

Document downloaded from:

<http://hdl.handle.net/10251/151047>

This paper must be cited as:

Pan, Q.; Ruiz García, R.; Alfaro-Fernandez, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*. 80:50-60. <https://doi.org/10.1016/j.cor.2016.11.022>



The final publication is available at

<https://doi.org/10.1016/j.cor.2016.11.022>

Copyright Elsevier

Additional Information

Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows

Quan-Ke Pan^a, Rubén Ruiz^{b,*}, Pedro Alfaro-Fernández^b

^a*State Key Laboratory of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, P. R. China*

^b*Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*

Abstract

In practice due dates usually behave more like intervals rather than specific points in time. This paper studies hybrid flowshops where jobs, if completed inside a due window, are considered on time. The objective is therefore the minimization of the weighted earliness and tardiness from the due window. This objective has seldom been studied and there are almost no previous works for hybrid flowshops. We present methods based on the simple concepts of iterated greedy and iterated local search. We introduce some novel operators and characteristics, like an optimal idle time insertion procedure and a two stage local search where, in the second stage, a limited local search on a exact representation is carried out. We also present a comprehensive computational campaign, including the reimplementation and comparison of 9 competing procedures. A thorough evaluation of all methods with more than 3000 instances shows that our presented approaches yield superior results which are also demonstrated to be statistically significant. Experiments also show the contribution of the new operators in the presented methods.

Keywords: hybrid flowshop, due date window, heuristics, iterated local search, iterated greedy

*Corresponding author. Tel: +34 96 387 70 07. Fax: +34 96 387 74 99
Email addresses: 2281393146@qq.com (Quan-Ke Pan), rruiz@eio.upv.es (Rubén Ruiz), pedro.al.fe@gmail.com (Pedro Alfaro-Fernández)

1. Introduction

Production scheduling is a very important step for manufacturing industries. Production schedules obtained by manual methods or without optimization techniques are known to have ample room for improvement as commented in Framinan et al. (2014). Therefore, using optimization methods results in large gains (McKay et al., 2002; Pinedo, 2012, among others). As a result, scheduling is a very active field inside operations research. Since the mid-fifties, thousands of papers have been published. However, the application of such research developments in real industry remains rare, or at least the proportion of papers tackling with realistic production settings is much lower than those papers dealing with simplified problem settings (MacCarthy and Liu, 1993; McKay et al., 2002; Ruiz and Maroto, 2006).

Among the different types of scheduling problems, we are interested in what is probably the most commonly found situation in **many** industries, which is the combination of the parallel machines and flowshop problems. The addition of the two is usually referred to as hybrid flowshop and can be defined as follows: There is a set M of $m \geq 2$ production stages where $M = \{1, \dots, m\}$. The stages are disposed in series. At each stage i , $i \in M$ we have a set M_i of m_i identical parallel machines where $M_i = \{1, \dots, m_i\}$ and $m_i \geq 1, \forall i \in M$ and $\exists i \in M, m_i > 1$. A set N of n jobs has to be processed on the stages. More precisely, each job j , $j \in N$ visits first stage 1, then stage 2 and so on until stage m . At each stage i , each job is to be processed by exactly one of the available m_i parallel machines. As a result, a job is made up of m different tasks, one per stage. The processing time of any job j at any stage i is a positive integer denoted by p_{ij} . Jobs are processed without interruptions.

The hybrid flowshop or HFS in short has attracted a lot of interest in the literature, given its potential practical applications. As such, there are several reviews published. Some of them are Linn and Zhang (1999), Vignier et al. (1999), Wang (2005), Quadts and Kuhn (2007) and more recently Ribas et al. (2010) and Ruiz and Vázquez-Rodríguez (2010). According to these last two most recent hybrid flowshop reviews, by far the most commonly studied optimization criterion is the minimization of the makespan or (C_{\max}), defined as $C_{\max} = \max_{j=1}^n C_j$ where C_j is the time at which job j is finished at stage m . C_j is commonly referred to as the completion time of job j . The HFS with makespan criterion can be represented as $((PM^{(i)})_{i=1}^m)/C_{\max}$, following the well known and accepted three field notation of Graham et al. (1979) and

38 the extension for hybrid flowshops proposed by Vignier et al. (1999). Note
39 that there are further assumptions stemming from the flowshop nature of
40 the problem, these are well known and are detailed elsewhere, for example
41 in Naderi et al. (2010) or in Framinan et al. (2014). In the HFS for each job
42 and stage we have to decide to which machine the job is assigned. Then, for
43 each machine at each stage, a sequence for the processing of the assigned
44 jobs has to be determined. The simplest possible HFS with only two stages
45 where one stage has one single machine and the other has just two parallel
46 machines was shown to be \mathcal{NP} -Hard by Gupta (1988). As a result, the best
47 state-of-the-art exact techniques can only cope with small problems with
48 few stages, machines and jobs. For real-sized problems metaheuristics are
49 currently the most promising solution approach.

50 While makespan minimization is the most studied objective in the schedul-
51 ing literature, accounting for more than 60% of the published papers in
52 the HFS according to the study of Ruiz and Vázquez-Rodríguez (2010), it
53 is not the most appropriate objective to study nowadays. As explained in
54 Framinan et al. (2014), makespan minimization increases machine utilization,
55 which made sense in the early days of manufacturing where costs had to
56 be minimized and production used to be the bottleneck. Presently, service
57 level objectives are much more important. We define by d_j the due date of
58 job j , $j \in J$. If $C_j > d_j$ then job j is tardy by $C_j - d_j$ units of time. The
59 tardiness of job j is $T_j = \max\{C_j - d_j, 0\}$ and the total tardiness objective
60 is $TT = \sum_{j=1}^n T_j$. About 6% of the published research in HFS considers this
61 objective. Additionally, not all jobs are equally important. Therefore, we can
62 use a weighted tardiness version by defining w_j as the weight of job j , $j \in J$
63 and the total weighted tardiness objective as $TWT = \sum_{j=1}^n w_j T_j$. About 2%
64 of the reviewed literature in Ruiz and Vázquez-Rodríguez (2010) consider this
65 objective. Note that when $C_j < d_j$ the tardiness is always 0 regardless of how
66 small C_j is. In real problems, finishing products early results in tied up inven-
67 tory and financial burden. A possibility is to account for the earliness as well,
68 defined as $E_j = \max\{d_j - C_j, 0\}$ or $TWE = \sum_{j=1}^n w'_j E_j$ with weights. Note
69 the different weight for earliness w'_j and tardiness w_j . Minimizing earliness
70 only potentially negates the tardiness and the solution is to minimize the
71 total earliness and tardiness or $TWET = \sum_{j=1}^n (w'_j E_j + w_j T_j)$. Only 1% of
72 the reviewed literature reviewed in Ruiz and Vázquez-Rodríguez (2010) deal
73 with this objective and most of the time without weights. Furthermore, the
74 introduction of earliness into the objective results in a non-regular function

75 with enormous implications as the objective function value can be improved
 76 by arbitrarily delaying the start or completion time of jobs at machines and
 77 semi-active schedules no longer contain the optimal solution.

78 Due dates that represent a specific point in time are not realistic (Sabun-
 79 cuoglu and Lejmi, 1999). Real due dates are actually due windows. Within
 80 the window the job is considered to be delivered on time. Therefore, we
 81 define job's j due window as $[d_j^-, d_j^+]$. Earliness and tardiness are redefined to
 82 consider this window as $T_j^{dw} = \max\{C_j - d_j^+, 0\}$ and $E_j^{dw} = \max\{d_j^- - C_j, 0\}$,
 83 respectively. The objective function that we consider is therefore the min-
 84 imization of the weighted earliness and tardiness from a distinct due date
 85 window for each job, with different weights for earliness and tardiness or
 86 $TWET^{dw} = \sum_{j=1}^n (w'_j E_j^{dw} + w_j T_j^{dw})$. The full problem studied is then den-
 87 oted as $((PM^{(i)})_{i=1}^m) // TWET^{dw}$ and we will refer to it as HFSDW in short.
 88 The result is a more realistic problem that might capture more easily industrial
 89 practice with the consideration of a more real objective function where not
 90 only the tardiness is considered but also earliness to avoid stocking product
 91 way before due dates. Distinct due dates for each job and different weights
 92 for earliness and tardiness allow practitioners to transfer to the objective
 93 priorities, rush orders and many other situations. Additionally, the important
 94 consideration of due windows also captures the reality where some jobs might
 95 have a tight due window of only one morning during the week whereas other
 96 jobs might be finished without penalties in longer time frames of several days.
 97 From the complexity hierarchies of objective functions given in Pinedo (2012)
 98 we conclude that the studied problem in this paper is \mathcal{NP} -Hard since the
 99 makespan version already belongs to this complexity class. Note that we
 100 could reach the same conclusion by considering that single machine problems
 101 with due-windows are already \mathcal{NP} -Hard. To the best of our knowledge, the
 102 problem studied in this paper with the $TWET^{dw}$ objective function has
 103 not been considered in the literature to this date. In this paper we propose
 104 simple methods based on Iterated Local Search and on Iterated Greedy for
 105 solving the problem. The choice of simple methods is mainly due to reasons:
 106 First, simple methods are easier to understand, implement and extend to
 107 other problem variants. They have fewer parameters and the results are easy
 108 to replicate. Second, recent studies in many different scheduling problems
 109 (Urlings et al., 2010b; Naderi et al., 2010; Framinan et al., 2014) indicate
 110 that simple methods give state-of-the-art results when compared to more
 111 complex approaches. As a result, these simple methods are easily transferable

112 to industries.

113 The remainder of this paper is organized as follows. In the next section
114 we briefly review the related literature. Sections 3 and 4 describe in detail
115 the proposed methods, which are later comprehensively tested in Section 5.
116 Finally, section 6 gives some concluding remarks and further research.

117 2. Literature review

118 Only 1% of the papers reviewed by Ruiz and Vázquez-Rodríguez (2010)
119 dealt with hybrid flowshop problems and earliness tardiness objectives (not
120 considering due windows). This includes Chang and Liao (1994) and Liu and
121 Chang (2000) where a *TWET* objective is considered in a linear combination
122 with other objectives. Janiak et al. (2007) also combine earliness and tardiness
123 (*ET*) with other functions. Finke et al. (2007) presented Tabu Search methods
124 to solve the HFS problem with *ET* objectives in a simplified setting where
125 job to machine assignments are given. Khalouli et al. (2010) presented an
126 Ant Colony Optimization (ACO) method for the HFS with *TWET* objective.
127 Behnamian et al. (2010a) presented a complex hybrid of population based
128 methods, ACO, Simulated Annealing and Variable Neighborhood Search
129 for the same problem but with the addition of sequence dependent setup
130 times. A similar work is that of Behnamian et al. (2010b) where in this case
131 Particle Swarm Optimization is used for a related problem of group scheduling.
132 More recently, Behnamian and Zandieh (2013) have included learning effects
133 into the problem. To the best of our knowledge, no other studies deal with
134 *ET* in HFS. Other less closely related works are those of Jolai et al. (2009)
135 where a no-wait hybrid flowshop with a common due date window is studied
136 with maximization of the profit for processing jobs. Sheikh (2013) adds the
137 earliness-tardiness criterion to the previous problem. Lastly, Pan et al. (2013)
138 studies a real problem in the steelmaking industry considering, among other
139 things, earliness and tardiness criterion. **Most existing research in the hybrid
140 flowshop literature considers, as mentioned, makespan or tardiness objectives,
141 as in Jun and Park (2015) but not earliness-tardiness.**

142 As regards due windows, most existing research has been carried out
143 on single machine problems. Starting with the work of Cheng (1988) where
144 the single machine with a common due window for all jobs is studied. Two
145 machine flowshop problems with common due windows are studied in Yeung
146 et al. (2004). The combination of hybrid flowshops and due windows, as stated
147 previously is almost non-existent in the literature. We only find recent related

148 papers, the first one by Huang and Yu (2013) which study a two stage hybrid
 149 flowshop with distinct due windows and an objective function which is a linear
 150 combination of makespan, ET without weights. In the second paper, the
 151 same authors Huang et al. (2014) introduce reentry of jobs into the problem.
 152 These two papers disregard different weights for the jobs and for the earliness
 153 and tardiness and only consider two stages in the hybrid flowshop. Therefore
 154 and as we can see, the general hybrid flowshop with m stages and $TWET^{dw}$
 155 objective has not been yet considered in the literature.

156 3. Iterated Local Search method

157 Iterated Local Search (ILS) is, as the name implies, an iterative method
 158 that, starting from an initial solution, applies a local search to obtain a local
 159 optimum. Then a perturbation is applied to escape from it. The process iterates
 160 with the application of local search again to obtain a new local optimum.
 161 The procedure is simple to code. This is ideal for scheduling problems as
 162 there are many potential variations and objectives. ILS is a natural choice
 163 for scheduling problems given its past achievements. For example, for the
 164 flowshop problem with makespan criterion, the ILS of Stützle (1998) is one
 165 of the best methods according to the evaluation of Ruiz and Maroto (2005)
 166 and as detailed in Lourenço et al. (2010) for problems like the single machine
 167 with total weighted tardiness criterion, parallel machines, jobshop, graph
 168 bipartitioning and MAX-SAT, etc. Lastly, Naderi et al. (2010) obtained the
 169 best results for a type of HFS problem with a variant of ILS. As a result,
 170 ILS is a promising approach to be applied to the HFSDW problem studied
 171 in this paper. We choose the simplest memory-less ILS variant, outlined
 in Figure 1. Basically all we need is a way to represent a solution for the

```

procedure ILS
   $\pi_0 :=$  GenerateInitialSolution
   $\pi :=$  LocalSearch( $\pi_0$ )
  while (termination criterion not satisfied) do
     $\pi' :=$  Perturbation( $\pi$ )
     $\pi'' :=$  LocalSearch( $\pi'$ )
     $\pi :=$  AcceptanceCriterion( $\pi'', \pi$ )
  endwhile
end

```

172 Figure 1: Iterated Local Search (ILS) method (Lourenço et al., 2010).
 173 HFSDW problem, an initialization procedure, local search, perturbation and
 174 acceptance criterion. These components are detailed in the following sections.

175 *3.1. Solution representation*

176 As discussed in Ruiz and Maroto (2006) there are several possible solution
177 representations for the HFS. The most direct representation would be to
178 have a list of jobs for every machine at each stage and then to decide the
179 start and completion times of each job at each machine. Of course, such
180 an exact representation would result in a very large search space and more
181 compact approaches are desirable. Different representations for complex HFS
182 problems were explored by Urlings and Ruiz (2007) and by Urlings et al.
183 (2010a) where it was demonstrated that the more detailed the representation,
184 the worse the results. Indirect representations employ surrogate heuristics as
185 decoding procedures for completing the solution and the outcome is usually
186 a much better solution than with a direct representation. As a result, the
187 vast majority of the literature uses an indirect permutation representation.
188 This job permutation indicates the order in which the jobs are going to be
189 processed in the first stage of the HFS. The machine assignment decision is
190 left for a dispatching rule. Many rules are proposed in Urlings et al. (2010a),
191 however, for the HFS with identical parallel machines, the most common is
192 the well known First Available Machine of FAM. Let us suppose we have a
193 solution represented by a permutation of jobs $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, where π_j
194 denotes the job occupying position j in the permutation. At the first stage
195 we have to decide to which machine we assign job π_1 . Since all machines are
196 free, we assign it to machine 1 at the first stage. Then jobs π_2 through π_n
197 are assigned to the machine that is available at the earliest time. When all jobs
198 have been completed at stage 1 we proceed with stage 2 through stage m .
199 Note that jobs can finish at different times so instead of using permutation
200 π to decide the launch order of jobs for the remaining stages we order the
201 jobs according to their completion times at the previous stage. Therefore,
202 $\pi^{(i)} = (\pi_1^{(i)}, \pi_2^{(i)}, \dots, \pi_n^{(i)})$ is the permutation that indicates the order in which
203 the jobs will be processed at stage i . This order is obtained by sorting the
204 jobs in ascending completion times at stage $i - 1$, where $\pi^{(1)} = \pi$. After the
205 jobs are sorted, the FAM rule is applied again for machine assignment. It is
206 important to note that during sorting, ties might occur with jobs having the
207 same completion time at a given stage. To break ties, we favor the job with
208 the smallest slack in the due date: $d_j^+ - C_{i-1,j}$. The computational complexity
209 of this decoding method is $\mathcal{O}(n \sum_{i=1}^m m_i + m \log(n))$. Note that $\sum_{i=1}^m m_i > m$
210 or otherwise we would not have a HFS problem so the complexity is then
211 $\mathcal{O}(n \sum_{i=1}^m m_i)$.

212

Let us give an example of the decoding method. We have a HFSDW problem with five jobs and two stages, where each stage contains two machines. The processing times, due windows and earliness and tardiness weights are the following:

$$\begin{aligned} (p_{i,j})_{2 \times 5} &= \begin{pmatrix} 4 & 3 & 6 & 2 & 1 \\ 5 & 4 & 1 & 1 & 4 \end{pmatrix}, \begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix}_{1 \times 5} = \begin{pmatrix} 8 & 7 & 10 & 7 & 9 \\ 10 & 9 & 11 & 10 & 11 \end{pmatrix}, \\ \begin{pmatrix} w'_j \\ w_j \end{pmatrix}_{1 \times 5} &= \begin{pmatrix} 1 & 2 & 1 & 3 & 1 \\ 2 & 1 & 2 & 1 & 3 \end{pmatrix} \end{aligned}$$

213 Now let us decode solution $\pi = (1, 2, 3, 4, 5)$. At the first stage job 1 is assigned
 214 to machine 1 with completion time $C_{1,1} = 4$, job 2 to machine 2 with $C_{1,2} = 3$.
 215 Job 3 is assigned to the first available machine (2), so $C_{1,3} = 3 + 6 = 9$. For
 216 job 4 the first available machine is 1 and $C_{1,4} = 4 + 2 = 6$. For job 5 we
 217 have $C_{1,5} = 6 + 1 = 7$. For the second stage jobs do not finish in the same
 218 order in which they were launched and after ordering the jobs by increasing
 219 completion times the permutation is $\pi^{(2)} = (2, 1, 4, 5, 3)$. Applying the same
 220 process and considering the completion times at the previous stage, job 2 is
 221 assigned to machine 1 with $C_{2,2} = 7$, job 1 to machine 2 ($C_{2,1} = 9$), job 4 to
 222 machine 1 ($C_{2,4} = 8$), job 5 to machine 1 ($C_{2,5} = 12$) and job 3 to machine 1
 223 ($C_{2,3} = 10$). We calculate the earliness and tardiness with the job completion
 224 times ($C_1 = 9$, $C_2 = 7$, $C_3 = 10$, $C_4 = 8$ and $C_5 = 12$). Jobs 1 through 4
 225 complete within their respective due windows and the only tardy job is 5
 226 with one unit of tardiness. Given that $w_5 = 3$ the objective function value
 227 for this solution π is $\sum_{j=1}^n (w'_j E_j^{dw} + w_j T_j^{dw}) = 3$. The Gantt chart for this
 228 solution is given in Figure 2.

229 3.2. Idle time insertion method

230 After decoding, jobs are scheduled as soon as possible in all machines in a
 231 semi-active schedule and many jobs will potentially complete way before their
 232 due windows, resulting in high earliness penalties. Therefore, a procedure
 233 to schedule the different tasks closer to their due windows is needed. The
 234 insertion of idle times is common in the earliness-tardiness literature. However,
 235 as we have seen from Section 2, little has been done in the HFS in this
 236 regard. We present an idle time insertion procedure that is applied after
 237 the decoding method. We will show that the proposed procedure results in
 238 optimal completion times once the sequence of jobs for each machine is given.
 239 Note that there are other exact methods to insert idle time in scheduling

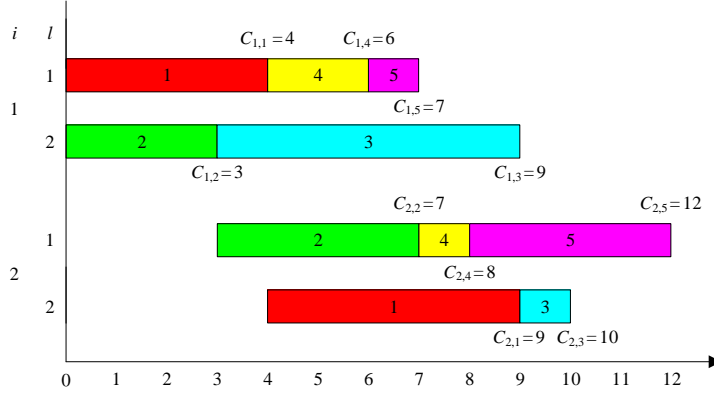


Figure 2: Gantt chart for permutation $\pi = (1, 2, 3, 4, 5)$ in the example.

240 problems with earliness and tardiness, most notably the one proposed by
 241 Hendel and Sourd (2007). This kind of methods are commonly referred to as
 242 timing algorithms (Sakuraba et al., 2009). However, most of these methods
 243 are not fast enough to be used for solution decoding. The procedure is loosely
 244 inspired by the Net Benefit of Movement (NBM) by Tseng and Liao (2008) on
 245 a type of flowshop lot-streaming problem and is an extension of the right sub-
 246 block move of Kedad-Sidhoum and Sourd (2010) that was initially proposed
 247 for the single machine earliness-tardiness minimization problem without due
 248 windows.

249 The procedure is applied to all machines in the last stage of the hybrid
 250 flowshop. Since jobs have been scheduled as soon as possible, only by forward
 251 movements or by right shifting we can improve the objective function, i.e., by
 252 the insertion of idle time before starting jobs. Given any machine in the last
 253 stage we will have jobs or blocks of consecutive jobs separated by idle times.
 254 Let us refer to any of these blocks of jobs as S_M . Now we divide the jobs in S_M
 255 into three different subsets: Jobs that are early or $S_E = \{j | E_j > 0, j \in S_M\}$,
 256 jobs that are tardy or $S_T = \{j | T_j \geq 0, j \in S_M\}$ and jobs that are on time,
 257 finishing within the due window or $S_D = \{j | C_j \geq d_j^-, C_j < d_j^+, j \in S_M\}$. Note
 258 that the minimum idle time insertion is one unit, that is why a job j with
 259 $C_j = d_j^+$ while on time and not tardy, is included in S_T since it would be
 260 tardy after inserting just one unit of idle time. Now if all the jobs in S_M
 261 are moved forward one unit, the earliness will decrease by $\sum_{j \in S_E} w'_j$. At the same
 262 time, the tardiness will increase by $\sum_{j \in S_T} w_j$. Therefore, we have a simple
 263 way of deciding if idle time should be inserted before S_M which is if the gains
 264 in earliness are greater than losses in tardiness, i.e., $\sum_{j \in S_E} w'_j > \sum_{j \in S_T} w_j$.

265 Actually we can keep inserting idle time until either the sets S_E or S_T change.
 266 Instead of inserting units one by one we can calculate the maximum possible
 267 idle time insertion for S_M as $\Delta_1 = \min \left\{ \min_{j \in S_E} \{E_j\}, \min_{j \in S_D} \{d_j^+ - C_j\} \right\}$.
 268 Insertion of more than Δ_1 units of idle time results in either a job moving
 269 from S_D to S_T and/or a job moving from S_E to S_D . Additionally, there might
 270 be another block of jobs after S_M so we denote by Δ_2 the idle time in between
 271 both blocks. Considering all of the above, the maximum idle time insertion
 272 before a block S_M is $\Delta = \min \{\Delta_1, \Delta_2\}$. This procedure is applied to all
 273 blocks of jobs at all machines in the last stage, starting from the last job
 274 and considering one job at a time. Let us now formalize it. We denote n_l
 275 to the number of jobs assigned to machine l inside the last stage m . Also,
 276 $\gamma_l = \{\gamma_{1,l}, \gamma_{2,l}, \dots, \gamma_{n_l,l}\}$ is the partial permutation of the jobs assigned to
 277 machine l . The pseudocode of the idle time insertion procedure is given in
 Figure 3. While the procedure looks intricate, it is actually very efficient.

```

procedure IdleTimeInsertion( $\pi$ )
  for  $l := 1$  to  $m_m$  do
     $j := n_l$ 
    while  $j > 0$  do
      Construct a block of jobs  $S_M$  starting from job  $\gamma_{j,l}$ 
      if there is a job to the right of  $S_M$  then Calculate idle time  $\Delta_2$ 
      else  $\Delta_2 := +\infty$ 
      Generate  $S_E, S_D$  and  $S_T$  from  $S_M$ 
      if  $\sum_{j \in S_E} w'_j > \sum_{j \in S_T} w_j$  then
        Calculate  $\Delta_1 = \min \left\{ \min_{j \in S_E} \{E_j\}, \min_{j \in S_D} \{d_j^+ - C_j\} \right\}$ 
        Insert  $\Delta = \min \{\Delta_1, \Delta_2\}$  time units before  $S_M$ 
        Set  $C_{m,k} = C_{m,k} + \Delta, \forall k \in S_M$ 
        Set  $E_k = E_k - \Delta, \forall k \in S_E$ 
        Set  $T_k = T_k + \Delta, \forall k \in S_T$ 
      else  $j := j - 1$ 
    endwhile
  endfor
end
  
```

Figure 3: Idle time insertion procedure.

278
 279 The outer loop goes for all machines in the last stage and the inner loop for
 280 all jobs assigned to each machine. In total there are n jobs assigned to the
 281 different machines. The calculation of Δ_1 also requires traversing all jobs so
 282 in the worst case the computational complexity of the idle time insertion
 283 heuristic is $\mathcal{O}(n^2)$.

The previous procedure can be trivially extended to insert idle times into all stages. However, for simplicity and as this would not improve the objective

function value and would increase the computational times, it is not carried out in this paper. This idle time insertion procedure is a significant extension to parallel machines and due windows of the Net Benefit of Movement (NBM) of Tseng and Liao (2008). The authors proved NBM to be optimal for inserting idle time in the last machine of a lot-streaming flowshop. Since we apply the same principle to all machines in the last stage of the HFS, it follows that it is also optimal. We proceed now with an example with eight jobs focused on only one machine among the m_m parallel machines at the last stage m :

$$\begin{aligned} (C_j)_{1 \times 8} &= (61, 65, 69, 75, 82, 86, 92, 97), \\ \begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix}_{1 \times 8} &= \begin{pmatrix} 62 & 67 & 70 & 74 & 79 & 88 & 93 & 97 \\ 64 & 68 & 72 & 76 & 81 & 90 & 95 & 99 \end{pmatrix}, \\ \begin{pmatrix} w'_j \\ w_j \end{pmatrix}_{1 \times 8} &= \begin{pmatrix} 1 & 2 & 1 & 3 & 1 & 2 & 1 & 4 \\ 2 & 1 & 2 & 1 & 2 & 1 & 2 & 3 \end{pmatrix} \end{aligned}$$

284 The initial schedule is shown in Figure 4a where $\pi = (1, 2, 3, 4, 5, 6, 7, 8)$. Jobs
285 1, 2, 3, 6 and 7 are early ($E_1 = 1, E_2 = 2, E_3 = 1, E_6 = 2, E_7 = 1$). Jobs 4 and
286 8 finish on time and job 5 is tardy ($T_5 = 1$). Therefore, $TWET^{dw} = 13$. The
287 procedure starts with the last job. Job 8 finishes on time, therefore $j := j - 1$
288 and job 7 is considered. Job 8 is to the right of this S_M and the idle time
289 is $\Delta_2 = 1$. Job 7 is early so we insert one unit of idle time before it. S_M
290 is recalculated. Jobs 7 and 8 form a block and $S_M = \{7, 8\}$. Now both jobs
291 7 and 8 are on time. Job 6 is now considered. It is 2 units early. $\Delta_2 = 1$
292 due to jobs 7 and 8. After inserting one unit of idle time before job 6 we
293 recalculate completion times, earliness and tardiness. The resulting schedule
294 is shown in Figure 4b with an improvement in the $TWET^{dw}$ of 3 units. Now
295 S_M has changed to $S_M = \{6, 7, 8\}$. There is no job/block to the right of this
296 S_M so $\Delta_2 = +\infty$. From the jobs in S_M , $S_E = \{6\}$ and $S_D = \{7, 8\}$ and
297 there are no tardy jobs. $\Delta_1 = \min \{ \min \{1\}, \min \{95 - 93, 99 - 97\} \} = 1$, so
298 $\Delta = 1$ which is what we insert before job 6, resulting in the schedule shown
299 in Figure 4c. The solution is further improved by 2 units. After this all three
300 jobs in the block are on time. Then we consider job 5, which is late. The
301 next job is 4, forming a block with 5. As job 4 is on time, delaying this block
302 is not advisable. We deal with job 3, forming a block with 4 and 5. Now
303 $S_M = \{3, 4, 5\}$, $S_E = \{3\}$, $S_D = \{4\}$ and $S_T = \{5\}$ so we calculate if inserting
304 idle time is profitable: $\sum_{j \in S_E} w'_j = 1$ and $\sum_{j \in S_T} w_j = 2$ so there is no gain.
305 We include job 2 in the block which is early and its earliness weight is 2 so,

306 together with job 3 makes the movement advisable. The procedure continues
 307 in a similar way until all jobs are considered.
 308 The final schedule is given in Figure 4d.

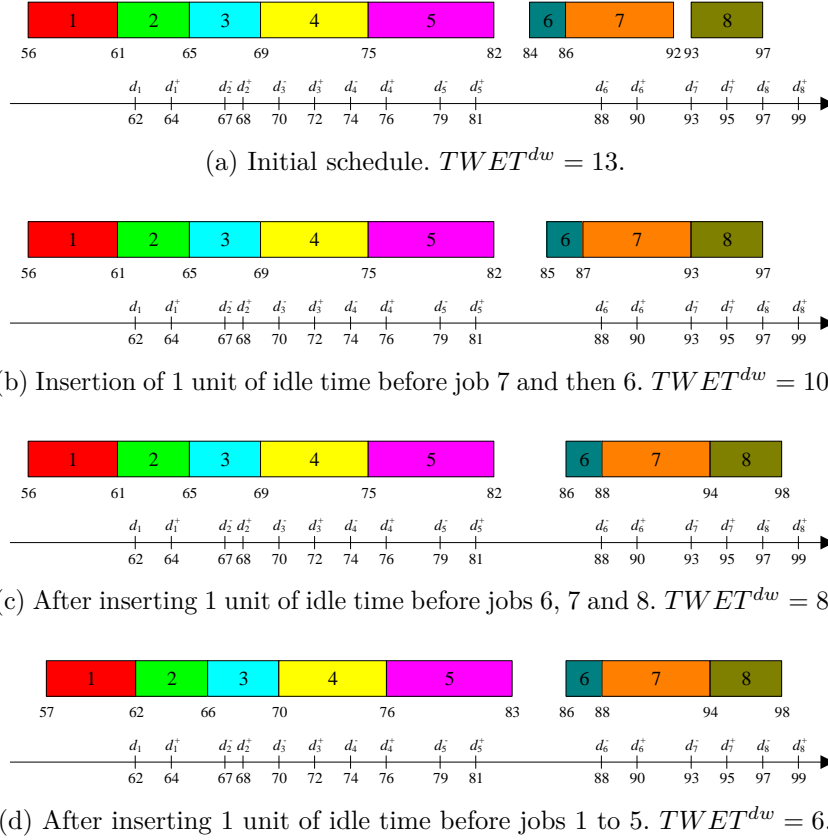


Figure 4: Gantt charts for the example application of the idle time insertion procedure.

309 3.3. Initialization

310 We select quick and reasonable heuristics for the initialization of the
 311 proposed methods. For problems with earliness and tardiness criterion, three
 312 dispatching rules are commonly used. The first one is the well known Earliest
 313 Due Date or EDD where jobs are simply sorted in increasing order of their due
 314 dates, i.e., the EDD produces a permutation of jobs $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ so that
 315 $d_{\pi(j)}^+ \leq d_{\pi(j+1)}^+, j = 1 \dots, n - 1$ where $\pi(j)$ denotes the job occupying the j -th
 316 position in the permutation. The second dispatching rule is the smallest slack
 317 on the last machine or LSL. Here we consider the processing times of the jobs at
 318 the last stage and sort them according to $d_{\pi(j)}^+ - p_{m, \pi(j)} \leq d_{\pi(j+1)}^+ - p_{m, \pi(j+1)}, j =$

319 $1 \dots, n - 1$. The third and last rule is the overall slack time or OSL. It
320 is a generalization of the previous rule in which the slack is calculated
321 with respect to the processing times at all stages, i.e., $d_{\pi(j)}^+ - \sum_{i=1}^m p_{i,\pi(j)} \leq$
322 $d_{\pi(j)}^+ - \sum_{i=1}^m p_{i,\pi(j+1)}$, $j = 1 \dots, n - 1$. All these heuristics are very fast needing
323 only $\mathcal{O}(n \log n)$ steps. We calculate all three given permutations and after
324 decoding and idle time insertion we keep the best solution as an initial schedule
325 for the proposed methods.

326 3.4. Local Search

327 After initialization (and also after the perturbation phase inside the main
328 loop, which is explained in next Section), a local search is carried out. We apply
329 a two stage local search. The first one works over the indirect representation
330 of a solution in the insertion and interchange neighborhoods. These two
331 neighborhoods are explored inside a simple Variable Neighborhood Descent
332 (VND) method. The application of VND with these two neighborhoods has
333 produced state-of-the-art performance in a number of flowshop and hybrid
334 flowshop problems as detailed in Tasgetiren et al. (2007) and Naderi and
335 Ruiz (2010). The procedure is simple, we take a solution π and apply a
336 single insertion movement, i.e., a job is randomly selected, extracted from its
337 position and inserted in another random position in the sequence. The new
338 permutation is decoded and the idle time insertion procedure is applied. If
339 the solution is not improved, in the next iteration a random interchange of
340 jobs is carried out. If improvements are found we go back to the insertion
movement as depicted in the simple VND procedure of Figure 5. Note that we

```

procedure VND( $\pi, loop_{\max}$ )
   $loop := 1$ 
  while  $loop < loop_{\max}$  do
     $N_{counter} := 1$ 
    while  $N_{counter} < 3$  do
      if  $N_{counter} = 1$  then  $\pi' = \text{InsertionMovement}(\pi)$ 
      if  $N_{counter} = 2$  then  $\pi' = \text{InterchangeMovement}(\pi)$ 
      if  $TWET^{dw}(\pi') < TWET^{dw}(\pi)$  then
         $\pi = \pi'$ 
         $N_{counter} := 1$ 
      else  $N_{counter} := N_{counter} + 1$ 
    endwhile
     $loop := loop + 1$ 
  endwhile
end

```

Figure 5: Variable Neighborhood Descent sampling local search in the permutation space.

341

342 do not explore the whole insertion and interchange neighborhoods, as doing
343 so, together with the solution decoding and idle time insertion procedure,
344 would result in a very slow local search mechanism. Alternatively, both
345 neighborhoods are iteratively explored until no improvements are found for
346 $loop_{\max}$ iterations, this being a parameter of the local search.

347 As we mentioned in Section 3.1, an indirect representation gives better
348 results. Still, with an indirect representation some potentially good solutions
349 are left unexplored. This fact was examined by Urlings and Ruiz (2007)
350 in their shifting representation search method. For our research, instead of
351 changing the representation we propose a second stage local search procedure
352 that works in the exact or complete representation. A similar local search
353 was presented by Pan et al. (2014) but for a problem with makespan criterion
354 and no due windows.

355 An exact or complete representation is an ordered list of jobs for each
356 machine at each stage. The neighborhood to explore is the result of extracting
357 a job from a position at a given machine and inserting it into all other positions
358 of that machine and all positions of the other machines in the same stage. In
359 total this means there are $\prod_{i=1}^m n(n-1)(m_i-1)$ neighbors of a given solution
360 considering all stages, which is challenging to explore effectively. Instead we
361 focus on a characteristic of the HFS problem at hand. Let us picture the
362 schedule given for the example in Figure 2. Machine 1 at stage 2 completes
363 job 2 at time 7 ($C_{2,2} = 7$). The decoding procedure schedules job 4 afterwards
364 since that machine is free before machine 2 (which is processing job 1) and also
365 because jobs are selected according to the completion times at the previous
366 stage, i.e., $\pi^{(2)} = (2, 1, 4, 5, 3)$. However, we observe that at time 7, when we
367 schedule job 4 according to $\pi^{(2)}$, job 5 was already finished at the first stage
368 and could have been scheduled as well. If we had done so, job 5 would have
369 been scheduled after job 2 on machine 1 of stage 2, with a completion time of
370 $C_{2,5} = 11$. Then jobs 4 and 3 are assigned, following the decoding procedure,
371 to machine 2 of stage 2 with completion times $C_{2,4} = 10$ and $C_{2,3} = 11$,
372 respectively. This schedule does not generate any idle time (which would
373 cause problems in downstream stages) and considering these new completion
374 times all jobs finish within their due windows and $TWET^{dw} = 0$. Therefore,
375 the idea to exploit is not to examine the complete neighborhood, but only
376 these situations where several jobs are ready to be scheduled and different
377 solutions can be obtained by making different choices of which job should be
378 scheduled first, without incurring in additional idle times. Let us now define

```

procedure FindLimitedNeighbors( $i, \pi^{(i)}$ )
   $M := \emptyset$ 
  for  $j := 1$  to  $n - 1$  do
    Find first available machine  $l^* = \underset{l=1}{\operatorname{argmin}} \{\varepsilon_l\}$  at stage  $i$ 
     $h := j + 1$ 
    while  $C_{i-1, \pi_h^{(i)}} \leq \varepsilon_{l^*}$  do
       $\delta^{(i)} :=$  exchange jobs  $\pi_j^{(i)}$  and  $\pi_h^{(i)}$  in permutation  $\pi^{(i)}$ 
       $M := M \cup \{\delta^{(i)}\}$ 
       $h := h + 1$ 
    endwhile
    Assign job  $\pi_j^{(i)}$  to machine  $l^*$ 
     $C_{i, \pi_j^{(i)}} = \max \left\{ \varepsilon_{l^*}, C_{i-1, \pi_j^{(i)}} + p_{i, \pi_j^{(i)}} \right\}$ 
     $\varepsilon_{l^*} = C_{i, \pi_j^{(i)}}$ 
  endfor
end

```

Figure 6: Procedure to find limited neighbors used in the decoding.

379 this limited neighborhood structure. As stated, $\pi^{(i)} = (\pi_1^{(i)}, \pi_2^{(i)}, \dots, \pi_n^{(i)})$
 380 is the job permutation generated for stage i after sorting the jobs by their
 381 completion times at stage $i - 1$, where $i \geq 2$. Recall that the decoding
 382 procedure will assign jobs to the first available machine at stage i following
 383 permutation $\pi^{(i)}$. We define by ε_l the availability time of machine l inside
 384 stage i (the time at which it completed its last assigned job). Therefore, the
 385 machine with the earliest availability time is $l^* = \underset{l=1}{\operatorname{argmin}} \{\varepsilon_l\}$. Let us assume
 386 that from this permutation and at the earliest availability time among all
 387 machines, jobs $\pi_j^{(i)}, \pi_{j+1}^{(i)}, \dots, \pi_{j+k}^{(i)}$ were completed in stage $i - 1$ before ε_{l^*} ,
 388 i.e., $C_{i-1, \pi_h^{(i)}} \leq \varepsilon_{l^*}, \forall h \in j, j + 1, \dots, j + k$. Then a neighboring solution of $\pi^{(i)}$
 389 is the one that results from exchanging jobs $\pi_j^{(i)}$ and $\pi_{j+k}^{(i)}, \forall j \in 1, \dots, k$ in
 390 the permutation $\pi^{(i)}$. All these neighbors can be extracted easily during the
 391 decoding method by using the algorithm depicted in Figure 6. The second
 392 stage local search based on this neighborhood consists of extracting and
 393 evaluating all M neighbors stage by stage. The complete procedure is detailed
 394 in Figure 7. The limited local search is applied only once after the VND
 395 search has obtained a complete solution. At every step, the idle time insertion
 396 procedure is applied. The local search step given in the pseudocode of Figure 1
 397 is just the consecutive application of the VND given in Figure 5 and the local
 398 search based on the limited exact representation of Figure 7.


```

procedure LimitedLocalSearch( $\pi$ )
  for  $i := 2$  to  $m$  do
     $M :=$ FindLimitedNeighbors( $i, \pi^{(i)}$ )
    while  $|M| > 0$  do
       $\delta^{(i)} :=$  extract permutation from  $M$ 
      Schedule stage  $i$  according to  $\delta^{(i)}$ 
      Schedule stages  $i + 1, i + 2, \dots, m$  using the decoding method
      if  $TWET^{dw}(\delta^{(i)}) < TWET^{dw}(\pi)$  then  $\pi^{(i)} = \delta^{(i)}$ 
    endwhile
  endfor
end

```

Figure 7: Second stage local search based on the limited exact representation space.

399 3.5. Perturbation

400 The perturbation step is central to the ILS procedure (Lourenço et al.,
401 2010). The perturbation disrupts the solution, moving it away in the solution
402 space, to a new region, with the hope of finding a higher quality solution.
403 We combine the simplicity of the random movements presented in Stützle
404 (1998) with the enhancements given in Naderi et al. (2010). More precisely,
405 we carry out a random number ω of job insertions and interchanges in the
406 permutation of a given solution. As a result, the new perturbed solution
407 is usually worse than the original solution. In order to avoid this problem
408 we direct the perturbation. The previous operator is performed ϖ times,
409 henceforth obtaining ϖ different solutions. All of them are decoded and the
410 idle time insertion procedure is applied. The perturbed solution with the best
411 $TWET^{dw}$ value is chosen for the next iteration. As shown in Naderi et al.
412 (2010), this perturbation operator resulted in a much more effective algorithm.
413 Increasing the value of ϖ too far has a strong effect on computational time.
414 Furthermore, it is expected that ϖ will interact with the number of movements
415 ω . Therefore, both factors will be calibrated in later sections.

416 3.6. Acceptance Criterion

417 At each loop the incumbent solution is perturbed and improved by a local
418 search. This improved solution, referred to as π'' in Figure 1, might be better
419 or worse than the incumbent solution π . Accepting only better solutions
420 results in algorithms that get easily stuck in strong local optima. The simplest
421 acceptance criterion for ILS is the one presented in Stützle (1998) where
422 worse solutions are accepted according to a simplified constant temperature
423 simulated annealing-like function. This temperature depends on the processing

424 times and number of machines as follows: $Temp = T \frac{\sum_{j=1}^n \sum_{i=1}^m p_{ij}}{10 \cdot \sum_{i=1}^m m_i}$ where T is a
425 parameter to calibrate. Many studies, like Stützle (1998) or more recently Pan
426 and Ruiz (2014) have demonstrated in statistical experiments, the robustness
427 of the T parameter for different scheduling problems.

428 We introduce a new improved operator. The importance of the acceptance
429 criterion should not be underrated. After a number of improvement moves,
430 every iterative method spends a large percentage of the CPU time with
431 non-improving solutions. The new operator is based on tournament selection.
432 We keep a list of historical solutions which at first contains the initial one
433 (after local search). Then if π'' is better than the best solution found so far,
434 the list is cleared. If π'' is worse than the incumbent π , it is appended at
435 the end of the list. Then, using a parameter θ we randomly select, without
436 repetition, θ solutions from this list and carry out a θ -tournament for selecting
437 the incumbent solution for the next iteration, i.e., the best solution among
438 the θ randomly picked ones from the historical list is selected. As we can see,
439 the value of θ is a direct parameter for controlling the intensification since
440 a very large value of θ is similar to only accepting better solutions. If at a
441 given moment the list of historical solutions contains less than θ elements, the
442 selection operator always results in the best solution found so far. As with
443 the other parameters, the value of θ will be calibrated.

444 4. Iterated Greedy approach

445 In this paper we also propose an adaptation of the Iterated Greedy (IG)
446 method of Ruiz and Stützle (2007) for the problem studied. Very few studies,
447 apart from Urlings and Ruiz (2007), Urlings et al. (2010a) and a few more
448 have studied IG as a methodology for solving HFS problems. IG is similar to
449 ILS. The main difference is that instead of a perturbation operation, IG has
450 a destruction and reconstruction operator. Therefore, it suffices to explain in
451 detail just this difference, as all other steps are identical to those of ILS. In
452 the destruction a complete solution has some of its elements removed. In the
453 reconstruction, these elements are reintroduced, one at a time, following a
454 high performance greedy heuristic, back into the solution. All other steps are
455 identical to ILS. IG has been successfully applied to many different scheduling
456 problems, often resulting in state-of-the-art performance. Apart from the
457 already cited results in the permutation flowshop with makespan criterion
458 of Ruiz and Stützle (2007) and complex hybrid flowshops of Urlings and

459 Ruiz (2007) and Urlings et al. (2010a), IG has also demonstrated excellent
 460 performance in parallel machines problems (Fanjul-Peyro and Ruiz, 2010),
 461 flowshops with blocking (Ribas et al., 2011), no-wait (Pan et al., 2008)
 462 amongst others. In this paper we propose a straightforward IG procedure. In
 463 the destruction operator, d jobs from the permutation are randomly selected
 464 and removed from the sequence. In the reconstruction, removed jobs are
 465 considered in the order in which they were eliminated and are reinserted,
 466 one by one, into all possible positions in the sequence. For each position the
 467 partial sequence is evaluated (and idle time inserted) and each removed job
 468 is finally placed in the position resulting in the lowest $TWET^{dw}$ value. The
 469 procedure ends when all jobs have been reinserted into the sequence.

470 5. Computational evaluations

471 We proceed with the details of the generated benchmark and the experi-
 472 mental setting for the computational evaluation. We will detail the competing
 473 methods that have been reimplemented and adapted to the HFSDW problem.
 474 All algorithms (proposed and competing) are calibrated and then evaluated
 475 in a complete computational campaign.

476 5.1. Benchmark, competing methods and experimental setting

477 While there exist benchmarks for HFS problems, the special feature of
 478 the HFSDW problem considered in this paper (the due windows) calls for
 479 special attention as regards due dates. Therefore, we have generated a large
 480 benchmark with four sets of instances. The first two are the small and large
 481 instance sets. The last two are the corresponding small and large calibration
 482 instances. We separate between test and calibration in order to avoid bias in
 483 the calibration. We control five factors in the instance generation: number of
 484 jobs n , number of stages m , number of identical parallel machines per stage m_i ,
 485 Tardiness Factor (T), Due Date Range (R) and the width of the due window
 486 with respect to the due date (W). Processing times are uniformly distributed
 487 in the range $U[1, 99]$ and earliness and tardiness weights in the range $U[1, 9]$ as
 488 it is common in the scheduling literature. Following Potts and Van Wassenhove
 489 (1982), the due dates are generated with a random uniform distribution as
 490 $d_j = \max\left(0, U\left[\lfloor P(1 - T - R/2) \rfloor, \lfloor P(1 - T + R/2) \rfloor\right]\right)$ where, $\lfloor x \rfloor$ indicates
 491 that x is rounded to the nearest integer. This procedure is common in the
 492 scheduling literature with due date based criteria, as shown in Vallada et al.

493 (2008). Depending on the values of T and R , the generated due dates range
 494 from easy to satisfy to hard to meet. P is a makespan lower bound. We have
 495 used the existing state-of-the-art bounds for the HFS, which are proposed
 496 by Hidri and Haouari (2011). In that paper the authors review and compare
 497 all effective bounds for the HFS and propose new ones. We use all these
 498 bounds for generating the makespan lower bound, i.e., LB^1, LB^2, \dots, LB^7 ,
 499 LB_{VHHL} and LB_{RER} . Note that this last bound uses a complex revisited
 500 energetic reasoning procedure which needs a substantial CPU time in the
 501 large instances. The final lower bound is the maximum among these 9. With
 502 the due date we generate the due window as: $d_j^- = \max(0, \lfloor d_j - d_j \cdot H/100 \rfloor)$
 503 and $d_j^+ = \max(0, \lfloor d_j + d_j \cdot H/100 \rfloor)$ where $H = U[1, W]$, i.e., the due window
 504 is centered around d_j and has a maximum width that is $W\%$ the d_j .

505 The combinations of the factors used for the small instances set are: $n =$
 506 $\{10, 15, 20\}$, $m = \{2, 3, 4\}$, $m_i = \{2, 3\}$, $T = \{0.2, 0.4, 0.6\}$, $R = \{0.2, 0.6, 1.0\}$
 507 and $W = \{10, 20\}$. We generate 5 replicates per combination with different
 508 seeds resulting in a total of $3 \cdot 3 \cdot 2 \cdot 3 \cdot 3 \cdot 2 \cdot 5 = 1620$ small instances.
 509 For the large instances, the factors T , R and W are used with the same
 510 combinations, however, the values of the other three factors are changed as:
 511 $n = \{50, 100, 150, 200\}$, $m = \{5, 10\}$ and $m_i = \{5, 10\}$. with 5 replicates so
 512 the total number of large instances is $4 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 2 \cdot 5 = 1440$.

513 The calibration benchmarks are simply generated by selecting, at random,
 514 one value of n , m , T , R and W from the sets of values above and creating a new
 515 different instance. This is repeated 10% of times the size of each set. Therefore,
 516 there are 162 small calibration instances and 144 large calibration instances.
 517 All instances, along with the best complete solutions found in this paper and
 518 makespan bounds are available to download at <http://soa.iti.es>.

519 From the literature on HFS we have selected the best 9 existing algorithms:
 520 1) the Artificial Immune System of Engin and Döyen (2004) (AIS), 2) the
 521 Genetic Algorithm of Ruiz and Maroto (2006) (GA_R), 3) the Ant Colony
 522 Optimization of Alaykýran et al. (2007) (ACO), 4) the Genetic Algorithm
 523 of Kahraman et al. (2008) (GA_K), 5) The Improved Simulated Annealing of
 524 Naderi et al. (2009) (HSA), 6) Ant Colony Optimization of Khalouli et al.
 525 (2010) (ACO_K), the 7) the Iterated Local Search of Naderi et al. (2010)
 526 (ILS_N), 8) The Discrete Colonial Competitive Algorithm of Behnamian and
 527 Zandieh (2011) (DCCA) and 9) The Artificial Bee Colony of Pan et al. (2013)
 528 (ABC_P). The selection of these methods has several motivations. First of
 529 all they are all recent and competitive methods. Second, they represent the
 530 best performance from different and varied techniques. Lastly, they have not

531 been compared (at least not all of them) before. All these methods have been
532 carefully reimplemented. They have been adapted to the HFSDW problem. It
533 is important to note that all methods include the proposed idle time insertion
534 procedure. Together with these methods we will test the proposed ILS and
535 IG procedures in two versions: the ones with the existing acceptance criterion
536 (ILS and IG) and the ones with the θ -tournament alternative (ILST and
537 IGT).

538 All methods are coded in C++ and compiled with Visual Studio 2013.
539 The methods share most important functions in the code. The calibration
540 experiments in this paper are performed on Virtual Windows XP machines
541 with 1 virtual processor and 1 GByte of RAM memory. A virtualization server
542 composed of 30 blades is employed. Each blade runs two Intel XEON E5420
543 processors running at 2.5 GHz. and 16 GBytes of RAM memory. For the
544 final experiments where the calibrated methods are compared, we switched
545 to Virtual Windows 7 machines with 1 virtual processor and 2 GBytes of
546 RAM memory. An updated virtualization server of 8 blades, each one with
547 four AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz. and 256
548 GBytes or RAM is used. The reason behind this machine change is just a
549 migration of servers while this research was being carried out.

550 5.2. Calibration of the proposed and competing methods

551 The 9 competing methods and the 4 proposed algorithms are calibrated.
552 For the proposed ILS, ILST, IG and IGT the common factors to calibrate
553 are the maximum number of iterations in the VND local search $loop_{\max}$
554 tested at 3 levels $\{100, 200, 300\}$ and the usage of the second stage local
555 search. We call this second factor 2LS and it is tested at 2 variants, yes
556 and no. ILS and ILST share the factors ω at 3 levels $\{100, 200, 300\}$ and ϖ
557 at 4 levels $\{1, 10, 30, 50\}$. On the other hand, IG and IGT share only the
558 number of destructed jobs d which is tested at 4 levels $\{1, 2, 3, 4\}$. Finally,
559 the proposed methods with the usual temperature based acceptance criterion
560 (ILS and IG) share T tested at levels $\{1, 3, 5, 7\}$ and the two methods with
561 the tournament acceptance (ILST and IGT) share θ at 4 levels $\{2, 3, 4, 5\}$.
562 Therefore, ILS and ILST have 288 combinations and IG and IGT 96. We
563 employ the Design of Experiments (DOE) methodology. The setting is four
564 full factorial experiments, one per algorithm. We separate the results for
565 the small and large calibration sets so there are 8 experiments in total. The
566 instance factors (n , m , T , R and W) are not controlled in order to have an
567 overall picture of performance and to avoid instance-specific calibration. We

568 run each treatment five different times with each instance. The total number
569 of results is therefore $288 \cdot (162 + 144) \cdot 5 = 440,640$ for ILS and ILST and
570 $96 \cdot (162 + 144) \cdot 5 = 146,880$. The grand total of results is 1,175,040 among
571 all four proposed methods.

572 The response variable is the Relative Deviation Index (*RDI*) calculated as
573 $\frac{Method_{sol} - Best_{sol}}{Worst_{sol} - Best_{sol}} \cdot 100$, where $Best_{sol}$ and $Worst_{sol}$ are the best and the worst
574 solutions obtained throughout the calibrations for any instance respectively,
575 and $Method_{sol}$ is the solution given by a particular combination of parameters
576 of an algorithm. An *RDI* between 0 and 100 is obtained. If there is a case where
577 $Worst_{sol} = Best_{sol}$ then *RDI* is replaced by 0 since it means that all methods
578 in all replicates resulted in the same value. All results are examined with the
579 Analysis of Variance (ANOVA) procedure which is a powerful parametric
580 statistical tool. As such, all hypotheses were checked and met. ANOVA has
581 been extensively used in the last 10 years in the scheduling literature to
582 calibrate methods with success. Finally, even though ANOVA is a powerful
583 technique and we are using a large number of results, the process is by no
584 means a fine calibration. After all, a full factorial is no more than a type of
585 screening design. Far more exhaustive techniques for algorithm calibration
586 are explained in Bartz-Beielstein et al. (2010).

587 All methods included in this paper need a stopping criterion. While many
588 authors stop algorithms after a number of iterations have elapsed (also
589 generations, or solution evaluations) we think it is much more precise to stop
590 after a predefined CPU time has elapsed. After all, the CPU time is the
591 budget that a method has for spending. Stating that a given algorithm or
592 configuration A is better than B if at the same time A needs much more CPU
593 time is at best, inconclusive. The real question is, if given the same CPU time,
594 is A still better than B? Therefore, we stop the methods in the calibration after
595 a predefined CPU time that is equal to $t = 30nm$ milliseconds. Note that this
596 CPU time also adjusts with the instance size so that larger instances have more
597 time than smaller ones. We will not show the full ANOVA tables due to space
598 constraints. The full calibration details of the proposed methods are available
599 as on-line materials. Some means plots with 95% Tukey's Honest Significant
600 Difference (HSD) confidence intervals are shown in Figure 8 for the large
601 instances. Overlapping confidence intervals indicate statistical insignificance
602 in the observed differences of the response variable among the overlapped
603 averages. As we expected, the number of perturbations ω strongly interacts
604 with the number of solutions generated and perturbed (ϖ). We see that small
605 perturbations of $\omega = 1$ or large perturbations of $\omega = 3$ do not yield good

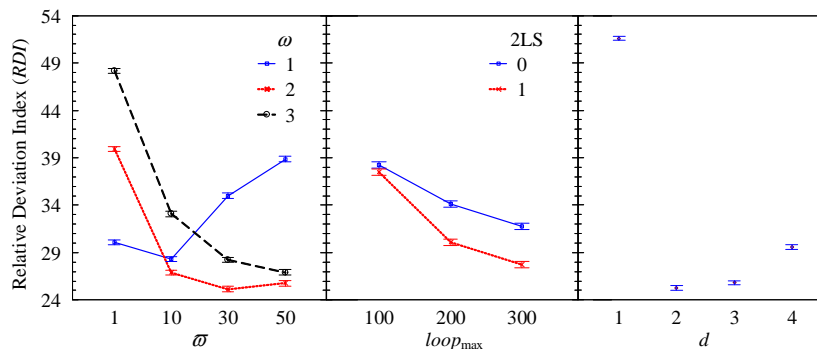


Figure 8: Means plot for the ANOVA large calibration experiment. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals. To the left the interaction between ω and ϖ for ILST. At the center the interaction between 2LS and $loop_{\max}$ for IG and to the right the factor d for IGT.

606 results. Also, we see that when $\omega = 1$ generating many perturbed solutions
607 ($\varpi = 30$) only wastes CPU time. The best is a combination of two movements
608 and generating 30 perturbed solutions to explore around the current solution.
609 The plot is very similar for ILS. Also we see in the middle plot of Figure 8
610 how the second stage local search (2LS= 1 or “yes”) improves solutions
611 considerably. However, it needs a large $loop_{\max}$ value of 200 or 300 since
612 for a value of 100 the effect of the second stage local search is small. The
613 fine local search is expensive and should only be applied to solutions that
614 have already been improved by the VND procedure. From these plots we see
615 that the improved perturbation for the ILS and the two stage local search,
616 provide very good results. Lastly, the plot on the right of Figure 8 shows that
617 destroying only one element in the IG methods results in bad performance. A
618 value of $d = 2$ seems better, which is in line with the perturbation value in
619 the ILS. Similar to most related studies (Ruiz and Stützle, 2007), the factor
620 T is not statistically significant. As a final note, the selected factor levels
621 after the calibration are for ILS: $\omega = 2/2$, $\varpi = 10/30$, $loop_{\max} = 100/300$,
622 2LS=yes/yes and $T = 7/7$ where the / separates the selected values for small
623 and large bechmarks, respectively. The sets of values for ILST, IG and IGT
624 are ($\omega = 2/2$, $\varpi = 10/30$, $loop_{\max} = 100/300$, 2LS=yes/yes and $\theta = 4/3$),
625 ($loop_{\max} = 100/300$, 2LS=yes/yes, $T = 7/7$, $d = 4/3$) and ($loop_{\max} = 100/300$,
626 2LS=yes/yes, $d = 4/2$, $\theta = 4/2$), respectively.

627 As for the competing methods we have comprehensively calibrated each
628 one of them following an identical procedure. However, no details are given

629 here due to space considerations. Instead, we have put together the results
630 of the calibration as on-line materials. In these materials all experiments
631 and results are detailed. For example, the GA_R of Ruiz and Maroto (2006)
632 was calibrated with a full factorial design of 240 treatments and the GA_K
633 of Kahraman et al. (2008) with 192. In total, also considering the proposed
634 methods, we have tested 1,236 treatments for small and the same number for
635 the large calibration benchmark. All this required a total of almost 342 CPU
636 days of computation.

637 5.3. Evaluation of results and statistical analyses

638 For the final evaluation we use the test benchmark with 1620 small and
639 1440 large instances. All 13 methods are run 10 different times with different
640 random seeds with each instance. In order to have a broader picture of
641 performance, we use three different stopping times as ρnm milliseconds where
642 $\rho = \{30, 60, 90\}$. This translates into 0.6 seconds in the smallest instances of
643 10 jobs and 2 stages when $\rho = 30$ to 180 seconds for the largest instances of
644 200 jobs and 10 stages if $\rho = 90$. To avoid self-correlation in the results, for
645 each stopping time each method is run independently from the beginning. All
646 methods have been coded in the same programming language, share identical
647 operating environments and most important functions, like objective function
648 evaluation and idle insertion and are run for the same CPU time on the
649 same computer. As a result, the comparison is fair and the results completely
650 comparable.

651 Considering that each one of the 13 test methods is run with 1620 small
652 and 1440 large instances, 10 different times and with 3 different CPU time
653 limits we have a total of $13 \cdot (1620 + 1440) \cdot 10 \cdot 3 = 1,193,400$ experimental
654 results. Very strong conclusions can be drawn from such a large dataset. The
655 CPU time needed for all experimentation was almost 387 CPU days. The
656 average relative deviation index, grouped by CPU time ρ , number of jobs n
657 and number of stages m is given in Table 1 for the small instances. The same
658 table but for the large instances is given in Table 2. In both tables each cell is
659 the average *RDI* over 1,800 results (combinations of m_i , T , R , W , instance
660 replicate and 10 replicates per algorithm). The complete spreadsheets with
661 all detailed results are available upon request from the authors.

662 Some of the adapted methods do not produce good results, despite cal-
663 ibration and insertion of idle times. This is the case for ACO and ACO_K .
664 DCCA and AIS are also far from the best results. GA_R , despite being among
665 the oldest methods tested, shows a relatively good performance. The recent

ρ	n	m	ABC _P	ACO	ACO _K	AIS	DCCA	GA _K	GA _R	HSA	ILS _N	IG	IGT	ILS	ILST	
30	10	2	1.67	85.18	25.91	3.26	4.59	2.86	1.95	1.60	1.59	0.76	0.76	0.82	0.81	
		3	2.60	83.06	43.40	5.05	5.81	4.10	3.00	2.50	2.43	1.11	1.11	1.14	1.11	
		4	1.54	70.72	58.98	3.78	4.68	3.18	2.02	1.61	1.46	0.70	0.70	0.69	0.70	
	15	2	1.56	87.81	39.85	4.19	6.33	2.81	2.36	1.00	0.95	0.36	0.36	0.36	0.38	
		3	1.71	77.19	68.32	4.90	6.96	3.53	2.57	1.03	1.00	0.42	0.41	0.41	0.42	
		4	1.36	57.73	80.72	4.38	5.69	3.23	2.35	0.81	0.80	0.32	0.30	0.32	0.29	
	20	2	2.54	89.78	46.27	5.99	7.82	3.68	2.87	0.84	0.83	0.44	0.41	0.46	0.46	
		3	2.31	76.57	72.80	5.85	7.09	3.82	3.01	1.02	1.06	0.57	0.55	0.63	0.59	
		4	2.04	52.98	84.34	5.15	5.76	3.15	2.91	0.91	0.96	0.54	0.51	0.57	0.54	
	60	10	2	1.66	83.74	22.35	3.14	4.26	2.72	1.82	1.55	1.55	0.70	0.73	0.74	0.72
			3	2.57	82.01	37.10	4.88	5.48	4.16	2.78	2.46	2.43	1.05	1.06	1.07	1.08
			4	1.51	69.32	51.91	3.54	4.37	3.04	1.86	1.52	1.45	0.63	0.63	0.65	0.65
15		2	1.32	87.23	36.26	3.58	5.70	2.66	2.06	0.86	0.86	0.25	0.26	0.23	0.25	
		3	1.48	76.65	63.30	4.30	6.72	3.27	2.23	0.96	0.92	0.31	0.30	0.30	0.31	
		4	1.18	57.16	74.97	3.87	5.50	2.99	2.01	0.72	0.71	0.21	0.22	0.20	0.20	
20		2	1.95	89.37	43.00	5.01	6.40	3.09	2.33	0.65	0.64	0.27	0.26	0.27	0.29	
		3	1.87	76.21	68.30	5.09	6.54	3.27	2.53	0.81	0.84	0.39	0.36	0.42	0.40	
		4	1.68	52.49	79.27	4.42	5.45	2.66	2.51	0.75	0.79	0.39	0.36	0.40	0.39	
90		10	2	1.65	83.11	20.54	3.01	4.14	2.77	1.82	1.52	1.54	0.67	0.68	0.70	0.68
			3	2.53	81.27	34.02	4.72	5.35	4.07	2.73	2.43	2.43	1.02	1.04	1.03	1.06
			4	1.50	68.54	47.69	3.42	4.28	3.08	1.79	1.49	1.45	0.58	0.59	0.61	0.60
	15	2	1.23	86.70	34.40	3.32	5.61	2.47	1.88	0.81	0.82	0.21	0.22	0.19	0.19	
		3	1.36	76.23	60.56	4.10	6.64	3.16	2.09	0.92	0.89	0.24	0.25	0.25	0.25	
		4	1.11	56.78	72.04	3.71	5.43	2.72	1.87	0.69	0.68	0.17	0.19	0.16	0.17	
	20	2	1.68	89.11	41.41	4.57	6.18	2.90	2.10	0.57	0.58	0.21	0.20	0.20	0.23	
		3	1.66	75.93	65.79	4.67	6.41	2.97	2.32	0.73	0.77	0.31	0.29	0.33	0.32	
		4	1.53	52.28	76.87	4.11	5.37	2.45	2.34	0.68	0.72	0.31	0.30	0.32	0.32	
	Average			1.73	75.01	53.72	4.30	5.72	3.14	2.30	1.16	1.15	0.49	0.48	0.50	0.50

Table 1: Average Relative Deviation Index (RDI) for small instances. Best results in bold.

666 methods ABC_P , HSA and ILS_N show good results. We see that the four
667 proposed approaches clearly dominate all other proposals in the small in-
668 stances and also dominate all other methods, except possibly HSA in the
669 large instances. Looking at average results for the large instances, the best
670 proposed method is ILST with an overall RDI of 0.31. The closest competing
671 method is HSA with 0.41, this means that ILST is approximately 32% better
672 than HSA. Furthermore, the ILST approach is much simpler and has less
673 calibration parameters than HSA. Finally, from the 43,200 results given in
674 the large instances (1440 instances, 3 termination criteria and 10 replicates)
675 ILST gives a better solution than HSA in 28,040 cases, HSA is better than
676 ILST in 15,129 and both tie in 31 cases.

ρ	n	m	ABC_P	ACO	ACO_K	AIS	DCCA	GA_K	GA_R	HSA	ILS_N	IG	IGT	ILS	ILST	
30	50	5	2.54	22.75	92.21	15.34	5.55	3.28	2.54	0.85	0.99	0.65	0.73	0.69	0.64	
		10	1.38	6.54	94.88	5.31	2.15	1.35	1.23	0.43	0.50	0.37	0.38	0.39	0.35	
	100	5	2.93	14.93	94.58	12.53	4.39	3.25	1.86	0.65	0.75	0.61	0.64	0.52	0.51	
		10	1.26	4.66	96.84	4.18	1.63	1.14	0.88	0.25	0.29	0.25	0.27	0.26	0.24	
	150	5	2.85	12.22	95.78	11.08	4.09	3.65	1.65	0.80	0.89	0.78	0.77	0.60	0.59	
		10	1.11	4.00	97.64	3.82	1.53	1.17	0.72	0.21	0.24	0.27	0.29	0.25	0.24	
	200	5	2.63	10.92	96.44	10.02	4.10	3.96	1.53	0.96	1.07	0.75	0.76	0.60	0.60	
		10	1.04	3.68	97.97	3.55	1.55	1.27	0.67	0.25	0.28	0.35	0.36	0.30	0.29	
	60	50	5	1.83	22.67	88.91	14.44	5.30	2.47	2.21	0.71	0.86	0.47	0.50	0.53	0.46
			10	1.07	6.53	92.85	5.07	2.05	1.04	1.12	0.37	0.45	0.29	0.27	0.31	0.26
100		5	2.04	14.89	92.24	12.01	4.24	2.24	1.61	0.42	0.50	0.41	0.41	0.36	0.33	
		10	0.93	4.65	95.57	4.03	1.57	0.85	0.80	0.20	0.23	0.18	0.17	0.19	0.16	
150		5	2.20	12.18	94.11	10.87	3.99	2.49	1.42	0.45	0.55	0.56	0.55	0.39	0.36	
		10	0.86	3.99	96.63	3.75	1.47	0.84	0.65	0.14	0.17	0.18	0.18	0.17	0.15	
200		5	2.11	10.89	95.03	9.87	4.01	2.74	1.26	0.56	0.64	0.57	0.56	0.40	0.39	
		10	0.83	3.67	97.12	3.49	1.50	0.89	0.60	0.14	0.17	0.25	0.26	0.20	0.19	
90		50	5	1.57	22.60	87.14	13.98	5.03	2.17	2.06	0.65	0.79	0.40	0.40	0.45	0.39
			10	0.93	6.52	91.56	4.93	1.95	0.93	1.07	0.34	0.42	0.25	0.22	0.27	0.22
	100	5	1.59	14.87	91.05	11.70	4.08	1.82	1.51	0.34	0.41	0.31	0.29	0.28	0.24	
		10	0.77	4.65	94.81	3.95	1.51	0.73	0.76	0.17	0.21	0.14	0.13	0.16	0.13	
	150	5	1.82	12.16	93.06	10.77	3.87	1.96	1.31	0.31	0.39	0.44	0.42	0.29	0.25	
		10	0.72	3.99	96.06	3.70	1.42	0.70	0.62	0.11	0.14	0.14	0.14	0.14	0.12	
	200	5	1.80	10.87	94.17	9.78	3.89	2.15	1.15	0.36	0.44	0.46	0.45	0.30	0.28	
		10	0.71	3.67	96.67	3.46	1.45	0.72	0.56	0.10	0.13	0.20	0.20	0.16	0.14	
	Average			1.56	9.94	94.31	7.98	3.01	1.83	1.24	0.41	0.48	0.39	0.39	0.34	0.31

Table 2: Average Relative Deviation Index (RDI) for large instances. Best results in bold.

677 Finally, we mention the effect of the proposed θ -tournament operator. The
678 versions with this operator (IGT and ILST) seem to either give the same
679 results, or, in the case of the large instance and ILST, better results than
680 those with the regular temperature based acceptance criterion (IG and ILS)
681 but the difference is not large. This needs careful examination. We are going

682 to check for statistical relevance of the averages shown in Tables 1 and 2. We
 683 carry out a multi-factor ANOVA where all the instance factors, as well as
 684 ρ and the type of algorithm are controlled. In the experiment for the small
 685 instances the methods ACO and ACO_K had to be removed as their deviations
 686 were so large that they were creating normality problems. Similarly, for the
 687 large instances, ACO, ACO_K and AIS were removed as well. The results of
 688 the experiment are sound. **For the small instances, the differences observed
 689 in the four proposed methods IG, IGT, ILS and ILST are not statistically
 690 significant and all have a statistically equivalent performance. However, all of
 691 them are statistically better than the closest competitors HSA and ILS_N.**

692 The average *RDI* values are statistically different for all the algorithms
 693 in the large instances. . Figure 9 shows the interaction between the four
 694 proposed algorithms and ρ on the left and a zoom for the four proposed
 methods on the right for the large instances. The statistical test confirms

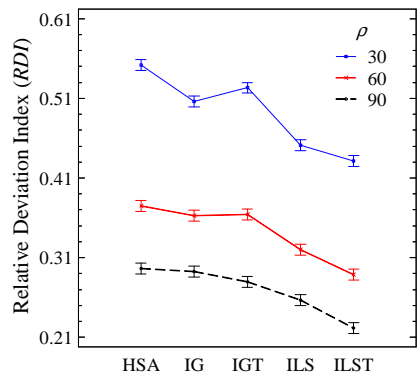


Figure 9: Interaction between the proposed methods, HSA and stopping criterion (ρ). All means with Tukey's Honest Significant Difference (HSD) 95% confidence intervals. Large instances.

695 that the closest competitor gives results that are statistically worse than the
 696 presented methods. As a matter of fact, it should be stressed that any of our
 697 proposed methods with the shortest processing time of $\rho = 30$ give better
 698 solutions than HSA with the largest time of $\rho = 90$. We can also observe in
 699 the zoom that the effect of the θ -tournament becomes statistically significant
 700 for the largest CPU times of $\rho = 90$. As a conclusion, ILS is slightly better
 701 than IG and in turn the θ -tournament also provides an advantage for large
 702 CPU times. Therefore, ILST is the best proposed method.
 703

704 6. Conclusions and future research

705 In this paper we have studied hybrid flowshop problems with due windows
706 with the minimization of the total weighted earliness and tardiness from the
707 due window. The literature review has shown that this problem has scarcely
708 been studied. After presenting and formalizing the problem we have proposed
709 an Iterated Local Search and Iterated Greedy procedures. These methods
710 are simple and have few parameters. Specific to earliness tardiness objectives
711 is the insertion of idle time before tasks. We have also devised an optimal
712 procedure that delays the completion of tasks by the exact amount necessary.
713 We have also presented a key procedure, which is a two stage local search. In
714 the first stage, a VND works with an indirect representation of a solution. In
715 the second stage, we identify the most promising movements and work over
716 the exact representation. This proposal, together with improvements in the
717 acceptance criterion and perturbation constitute the core of our algorithmic
718 contribution. In our opinion, experimentation should be complete and thor-
719 ough. In order to compare our proposed methods we have reimplemented and
720 adapted 9 high performing competing methods. Then, all procedures have
721 been calibrated using the DOE approach. A comprehensive benchmark of
722 more than 3000 instances has been used in a complete computational and
723 statistical evaluation to analyze the effect of the proposed contributions and
724 to compare with existing algorithms. Results have shown that the proposed
725 methods, particularly those with the new θ -tournament acceptance criterion,
726 produce the best results. There are several future research avenues. Introduc-
727 ing more realistic elements into the problem, like setup times, release dates, or
728 unrelated parallel machines at each stage is a possibility. Further exploration
729 of the effect of the different possible representations on other objectives seems
730 also promising. **Finally, we would like to call the attention of the scheduling**
731 **community as to how earliness penalties are usually calculated. Delaying the**
732 **last manufacturing stage (and therefore the completion time) of a job that**
733 **was started at a given point in time does not reduce earliness costs in most**
734 **real cases. What usually counts is when the job was started in the first stage**
735 **as at that moment resources are mobilized, raw materials used, etc. Therefore,**
736 **more sensible earliness costs penalties must be considered in order to make**
737 **the objective function more realistic.**

738 Acknowledgments

739 The authors would like to thank Professors Lofti Hidri and Mohamed
740 Haouari for sharing with us the source codes and explanations of the lower
741 bounds. Quan-Ke Pan is supported by the National Natural Science Founda-
742 tion of China (Grant No. 51575212), Program for New Century Excellent
743 Talents in University (Grant No. NCET-13-0106), Science Foundation of Hubei
744 Province in China (Grant No. 2015CFB560), Specialized Research Fund for
745 the Doctoral Program of Higher Education (Grant No. 20130042110035), Key
746 Laboratory Basic Research Foundation of Education Department of Liaon-
747 ing Province (LZ2014014), Open Research Fund Program of the State Key
748 Laboratory of Digital Manufacturing Equipment and Technology, Huazhong
749 University of Science and Technology, China. Rubén Ruiz and Pedro Alfaro-
750 Fernández are supported by the Spanish Ministry of Economy and Com-
751 petitiveness, under the project “SCHEYARD – Optimization of Scheduling
752 Problems in Container Yards” (No. DPI2015-65895-R) financed by FEDER
753 funds.

754 References

- 755 Alaykýran, K., Engin, O., and Döyen, A. (2007). Using ant colony optimization to solve hybrid
756 flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*,
757 35(5-6):541–550.
- 758 Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors (2010). *Experimental*
759 *Methods for the Analysis of Optimization Algorithms*. Springer, New York.
- 760 Behnamian, J., Fatemi Ghomi, S. M. T., and Zandieh, M. (2010a). Development of a hybrid
761 metaheuristic to minimise earliness and tardiness in a hybrid flowshop with sequence-dependent
762 setup times. *International Journal of Production Research*, 48(5):1415–1438.
- 763 Behnamian, J. and Zandieh, M. (2011). A discrete colonial competitive algorithm for hybrid
764 flowshop scheduling to minimize earliness and quadratic tardiness penalties. *Expert Systems*
765 *with Applications*, 38(12):14490–14498.
- 766 Behnamian, J. and Zandieh, M. (2013). Earliness and tardiness minimizing on a realistic hy-
767 brid flowshop scheduling with learning effect by advanced metaheuristic. *Arabian Journal for*
768 *Science and Engineering*, 38(5):1229–1242.
- 769 Behnamian, J., Zandieh, M., and Fatemi Ghomi, S. M. T. (2010b). Due windows group scheduling
770 using an effective hybrid optimization approach. *International Journal of Advanced Manufac-*
771 *turing Technology*, 46(5-8):721–735.
- 772 Chang, S.-C. and Liao, D.-Y. (1994). Scheduling flexible flow shops with no setup effects. *IEEE*
773 *Transactions on Robotics and Automation*, 10(2):112–122.
- 774 Cheng, T. C. E. (1988). Optimal common due-date with limited completion time deviation.
775 *Computers and Operations Research*, 15(2):91–96.
- 776 Engin, O. and Döyen, A. (2004). A new approach to solve hybrid flow shop scheduling problems
777 by artificial immune system. *Future Generation Computer Systems*, 20(6):1083–1095.
- 778 Fanjul-Peyro, L. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel
779 machine scheduling. *European Journal of Operational Research*, 207(1):55–69.

- 780 Finke, D. A., Medeiros, D. J., and Trabant, M. T. (2007). Multiple machine jit scheduling: a
781 tabu search approach. *International Journal of Production Research*, 45(21):4899–4915.
- 782 Framinan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Inte-*
783 *grated View on Models, Methods and Tools*. Springer, New York.
- 784 Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization
785 and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete*
786 *Mathematics*, 5:287–326.
- 787 Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Opera-*
788 *tional Research Society*, 39(4):359–364.
- 789 Hendel, Y. and Sourd, F. (2007). An improved earliness-tardiness timing algorithm. *Computers*
790 *and Operations Research*, 34(10):2931–2938.
- 791 Hidri, L. and Haouari, M. (2011). Bounding strategies for the hybrid flow shop scheduling problem.
792 *Applied Mathematics and Computation*, 217(21):8248–8263.
- 793 Huang, R.-H. and Yu, S.-C. (2013). Clarifying cutting and sewing processes with due windows
794 using an effective ant colony optimization. *Mathematical Problems in Engineering*, 2013:1–12.
- 795 Huang, R.-H., Yu, S.-C., and Kuo, C.-W. (2014). Reentrant two-stage multiprocessor flow shop
796 scheduling with due windows. *International Journal of Advanced Manufacturing Technology*,
797 71(5-8):1263–1276.
- 798 Janiak, A., Kozan, E., Lichtenstein, M., and Oğuz, C. (2007). Metaheuristic approaches to the
799 hybrid flow shop scheduling problem with a cost-related criterion. *International Journal of*
800 *Production Economics*, 105(2):407–424.
- 801 Jolai, F., Sheikh, S., Rabbani, M., and Karimi, B. (2009). A genetic algorithm for solving no-
802 wait flexible flow lines with due window and job rejection. *International Journal of Advanced*
803 *Manufacturing Technology*, 42(5-6):523–532.
- 804 Jun, S. and Park, J. (2015). A hybrid genetic algorithm for the hybrid flow shop scheduling prob-
805 lem with nighttime work and simultaneous work constraints: A case study from the transformer
806 industry. *Expert Systems with Applications*, 42(15-16):6196–6204.
- 807 Kahraman, C., Engin, O., Kaya, İ., and Yilmaz, M. K. (2008). An application of effective ge-
808 netic algorithms for solving hybrid flow shop scheduling problems. *International Journal of*
809 *Computational Intelligence Systems*, 1(2):134–147.
- 810 Kedad-Sidhoum, S. and Sourd, F. (2010). Fast neighborhood search for the single machine
811 earliness-tardiness scheduling problem. *Computers and Operations Research*, 37(8):1464–1471.
- 812 Khalouli, S., Ghedjati, F., and Hamzaoui, A. (2010). A meta-heuristic approach to solve a JIT
813 scheduling problem in hybrid flow shop. *Engineering Applications of Artificial Intelligence*,
814 23(5):765–771.
- 815 Linn, R. and Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers and Industrial*
816 *Engineering*, 37(1-2):57–61.
- 817 Liu, C.-Y. and Chang, S.-C. (2000). Scheduling flexible flow shops with sequence-dependent setup
818 effects. *IEEE Transactions on Robotics and Automation*, 16(4):408–419.
- 819 Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). Iterated local search: Framework and
820 applications. In Gendreau, M. and Potvin, J. Y., editors, *Handbook of Metaheuristics*, vol-
821 ume 14 of *International Series in Operations Research & Management Science*, chapter 12,
822 pages 363–397. Kluwer Academic Publishers, Norwell, MA, USA, second edition.
- 823 MacCarthy, B. L. and Liu, J. (1993). Addressing the gap in scheduling research: a review of opti-
824 mization and heuristic methods in production scheduling. *International Journal of Production*
825 *Research*, 31(1):59–79.
- 826 McKay, K. N., Pinedo, M., and Webster, S. (2002). Practice-focused research issues for scheduling
827 systems. *Production and Operations Management*, 11(2):249–258.
- 828 Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Com-*
829 *puters and Operations Research*, 37(4):754–768.
- 830 Naderi, B., Ruiz, R., and Zandieh, M. (2010). Algorithms for a realistic variant of flowshop

- 831 scheduling. *Computers and Operations Research*, 37(2):236–246.
- 832 Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A., and Roshanaei, V. (2009). An improved
833 simulated annealing for hybrid flowshops with sequence-dependent setup and transportation
834 times to minimize total completion time and total tardiness. *Expert Systems with Applications*,
835 36(6):9625–9633.
- 836 Pan, Q.-K. and Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle
837 permutation flowshop scheduling problem. *OMEGA, the International Journal of Management
838 Science*, 44(1):41–50.
- 839 Pan, Q.-K., Wang, L., Li, J.-Q., and Duan, J.-H. (2014). A novel discrete artificial bee colony
840 algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *OMEGA,
841 the International Journal of Management Science*, 45:42–56.
- 842 Pan, Q.-K., Wang, L., Mao, K., Zhao, J.-H., and Zhang, M. (2013). An effective artificial bee
843 colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE
844 Transactions on Automation Science and Engineering*, 10(2):307–322.
- 845 Pan, Q.-K., Wang, L., and Zhao, B. H. (2008). An improved iterated greedy algorithm for the no-
846 wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced
847 Manufacturing Technology*, 38(7-8):778–786.
- 848 Pinedo, M. (2012). *Scheduling: Theory, Algorithms and Systems*. Springer, New York, fourth
849 edition.
- 850 Potts, C. N. and Van Wassenhove, L. N. (1982). A decomposition algorithm for the single machine
851 total tardiness problem. *Operations Research Letters*, 1(5):177–181.
- 852 Quadt, D. and Kuhn, D. (2007). A taxonomy of flexible flow line scheduling procedures. *European
853 Journal of Operational Research*, 178(3):686–698.
- 854 Ribas, I., Companys, R., and Tort-Martorell, X. (2011). An iterated greedy algorithm for the flow-
855 shop scheduling problem with blocking. *OMEGA, The International Journal of Management
856 Science*, 39(3):293–301.
- 857 Ribas, I., Leisten, R., and Framinan, J. M. (2010). Review and classification of hybrid flow
858 shop scheduling problems from a production system and a solutions procedure perspective.
859 *Computers and Operations Research*, 37(8):1439–1454.
- 860 Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop
861 heuristics. *European Journal of Operational Research*, 165(2):479–494.
- 862 Ruiz, R. and Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent
863 setup times and machine eligibility. *European Journal of Operational Research*, 169(3):781–800.
- 864 Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permu-
865 tation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–
866 2049.
- 867 Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European
868 Journal of Operational Research*, 205(1):1–18.
- 869 Sabuncuoglu, I. and Lejmi, T. (1999). Scheduling for non regular performance measure under the
870 due window approach. *OMEGA, the International Journal of Management Science*, 27(5):555–
871 568.
- 872 Sakuraba, C. S., Ronconi, D. P., and Sourd, F. (2009). Scheduling in a two-machine flowshop for
873 the minimization of the mean absolute deviation from a common due date. *Computers and
874 Operations Research*, 36(1):60–72.
- 875 Sheikh, S. (2013). Multi-objective flexible flow lines with due window, time lag, and job rejection.
876 *International Journal of Advanced Manufacturing Technology*, 64(9-12):1423–1433.
- 877 Stützle, T. (1998). Applying iterated local search to the permutation flow shop problem. Technical
878 Report AIDA-98-04, FG Itellektik, FB Informatik, TU Darmstadt.
- 879 Tasgetiren, M., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G. (2007). A particle swarm optimiza-
880 tion algorithm for makespan and total flowtime minimization in the permutation flowshop
881 sequencing problem. *European Journal of Operational Research*, 177(3):1930–1947.

- 882 Tseng, C.-T. and Liao, C.-J. (2008). A discrete particle swarm optimization for lot-streaming
883 flowshop scheduling problem. *European Journal of Operational Research*, 191(2):360–373.
- 884 Urlings, T. and Ruiz, R. (2007). Local search in complex scheduling problems. In Stützle, T.,
885 Birattari, M., and Hoos, H. H., editors, *Engineering Stochastic Local Search Algorithms. De-*
886 *signing, Implementing and Analyzing Effective Heuristics*, volume 4638 of *Lecture Notes in*
887 *Computer Science*, pages 202–206. Springer-Verlag, Brussels, Belgium.
- 888 Urlings, T., Ruiz, R., and Sivrikaya-Şerifoğlu, F. (2010a). Genetic algorithms for complex hybrid
889 flexible flow line problems. *International Journal of Metaheuristics*, 1(1):30–54.
- 890 Urlings, T., Ruiz, R., and Stützle, T. (2010b). Shifting representation search for hybrid flexible
891 flowline problems. *European Journal of Operational Research*, 207(2):1086–1095.
- 892 Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m -machine
893 flowshop problem: a review and evaluation of heuristics and metaheuristics. *Computers and*
894 *Operations Research*, 35(4):1350–1373.
- 895 Vignier, A., Billaut, J.-C., and Proust, C. (1999). Les problèmes d’ordonnement de type
896 flow-shop hybride: État de l’art. *RAIRO Recherche opérationnelle*, 33(2):117–183. (in French).
- 897 Wang, H. (2005). Flexible flow shop scheduling: optimum, heuristics and artificial intelligence
898 solutions. *Expert Systems*, 22(2):78–85.
- 899 Yeung, W. K., Oğuz, C., and Cheng, T. C. E. (2004). Two-stage flowshop earliness and tardiness
900 machine scheduling involving a common due window. *International Journal of Production*
901 *Economics*, 90(4):421–434.