, obtaining $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$
Calculate $P_j = \sum_{i=1}^{m} p_{i,j} \quad \forall j, \in U$
Order the jobs in non-descending order of $P_j$, obtaining $\beta = \beta_1, \ldots, \beta_n$
for $l$=1 to $x$ do
    $\pi^l = \{\alpha_l\}$
    $U = U - \{\alpha_l\}$
    for $k$=2 to $d$ do
        Select the job $j$ with the $\xi_{j,k}(lowestvalue) \in U$. Place it at the end of $\pi^l$
        $U = U - j$
    end for
    Order the jobs in $U$ according to the list $\beta$, obtaining $\delta = \{\delta_1, \delta_2, \ldots, \delta_{n-d}\}$
    $R = \emptyset$
    for $k$=1 to $n - d$ do
        Insert job $\delta_k$ in $\pi^l$ in the position that results in the lowest value of $\sum C_j$
        $L$ = job selection ($\pi^l$, $R$, $L$, $k$) (Algorithm 3)
        for $q$=1 to min$\{(n - d) + k, y\}$ do
            $\pi = \pi^l$
            Reinsert the job $L_q$, searching a position in $\pi$ which minimizes the $\sum C_j$ value
            Insert $L_q$ in the front of list $R$. If $|R| \geq |A|$, remove the last job of $R$
            if TFT($\pi$) < TFT($\pi^l$) then
                $\pi^l = \pi$
            end if
        end for
    end for
end for
return the sequence $\pi^l \in \{\pi^1, \pi^2, \ldots, \pi^x\}$ with the lowest value of $\sum C_j$
```

Consider L as a given reinsertion list, $|L|$ is the size of the list L, R is the list of jobs that are not allowed to be reinserted, A is the limit for the number of jobs in R and y is a heuristic parameter which defines the maximum number of reinsertions to be performed.

A list L is generated using the index W_j. In the reinsertion step, the jobs from L are placed in the list R, from the top to the bottom. If the list size is greater than A, the jobs of the last positions of R are removed until R has only A jobs. The list L will be composed by the jobs in the current sequence that are not listed in R, based on the ascending order of the W_j index. When $|L|$ is smaller than y ,the remaining positions of the list L are filled by the following index W_j, provided that there are no repeated jobs in the list L. An example for the proposed heuristic is provided in the online supplementary material.

In the next section, computational experiments and the statistical analysis is presented. Besides, a comparison is done regarding solution quality and computational effort. The Proposed(x,y) heuristic pseudocode is shown in Algorithm 2, as the selection procedure pseudocode is illustrated in Algorithm 3.

4. Computational and statistical experiments

All methods were run in C++, in an i7 4770 3.9 GHz processor, with 8 GB RAM. To test the proposed heuristics, the benchmark of no-idle permutation flowshop problem instances has been

Algorithm 3 Job selection procedure for reinsertion.

```
L=∅
Calculate $W_j = TFT(\pi^l - j), \quad \forall j \in \pi^l$
Order $W_j$ in ascending order, obtaining $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{(n-d)+k)}\}$
$u$=1
while $|L| \leq \min\{(n-d)+k, y)\}$ and $u \leq \{(n-d)+k\}$ do
    if ($\gamma_u$ is not in $R$)
        $L = L + \gamma_u$
    end if
    $u = u + 1$
end while
$u = 1$
while $|L| \leq \min\{(n-d)+k, y\}$
    if ($\gamma_u$ is not in $L$)
        $L = L + \gamma_u$
    end if
    $u = u + 1$
end while
return list $L = \{L_1, L_2, \wr \ldots, L_{\min\{(n-d)+k,y\}}\}$
```

used, proposed by Ruiz, Vallada, and Fernández-Martínez (2009). This test contains 250 instances with combinations of $n = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500$ and $m = 10, 20, 30, 40, 50$. To accelerate the procedures, a method developed by Tasgetiren *et al.* (2013) was applied.

Some necessary adaptations were done for statistical and computational evaluation, such as changing the heuristics for the flowshop scheduling problem followed by the minimization total flowtime criterion and the no-idle restriction. The IG algorithm (vIG DE) has also been compared with the work by Tasgetiren *et al.* (2013) with the variation proposed in this article. The compared heuristics, authors, and implemented adaptations are presented in Table 1.

The Proposed(x,y) heuristic was evaluated regarding different internal parameters, that are: d, which defines the size of the partial sequence resulting from the application of the method LR(x), and A, which defines the size of the jobs list which is not allowed for reinsertion, R.

To find a good combination of the d and A parameters, computational experiments with $d = (\frac{3n}{4}, \frac{n}{2}, \frac{n}{4})$ and $A = (\frac{3n}{4}, \frac{n}{2}, \frac{n}{4})$ were performed, considering $x = 5$ and $y = 15$, that is, Proposed(5,15).

The parameter values were obtained experimentally. Only the best ones were selected. Average relative percentage deviation (ARPD) was used to measure the heuristics performance, as well as to

Table 1. Heuristics compared, authors and adaptations.

Heuristic	Author	Adaptation
NEH	Nawaz, Enscore, and Ham (1983)	Objective function Initial order changed from the longest processing time (LPT) to the shortest processing time (SPT)
WY	Woo and Yim (1998)	Objective function
LR(x)	Liu and Reeves (2001)	Objective function Modification of the priority index $\xi_{j,k}$ (Equation (4))
FL-LS	Laha and Sarin (2009)	Objective function.
LR-NEH(x)	Pan and Ruiz (2013)	Objective function Modification of the priority index $\xi_{j,k}$ (Equation (4))
FF(x)	Fernandez-Viagas and Framinan (2015)	Objective function Modification of the priority index $\xi'_{j,k}$ (Equation (9)).
vIG DE	Tasgetiren *et al.* (2013)	No adaptation required

compare the heuristics regarding solution quality,

$$ARPD(TFT(\pi^h)) = \frac{1}{N}\sum\left(\frac{TFT(\pi^h) - TFT^*}{TFT^*} * 100\right) \tag{11}$$

where $TFT(\pi^h)$ is the total flowtime of the π^h sequence, generated by heuristic h, and TFT^* is the best solution found among all the heuristics compared.

Complete results of parameter variation, ARPD, and CPU time are shown in the online supplementary material. Figure 2 summarizes these results graphically. The best solution is highlighted in grey, while the others are in black.

Heuristic performance is affected significantly when d assumes the values $\frac{3n}{4}$ and $\frac{n}{2}$, since the best result is obtained by $d = \frac{3n}{4}$ and the worst is assigned to $d = \frac{n}{2}$. The d values also interfere in the CPU time, as small d values are related to the greatest CPU time. This shows a benefit of using the LR method to construct part of the sequence ($n \times d$ jobs), because fewer steps are needed in the main loop and the Proposed(x,y) gains speed.

The best results were found for $d = \frac{3n}{4}$ and $A = \frac{n}{4}$. Thus, this combination was adopted in the subsequent experiments.

The x and y variables are defined for every heuristic: (i) LR(x) and FF(x) assume $x = \left\{\frac{n}{10}, n\right\}$; (ii) LR-NEH($x$) with $x = 5, 10, 15$; (iii) Proposed(x,y) is tested with $x = 5, 10$ and $y = 15, 20$.

The results of simple heuristics evaluation considering all problems are presented in Table 2. Complete results are available in the online supplementary material.

The FF($\frac{n}{10}$) heuristic showed worse (ARPD = 22.73) than the LR($\frac{n}{10}$) (22.08), meaning that the artificial job considered in the index $\xi_{j,k}$ (Equation (4)) contributed to better results. These heuristics

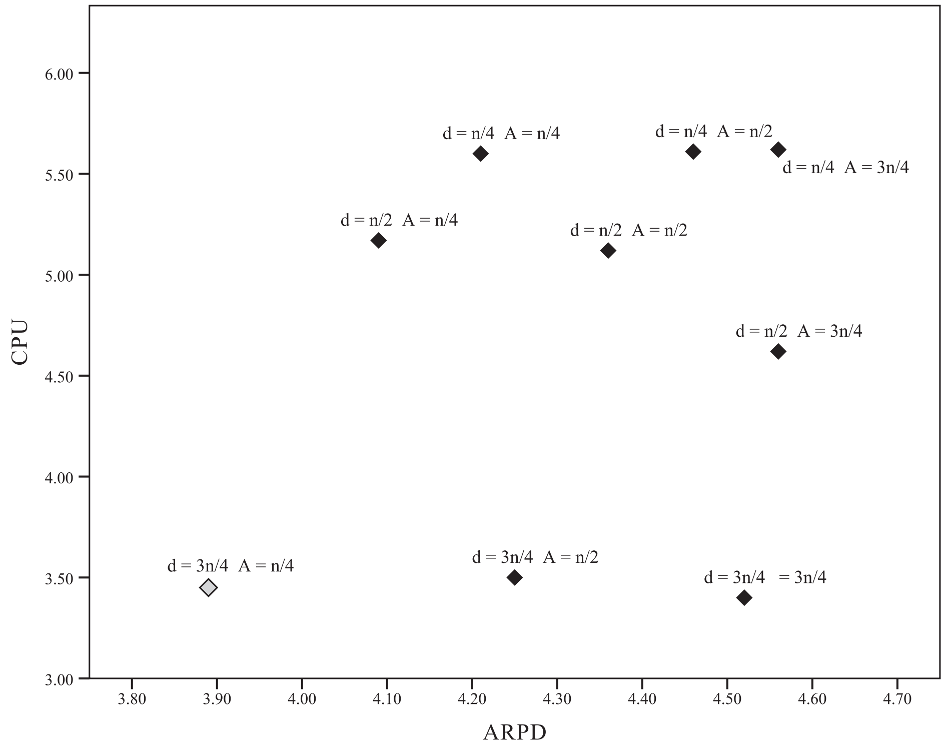

Figure 2. ARPD and CPU time average for the tested parameters combinations. The Pareto dominating combination is depicted in grey.

Table 2. ARPD and the average CPU time, in seconds, of each heuristic.

Heuristics	ARPD	CPU time
Proposed(10, 20)	3.27	7.58
Proposed(10, 15)	3.53	6.13
Proposed(5, 20)	3.63	3.65
Proposed(5, 15)	3.89	3.45
FL-LS	4.76	14.60
LR-NEH(15)	7.37	3.17
WY	7.43	8.07
LR-NEH(10)	7.55	2.13
LR-NEH(5)	7.95	1.07
NEH	10.84	0.07
LR(n)	17.30	80.08
LR($\frac{n}{10}$)	19.38	7.91
FF(n)	22.08	7.03
FF($\frac{n}{10}$)	23.73	0.69

presented results below the other simple heuristics (presented ahead). This was expected since a robust construction phase was lacking in these methods.

The NEH heuristic presented computational efficiency (CPU time average of 0.07 seconds), achieving higher results than the heuristics LR($\frac{n}{10}$) (7.91 seconds) and FF($\frac{n}{10}$) (0.69 seconds). Regarding solution quality, NEH is also better than the other ones. It was observed that increasing the number of generated sequences in FF(n) and LR(n) improves the results considerably ($ARPD_{FF(n)} = 19.38$, $ARPD_{LR(n)} = 17.30$), although its CPU time is tenfold.

The LR-NEH(x) heuristic presented a good balance between solution quality and computational effort, overcoming the heuristics analysed so far.

The WY heuristic reached comparable results to the LR-NEH(10) heuristic, with a mean CPU time greater than the LR-NEH (x) heuristic. The FL-LS heuristic obtained an excellent ARPD (4.76), although its CPU time was the worst so far (14.60 seconds).

The Proposed(10,20) heuristic reached the best result regarding ARPD (3.27). The Proposed(x,y) variations ($x = \{5, 10\}$ and $y = \{15, 20\}$) provided the smallest ARPD. Besides, the Proposed(10,20) heuristic proved effective, with a mean CPU time 48% lower than the FL-LS heuristic. Proposed(5,20) presented superior results to the FL-LS heuristic in terms of mean CPU time, that means, the first took 75% less time than the other one.

The lower average of the CPU time results from the Proposed(x,y), with the selection of a limited set of jobs to be reinserted. While FL-LS selects all the jobs in the sequence for reinsertion, resulting in a very costly computational procedure, the Proposed(x,y) intelligently selects only $y = \{15, 20\}$ jobs from the sequence, resulting in better partial solutions while it keeps its computational efficiency. Figure 3 presents the intervals of the ARPD of each main simple heuristic compared with a confidence interval of 95%.

The LR-NEH(15) heuristic did not present a significant difference when compared to the WY heuristic, despite its CPU time being significantly smaller (see Figure 3). The Proposed(10,20) heuristic presents a significant statistical difference compared to the FL-LS heuristic, with much better results than all the other simple heuristics assessed. The Tukey test was carried out to determine if the averages are statistically different among them and this is provided in the online supplementary material. The results indicate that there is a statistical difference between the Proposed(x; y) heuristic and the other constructive heuristics.

4.1. Evaluation of the vIG DE algorithms

The algorithm vIG DE from the work by Tasgetiren *et al.* (2013), is the best metaheuristic for the no-idle flowshop problem with flowtime criteria so far. It is a variable of the IG algorithm, in which

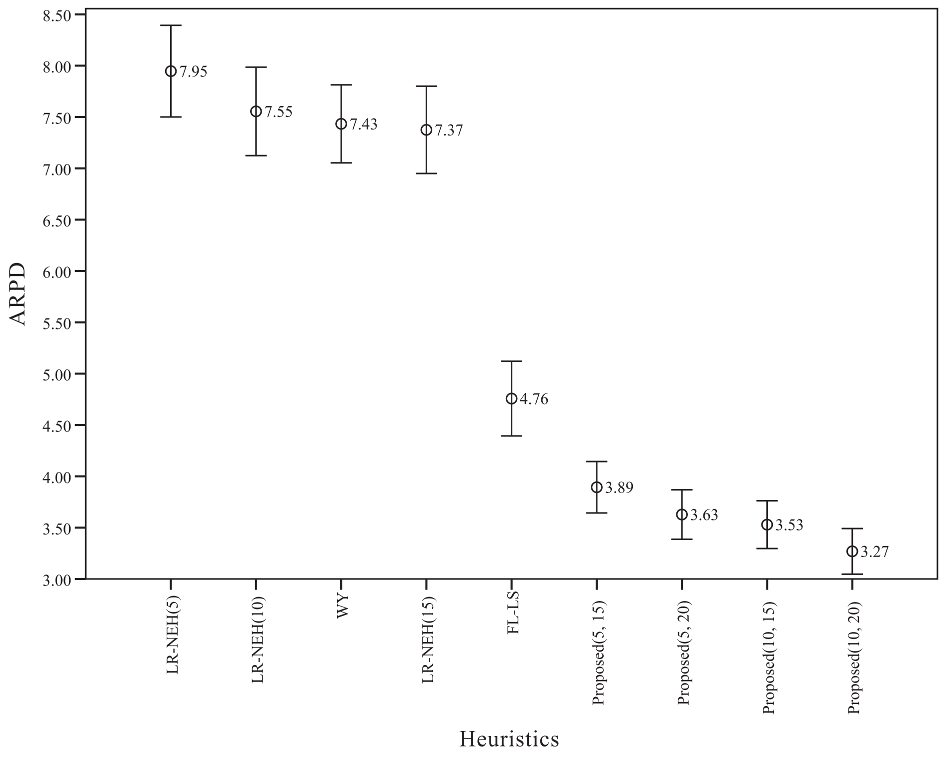

Figure 3. ARPD for each simple heuristic, with a confidence interval of 95%.

the parameters are adjusted by the differential evolution (DE) algorithm (the destruction size and the probability of applying the IG algorithm to an individual).

The vIG DE algorithm starts with an initial population generated by the NEH heuristic, where each individual in the target population is represented by a sequence π_i, a destruction size (d_i) and the probability (p_i) of applying the IG algorithm. The algorithm applies an iterative improvement scheme for each individual consisting of the following procedures:

- Mutation and crossover: A trial individual is generated by applying a mutation and arithmetic crossover operator, with the size of destruction (d_i) and probability (p_i) parameters. The parameters of the generated trial individual are denoted as $u_{i,1}$, for the destruction size, and $u_{i,2}$, for the probability of applying the IG algorithm.
- The probability of applying the IG algorithm: A uniform random number r is generated. If r is less than the probability ($u_{i,2}$), the trial individual is directly produced by applying the IG algorithm.
- Destruction phase: $u_{i,1}$ jobs are randomly removed from sequence π_i without repetition. The sequence without these jobs is denoted by π_D.
- Construction phase: The $u_{i,1}$ jobs removed are inserted one by one in the sequence π_D following the same procedure as in the insertion phase of the NEH heuristic.
- Local search: The solution generated in the construction phase is improved by an improved version of the local search proposed by Pan, Tasgetiren, and Liang (2008).
- Selection: The selection is applied based on the survival of the fittest among the trial and target individuals.

The heuristic Proposed(x, y) was utilized to generate the initial population for the vIG DE, instead of using NEH. Thus, in this new algorithm, the sequence for the first individual is constructed by

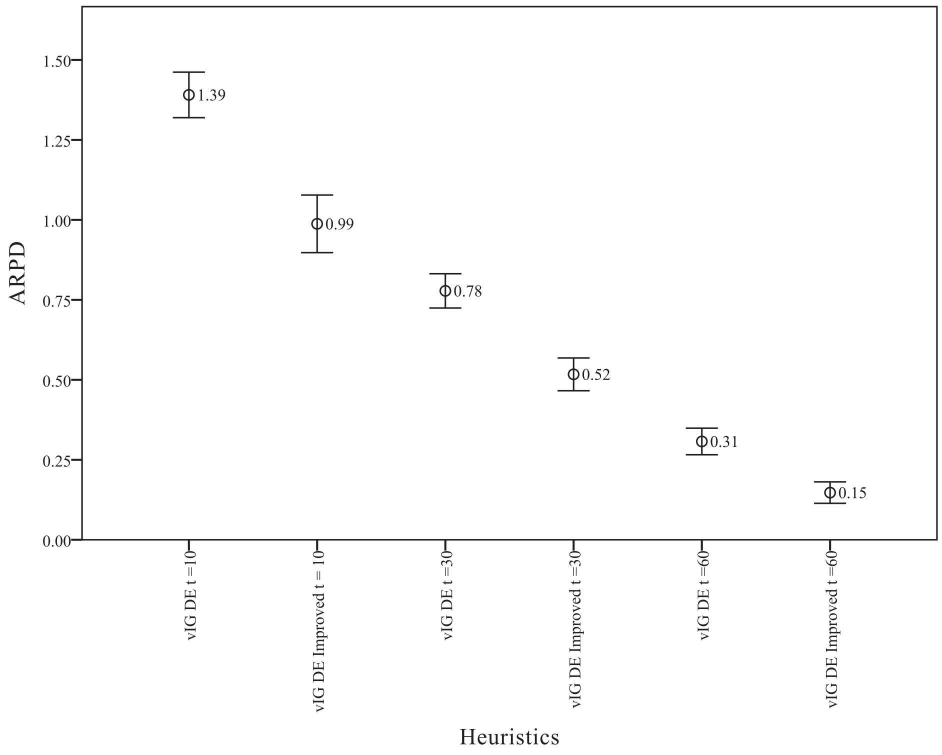

Figure 4. ARPD for IG algorithms, with a confidence interval of 95%.

the Proposed(1,15) heuristic. The remaining individuals in the target population are randomly constructed, and the modified NEH, with reinsertions utilized in Proposed(x, y), is applied to each of them. This new version of the IG was denoted as vIG DE Improved. To compare the algorithms, the same test from the previous section was used, using five replicates for each instance.

For the stop rule, the algorithms were run with $T_{max} = n(\frac{m}{2}) \times t$ ms, the same criterion used by Tasgetiren *et al.* (2013). In the online supplementary material there is a figure that presents the intervals of the ARPD of each algorithm, with a confidence interval of 95%. The vIG DE Improved version presented the best ARPD for each stopping time, with average results 0.99, 0.52, and 0.15%, against 1.39, 0.78, and 0.31%, for t values equal to 10, 30, and 60 ms, respectively. In Figure 4, vIG DE Improved presents a significant statistical difference from vIG DE. Complete results of the ARPD and Tukey test for each version of the IG algorithm are in the online supplementary material.

Clearly, the IG algorithms produce considerably better solutions when compared to the simple heuristics from the previous section. However, note that for $t = 60$ the largest instances of 500 jobs and 50 machines take approximately 13 minutes to run, and around 65 minutes when five replications are considered, against the average of the CPU time of 22 seconds, for problems with 500 jobs from the best simple heuristic. This shows that the proposed simple heuristic is very computationally efficient.

The integration of the Proposed(x, y) heuristic with the vIG DE contributes to the generation of better solutions, which can be attributed to the fact that a high-quality initial population is efficiently generated by the proposed initialization procedure.

5. Conclusion

In this article, the production scheduling problem in a no-idle flowshop environment with minimizing total flowtime was addressed. After an extensive search in the literature, it was noted that this issue has not been developed deeply in the articles published, especially regarding constructive heuristics. To cover this deficiency, simple constructive high-quality heuristics of the most general problem, without no-idle restriction, were adapted to the problem in question.

A simple heuristic, called Proposed(x,y), was presented. The aim of the proposed heuristic was to optimize the partial sequences generated by the NEH using reinsertion of a limited set of jobs. The proposed heuristic and the adapted methods were compared using an extended benchmark composed by 250 problems instances with up to 500 jobs and 50 machines. The experiments show that the construction strategy used in the heuristic Proposed(x,y) ($x = \{5, 10\}$ and $y = \{15, 20\}$) results in better results (statistically significant) than the adapted methods, maintaining its computational efficiency, especially when compared to the FL-LS heuristic. For instance, FL-LS obtains an ARPD of 4.76 with a mean of CPU time of 14.60 seconds, while the method Proposed(10, 20) obtains an ARPD of 3.27 in almost half the time, 7.58 seconds. Among the least computationally intensive heuristics, the method LR-NEH(x) presented very good results, surpassing the NEH heuristic, when adapted to the problem.

Furthermore, the proposed heuristic was incorporated in the initialization procedure of the variable IG algorithm proposed by Tasgetiren *et al.* (2013). This new version was called vIG DE - Improved. The computational experiments show this integration significantly improves the solution quality of the algorithm.

For future research, it is possible to develop composite heuristics using the method proposed in this article. In addition, the proposed approaches can be considered for other scheduling problems with different objective functions, such as total tardiness and makespan.

Acknowledgments

This work was supported by the Brazilian National Research Council (CNPq) under grant number 448161/2014-1, 308047/2014-1 and 306075/2017-2.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Marcelo Seido Nagano http://orcid.org/0000-0002-0239-1725

References

Adiri, I., and D. Pohoryles. 1982. “Flowshop/No-Idle or No-Wait Scheduling to Minimize the Sum of Completion Times.” *Naval Research Logistics Quarterly* 29 (3): 495–504.
Benavides, A. J, and M. Ritt. 2016. “Two Simple and Effective Heuristics for Minimizing the Makespan in Non-Permutation Flow Shops.” *Computers & Operations Research* 66 (Supplement C): 160–169.
Čepek, O., M. Okada, and M. Vlach. 2000. “Note: On the Two-Machine No-Idle Flowshop Problem.” *Naval Research Logistics* 47 (4): 353–358.
Deng, G., and X. Gu. 2012. “A Hybrid Discrete Differential Evolution Algorithm for the No-Idle Permutation Flow Shop Scheduling Problem With Makespan Criterion.” *Computers and Operations Research* 39 (9): 2152–2160.
Dong, X., H. Huang, and P. Chen. 2008. “An Improved NEH-Based Heuristic for the Permutation Flowshop Problem.” *Computers & Operations Research* 35 (12): 3962–3968.
Fernandez-Viagas, V., and J. M. Framinan. 2014. “On Insertion Tie-Breaking Rules in Heuristics for the Permutation Flowshop Scheduling Problem.” *Computers & Operations Research* 45: 60–67.
Fernandez-Viagas, V., and J. M. Framinan. 2015. “A New Set of High-Performing Heuristics to Minimise Flowtime in Permutation Flowshops.” *Computers & Operations Research* 53: 68–80.
Fernandez-Viagas, Victor, Rainer Leisten, and Jose M Framinan. 2016. “A Computational Evaluation of Constructive and Improvement Heuristics for the Blocking Flow Shop to Minimise Total Flowtime.” *Expert Systems with Applications* 61: 290–301.
Framinan, J. M., J. N. D. Gupta, and R. Leisten. 2004. “A Review and Classification of Heuristics for Permutation Flow-Shop Scheduling With Makespan Objective.” *The Journal of the Operational Research Society* 55 (12): 1243–1255.

Framinan, J. M., and R. Leisten. 2003. “An Efficient Constructive Heuristic for Flowtime Minimisation in Permutation Flow Shops.” *Omega* 31 (4): 311–317.

Framinan, J. M., R. Leisten, and R. Ruiz-Usano. 2002. “Efficient Heuristics for Flowshop Sequencing With the Objectives of Makespan and Flowtime Minimisation.” *European Journal of Operational Research* 141 (3): 559–569.

Framinan, J. M., R. Leisten, and R. Ruiz-Usano. 2005. “Comparison of Heuristics for Flowtime Minimisation in Permutation Flowshops.” *Computers & Operations Research* 32 (5): 1237–1254.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. 1979. “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.” *Annals of Discrete Mathematics* 5: 287–326.

Huang, J. D., J. J. Liu, Q. X. Chen, and N. Mao. 2017. “Minimizing Makespan in a Two-Stage Flow Shop With Parallel Batch-Processing Machines and Re-Entrant Jobs.” *Engineering Optimization* 49 (6): 1010–1023.

Kalczynski, Pawel Jan, and Jerzy Kamburowski. 2005. “A Heuristic for Minimizing the Makespan in No-idle Permutation Flow Shops.” *Computers & Indunstrial Engineering* 49 (1): 146–154.

Kalczynski, P. J., and J. Kamburowski. 2007. “On the NEH Heuristic for Minimizing the Makespan in Permutation Flow Shops.” *Omega* 35 (1): 53–60.

Kalczynski, Pawel J, and Jerzy Kamburowski. 2008. “An Improved NEH Heuristic to Minimize Makespan in Permutation Flow Shops.” *Computers & Operations Research* 35 (9): 3001–3008.

Kalczynski, P. J., and J. Kamburowski. 2009. “An Empirical Analysis of the Optimality Rate of Flow Shop Heuristics.” *European Journal of Operational Research* 198 (1): 93–101.

Kalczynski, P. J., and J. Kamburowski. 2011. “On Recent Modifications and Extensions of the Neh Heuristic for Flow Shop Sequencing.” *Foundations of Computing and Decision Sciences* 36: 18–33.

Laha, D., and S. C. Sarin. 2009. “A Heuristic to Minimize Total Flow Time in Permutation Flow Shop.” *Omega* 37 (3): 734–739.

Li, X., Q. Wang, and C. Wu. 2008. “Heuristic for No-Wait Flow Shops With Makespan Minimization.” *International Journal of Production Research* 46 (9): 2519–2530.

Liu, J., and C. R. Reeves. 2001. “Constructive and Composite Heuristic Solutions to the $P//\Sigma C_t$ Scheduling Problem.” *European Journal of Operational Research* 132 (2): 439–452.

Liu, W., Y. Jin, and M. Price. 2016. “A New Nawaz-Enscore-Ham-Based Heuristic for Permutation Flow-Shop Problems With Bicriteria of Makespan and Machine Idle Time.” *Engineering Optimization* 48 (10): 1808–1822.

Liu, W., Y. Jin, and M. Price. 2017. “A New Improved NEH Heuristic for Permutation Flowshop Scheduling Problems.” *International Journal of Production Economics* 193 (Supplement C): 21–30.

Lu, Y-Y. 2016. “Research on No-Idle Permutation Flowshop Scheduling With Time-Dependent Learning Effect and Deteriorating Jobs.” *Applied Mathematical Modelling* 40 (4): 3447–3450.

Nagano, M. S., and J. V. Moccellin. 2002. “A High Quality Solution Constructive Heuristic for Flow Shop Sequencing.” *Journal of the Operational Research Society* 53 (12): 1374–1379.

Nagano, M. S., F. L. Rossi, and C. P. Tomazella. 2017. “A New Efficient Heuristic Method for Minimizing the Total Tardiness in a No-Idle Permutation Flow Shop.” *Production Engineering* 11 (4): 523–529.

Narain, L., and P. C. Bagga. 2005. “Flowshop/No-Idle Scheduling to Minimise the Mean Flowtime.” *The ANZIAM Journal* 47 (2): 265–275.

Nawaz, M., E. E. Enscore, and I. Ham. 1983. “A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem.” *Omega* 11 (1): 91–95.

Pan, Q-K., and R. Ruiz. 2013. “A Comprehensive Review and Evaluation of Permutation Flowshop Heuristics to Minimize Flowtime.” *Computers & Operations Research* 40 (1): 117–128.

Pan, Q-K., and R. Ruiz. 2014. “An Effective Iterated Greedy Algorithm for the Mixed No-Idle Permutation Flowshop Scheduling Problem.” *Omega* 44: 41–50.

Pan, Q-K., M. F. Tasgetiren, and Y-C. Liang. 2008. “A Discrete Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem.” *Computers & Industrial Engineering* 55 (4): 795–816.

Pan, Q-K., and L. Wang. 2008a. “No-Idle Permutation Flow Shop Scheduling Based on a Hybrid Discrete Particle Swarm Optimization Algorithm.” *The International Journal of Advanced Manufacturing Technology* 39 (7): 796–807.

Pan, Q-K., and L. Wang. 2008b. “A Novel Differential Evolution Algorithm for No-Idle Permutation Flow-Shop Scheduling Problems.” *European Journal of Industrial Engineering* 2 (3): 279–297.

Ponnambalam, S G, P Aravindan, and S Chandrasekaran. 2001. “Constructive and Improvement Flow Shop Scheduling Heuristics: An Extensive Evaluation.” *Production Planning & Control* 12 (4): 335–344.

Rad, S. F., R. Ruiz, and N. Boroojerdian. 2009. “New High Performing Heuristics for Minimizing Makespan in Permutation Flowshops.” *Omega* 37 (2): 331–345.

Ribas, I., R. Companys, and X. Tort-Martorell. 2010. “Comparing Three-Step Heuristics for the Permutation Flow Shop Problem.” *Computers & Operations Research* 37 (12): 2062–2070.

Ribas, I., R. Companys, and X. Tort-Martorell. 2017. “Efficient Heuristics for the Parallel Blocking Flow Shop Scheduling Problem.” *Expert Systems with Applications* 74 (Supplement C): 41–54.

Rossi, F. L., M. S. Nagano, and J. K. Sagawa. 2017. “An Effective Constructive Heuristic for Permutation Flow Shop Scheduling Problem With Total Flow Time Criterion.” *The International Journal of Advanced Manufacturing Technology* 90 (1): 93–107.

Rossi, F. L., M. S. Nagano, and R. F. Tavares Neto. 2016. “Evaluation of High Performance Constructive Heuristics for the Flow Shop With Makespan Minimization.” *The International Journal of Advanced Manufacturing Technology* 87 (1): 125–136.

Ruiz, R., E. Vallada, and C. Fernández-Martínez. 2009. “Scheduling in Flowshops with No-Idle Machines.” In *Computational Intelligence in Flow Shop and Job Shop Scheduling*, edited by U. K. Chakraborty, 21–51. Berlin: Springer.

Saadani, N. E. H., A. Guinet, and M. Moalla. 2005. “A Travelling Salesman Approach to Solve the $F/No-Idle/C_{max}$ Problem.” *European Journal of Operational Research* 161 (1): 11–20.

Shao, W., D. Pi, and Z. Shao. 2017. “Memetic Algorithm With Node and Edge Histogram for No-Idle Flow Shop Scheduling Problem to Minimize the Makespan Criterion.” *Applied Soft Computing* 54 (Supplement C): 164–182.

Tasgetiren, M. F., Q-K. Pan, P. N. Suganthan, and O. Buyukdagli. 2013. “A Variable Iterated Greedy Algorithm With Differential Evolution for the No-Idle Permutation Flowshop Scheduling Problem.” *Computers & Operations Research* 40 (7): 1729–1743.

Tasgetiren, M. F., Q-K. Pan, P. N. Suganthan, and T. J. Chua. 2011. “A Differential Evolution Algorithm for the No-Idle Flowshop Scheduling Problem With Total Tardiness Criterion.” *International Journal of Production Research* 49 (16): 5033–5050.

Woo, H-S., and D-S. Yim. 1998. “A Heuristic Algorithm for Mean Flowtime Objective in Flowshop Scheduling.” *Computers & Operations Research* 25 (3): 175–182.

Ying, K-C., S-W. Lin, C-Y. Cheng, and C-D. He. 2017. “Iterated Reference Greedy Algorithm for Solving Distributed No-Idle Permutation Flowshop Scheduling Problems.” *Computers & Industrial Engineering* 110 (Supplement C): 413–423.

Zheng, J-X., P. Zhang, F. Li, and G-L. Du. 2016. “The High Performing Backtracking Algorithm and Heuristic for the Sequence-Dependent Setup Times Flowshop Problem With Total Weighted Tardiness.” *Engineering Optimization* 48 (9): 1571–1592.

Zhou, Y., H. Chen, and G. Zhou. 2014. “Invasive Weed Optimization Algorithm for Optimization No-Idle Flow Shop Scheduling Problem.” *Neurocomputing* 137: 285–292.