European Journal of Operational Research 266 (2018) 950–962

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

Heuristics for a flowshop scheduling problem with stepwise job objective function

Luciana S. Pessoa[a,*], Carlos E. Andrade[b]

[a] Department of Industrial Engineering, PUC-Rio, Rua Marquês de São Vicente, 225, Gávea 22453-900, Rio de Janeiro (RJ), Brazil
[b] AT&T Labs Research, 200 South Laurel Avenue, Middletown, NJ 07748, USA

ARTICLE INFO

Article history:
Received 2 January 2017
Accepted 21 October 2017
Available online 31 October 2017

Keywords:
Flowshop scheduling
Heuristics
Metaheuristics

ABSTRACT

In this work, we introduce the Flowshop Scheduling Problem with Delivery Dates and Cumulative Payoffs. This problem is a variation of the flowshop scheduling problem with job release dates that maximizes the total payoff with a stepwise job objective function. This paper contributes towards proposing a mathematical formulation for this new problem and an original constructive heuristic. Additionally, we develop a new benchmark of 300 hard instances which are available in a public repository. Besides, we provide primal and dual bounds for them. Extensive computational experiments were carried out taking into account classical constructive heuristics and hybrid local searches. Results show the merit of the FF heuristic when compared to other classical heuristics for flowshop scheduling problems. Additionally, we compare an Iterated Local Search (ILS), an Iterated Greedy Search (IGS), and a Biased Random-Key Genetic Algorithm (BRKGA), both customized for the studied problem, and a commercial mixed integer programming solver. The comparison between these methods showed that the BRKGA starting with a solution from FF heuristic is able to find the best solutions in a very short period of time. Iterated local Search and the commercial solver presented significantly worse results than the aforementioned methods.

1. Introduction

In a flowshop scheduling problem, there is a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that must be sequentially processed on a set of m machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Each job $J_j \in \mathcal{J}$ has a processing time $p_{j,\ell} > 0$ on machine $M_\ell \in \mathcal{M}$. A schedule S is defined as a vector of starting times of the jobs on each machine: $S = (s_{1,1}, \ldots, s_{1,m}, s_{2,1}, \ldots, s_{2,m}, \ldots, s_{n,1}, \ldots, s_{n,m})$, where $s_{j\ell}$ is the starting time of job J_j on machine M_ℓ in schedule S. Given a schedule S, we denote by $C_{j\ell}(S)$ ($C_{j\ell}$ for short) the completion time of job J_j on machine M_ℓ: $C_{j\ell} = s_{j\ell} + p_{j\ell}$. For shortness, let us denote $C_{j\ell}$ by C_j as the time where job J_j finishes, assuming that machine M_ℓ is the last machine where J_j should be processed.

In the *Flowshop Scheduling Problem with Delivery Dates and Cumulative Payoffs*, each job $J_j \in \mathcal{J}$ has a release date $r_j \geq 0$. Moreover, a set of K delivery dates $\mathcal{D} = \{D_1, D_2, \ldots, D_K\}$ is given, where $0 < D_1 < \cdots < D_K$. Job preemption is not allowed and all parameters are integer numbers. The objective value of a given schedule S is equal to $\sum_{j=1}^{n} \mathcal{F}(C_j(S))$, where $\mathcal{F}(t)$ is a non-increasing step function on t. The problem consists in finding a maximum payoff scheduling S^* such that jobs do not overlap in a given machine in the same time, and are processed sequentially into machines $\mathcal{M}$. Extending the three-field notation of Graham, Lawler, Lenstra, and Kan (1979), the problem addressed in this work can be defined as $F|r_j, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$.

Although the payoff $\mathcal{F}$ can be any non-increasing step function, in this work we consider that this function is computed in the same way as for the single machine problem (Seddik, Gonzales, & Kedad-Sidhoum, 2013), i.e., given a set of K delivery dates $\mathcal{D} = \{D_1, D_2, \ldots, D_K\}$, we define:

$$\mathcal{F}(C_j) = \begin{cases} 0 & \text{if} \quad D_K < C_j \\ 1 & \text{if} \quad D_{K-1} < C_j \leq D_K \\ \vdots & \\ K & \text{if} \quad 0 < C_j \leq D_1. \end{cases} \tag{1}$$

Fig. 1 illustrates a schedule for a permutation flowshop whose instance is defined as follows:

- Number of machines $m = 3$, number of jobs $n = 4$, number of delivery dates $K = 3$.
- Release dates: $r_1 = 2,\ r_2 = 7,\ r_3 = 9,\ r_4 = 12$.
- Processing time tuples (in machine order): $p_1 = (1, 2, 2),\ p_2 = (3, 2, 2),\ p_3 = (3, 6, 2),\ p_4 = (5, 1, 4)$.
- Delivery dates: $D_1 = 16,\ D_2 = 21,\ D_3 = 26$.

* Corresponding author.
E-mail addresses: lucianapessoa@puc-rio.br (L.S. Pessoa), cea@research.att.com (C.E. Andrade).

https://doi.org/10.1016/j.ejor.2017.10.045
0377-2217/© 2017 Elsevier B.V. All rights reserved.

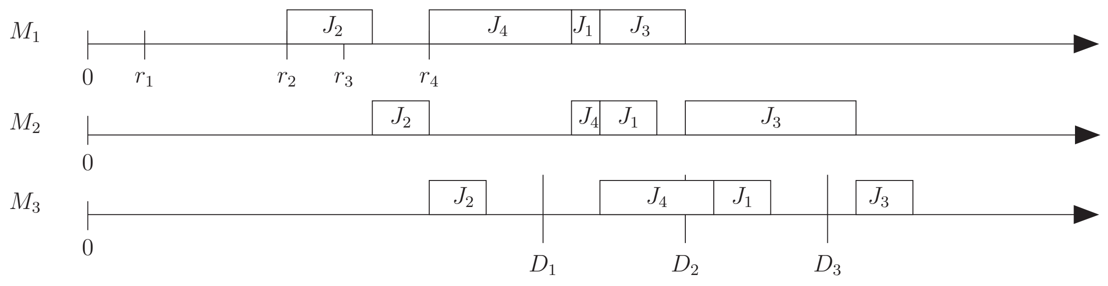


Fig. 1. A schedule for a permutation flowshop instance.

Note that since J_2 finishes before the first delivery date D_1, the payoff for J_2 is 3. Both J_1 and J_4 finishes after D_2 but before D_3 generating a payoff of 1 each. J_3 finishes after D_3 and gets a payoff of zero. The total payoff is 5.

A real application of this problem is found in the context of book digitization, as described by Seddik et al. (2013) who defined a related single machine problem. The book digitization process is linear and includes several steps (although two main ones: digitization and segmentation). While the single machine problem can model a bottleneck machine, a flowshop problem models more accurately the real machine configuration. This problem is defined from the perspective of a digitization firm hired to process many books. Each delivery date is a milestone when the payment of the firm is calculated regarding the books that were digitized in the last leap. The sooner the digitization of a book, the greater is the payoff associated to it. Besides, the calculus of a book payoff obey a step-function that decreases at each milestone. Each book is available from a release date and it is going to be processed during a certain time in each phase of digitization operation.

Note that since $\sum_{j=1}^{n} \mathcal{F}(C_j)$ is a regular criterion, left-shifted schedules are dominant, as defined by Seddik et al. (2013). Hence, a feasible left-shifted schedule can be represented simply by a sequence, since each machine process the jobs in the same order. For instance, the schedule of Fig. 1 is a left-shifted schedule that can be represented by the sequence (J_2, J_4, J_1, J_3). Hence, this problem can be modeled as an integer program with a positional variables formulation, as described below.

1.1. Formal definition

We propose a Mixed-Integer Linear Program (MILP) model for this problem, formulated with positional variables. According to Della Croce, Gupta, and Tadei (2011), this kind of formulation is better than other models based on disjunctive variables and constraints. Consider the binary decision variable X_{ij} such that $X_{ij} = 1$ if job J_i is the jth job of the sequence and $X_{ij} = 0$ otherwise. Variable $C_{j\ell}^{pos} \in \mathbb{Z}^+$ denotes the completion time of the job on the jth position on machine M_ℓ for $J_j \in \mathcal{J}$ and $M_\ell \in \mathcal{M}$. The $F|r_j, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$ can be formulated as following:

$$\max \quad \sum_{j=1}^{n} \mathcal{F}(C_{jm}^{pos}) \tag{2a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} X_{ij} = 1 \ \forall j \in \{1, \dots, n\}, \tag{2b}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \ \forall i \in \{1, \dots, n\}, \tag{2c}$$

$$C_{j1}^{pos} \geq \sum_{i=1}^{n} (p_{i1} + r_i) X_{ij} \ \forall j \in \{1, \dots, n\}, \tag{2d}$$

$$C_{j\ell}^{pos} \geq C_{j-1,\ell}^{pos} + \sum_{i=1}^{n} p_{i\ell} X_{ij} \ \forall j \in \{2, \dots, n\} \quad \forall \ell \in \{i, \dots, m\}, \tag{2e}$$

$$C_{j,\ell+1}^{pos} \geq C_{j\ell}^{pos} + \sum_{i=1}^{n} p_{i,\ell+1} X_{ij} \ \forall j \in \{2, \dots, n\} \quad \forall \ell \in \{1, \dots, m-1\}, \tag{2f}$$

$$C_{j\ell}^{pos} \geq 0 \ \forall j \in \{2, \dots, n\} \quad \forall \ell \in \{i, \dots, m\}, \tag{2g}$$

$$X_{ij} \in \{0, 1\} \ \forall i, j \in \{1, \dots, n\}. \tag{2h}$$

Objective function (2a) tries to maximize sum of the payoffs given by Eq. (1). Constraints (2b) and (2c) ensure that each job is attributed to one position and vice-versa. Constraint (2d) ensures that on the first machine job J_j cannot start before its release date. Constraint (2e) forbids the overlapping of a job and its predecessor on a machine. Constraint (2f) establishes that a job cannot be processed on a machine before its completion on the preceding machine. Constraints (2g) and (2h) give the definition domain of the variables.

$F|r_j, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$ is strongly NP-hard, since the single-machine problem $1|r_j|\sum_{j=1}^{n} \mathcal{F}(C_j)$, defined by Seddik et al. (2013), is strongly NP-hard. The special case with two machines, no release dates and a common delivery date $F2|D_i = D, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$, equivalent to problem $F2|perm|\Sigma C_j$, is already NP-hard, as shown by Della Croce et al. (2011).

1.2. Literature review

The flowshop scheduling is one of the most studied problem in the field of operations research. Since the seminal paper of Johnson (1954), many variants or this problem have been proposed in the literature where different characteristics regarding optimization criterion, job properties and machine environment are considered (Pinedo, 2016). Among these variants, the permutation flowshop scheduling problem (PFSP) with makespan minimization criterion has received special attention of the researchers and the methods developed for this classical PSFP are usually extended for other related problems. For more detailed discussions and recent advances on PFSP, interested readers are referred to Framinan, Gupta, and Leisten (2004), Ruiz and Maroto (2005), Fernandez-Viagas, Ruiz, and Framinan (2017). For an overview of the field, the reader is referred to Gupta and Stafford (2006) which reviews fifty years of scheduling problems since Johnson's paper (Johnson, 1954). Other related problems are flowshop problems with regular sum objective functions $\sum_{i=1}^{n} f_i(C_i)$, but without release dates. Della Croce et al. (2011) propose a matheuristic for solving the two machine flowshop problem $F2||\Sigma C_i$; Della Croce, Gupta, and Tadei (2000) solve $F2|d_i = d|\sum U_i$ with a Branch and

Bound method. Total flowtime criterion was tackled Liu and Reeves (2001), which developed a constructive heuristic. Besides, Dong, Huang, and Chen (2009) solved such problem applying an Iterated Local Search heuristic. Comparative reviews on methods for PFSP with sum-form objective functions were developed by Vallada, Ruiz, and Minella (2008), which compare several heuristics for solving $F||\Sigma T_i$ (total tardiness minimization). Additionally, Pan and Ruiz (2013) and Fernandez-Viagas and Framinan (2015) consider the total flowtime minimization criterion for diverse heuristics evaluation.

To the best of our knowledge, none flowshop problem with stepwise job cost functions has been addressed in the literature. Indeed, such objective function was only studied on single machine (Detienne, Dauzère-Pérès, & Yugma, 2011; Seddik et al., 2013; 2015) and parallel machines scheduling problems (Curry & Peters, 2005; Detienne et al., 2011). However, the algorithms for single and parallel machines problem cannot be easily adapted to flowshop because they often rely on structural properties that do not apply any more.

The most closely related problems are flowshops with release dates and regular sum objective functions $\sum_{i=1}^{n} f_i(C_i)$, for which there are only few papers in the literature. Ladhari and Rakrouki (2009) consider $F2|r_i|\Sigma C_i$, for which they propose some Branch and Bound methods, solving instances with up to 100 jobs for some classes of instances and 30 jobs for the hardest class of instances. Besides, they consider the problem $F2|r_i, perm|\Sigma C_i$, for which they develop lower bounds, heuristics and a genetic algorithm. This work was recently extended by Rakrouki, Kooli, Chalghoumi, and Ladhari (2017) when they proposed some advances on the original Branch and Bound of Ladhari and Rakrouki (2009). Additionally, Chalghoumi, Mrad, and Ladhari (2013) presented some new lower bounds when dealing with the total completion time criterion.

1.3. Main contributions

The main contributions of this work are:

- The introduction of a new flowshop problem with an objective function based on stepwise jobs payoffs, and the development of benchmark instances to assess the performance of algorithms.
- The proposition of a new greedy constructive heuristic whose complexity is smaller than the classical NEH heuristic (Nawaz, Enscore, & Ham, 1983) for flowshop scheduling problems. Additionally, experimental evaluation shows that its results are competitive to those found by NEH method.
- The proposition of three metaheuristics for the flowshop scheduling problem with delivery dates and cumulative payoffs, both able to obtain better results than a commercial mixed integer programming solver.
- Finally, the establishment of primal and dual bounds for the flowshop scheduling problem with delivery dates and cumulative payoffs. These value bounds can be used as reference and guide future researches.

The structure of the paper is as follows. Section 2 describes constructive heuristics, while local search methods are presented in Section 3. A hybrid method based on the Iterated Local Search metaheuristic is customized in Section 4. Experimental settings are detailed in Section 7. The proposed methods are evaluated by extensive computational experiments reported in Section 8. The last section makes concluding remarks.

2. Constructive heuristics

Several heuristics and exact algorithms are able to perform very well if they start from a good initial solution. Due to this, we evaluate four constructive heuristics for the flowshop scheduling problem with delivery dates and cumulative payoffs. In this section, we present three methods found in the literature, and propose a new greedy constructive heuristic.

R (Release dates): this heuristic produces a left-shifted schedule where jobs are ordered w.r.t. nondecreasing release dates. Potts (1985) applied this $O(n\log n)$ method for the two-machine permutation flow-shop problem that minimizes the makespam.

ECT (Earliest Completion Time): this heuristic produces a left-shifted schedule where jobs are sorted in the increasing order of their earliest possible completion time $E_i = r_i + \sum_{\ell=1}^{m} p_{i\ell}$. This $O(n\log n)$ algorithm was proposed by Ladhari and Rakrouki (2009) for the minimization of the total completion time on a two-machine flowshop problem with release dates.

NEH: the steps of an algorithm based on the well-known Nawaz et al. (1983) heuristic are the following:

1. Sort the jobs in the increasing order of $E_i = r_i + \sum_{\ell=1}^{m} p_{i\ell}$.
2. Consider the two jobs J_{i_1}, J_{i_2} with the smallest values of E_i. Among the two left-shifted partial schedules corresponding to sequences (J_{i_1}, J_{i_2}) and (J_{i_2}, J_{i_1}), choose the one with the greatest payoff as the current schedule.
3. Repeat until all jobs are scheduled: Let J_e be the unscheduled job ith the smallest value of E_i. Update the current schedule, by inserting J_e, in the position that maximizes the payoff, while keeping a left-shifted partial schedule.

Step 3 concentrates the largest computational effort of the algorithm. For each tentative position, the method evaluates the solution cost in $O(nm)$ steps. As each job can be inserted in $O(n)$ positions, the complexity of NEH heuristic for the problem under consideration is $O(n^3m)$. The NEH heuristic, originally proposed by Nawaz et al. (1983) for a flowshop problem without release dates, was modified by Ladhari and Rakrouki (2009) by considering the sorting criterion described above. The method described in Ladhari and Rakrouki (2009) is easily adapted for the addressed problem by simply changing the payoff evaluation.

FF: this heuristic, proposed by Fernandez-Viagas and Framinan (2015), is an improved version of the constructive method developed by Liu and Reeves (2001). Starting from scratch, the solution is built by iteratively appending a job at the end of a partial solution. At each iteration, each unscheduled job is evaluated according to an index function ξ' that considers the idle time and the completion time originated if such a job is placed in the last position of the scheduling. The job with the smallest value of ξ' is chosen to be added in the solution. These steps define a time complexity of $O(n^2m)$ for the method. The index function, as defined by Fernandez-Viagas and Framinan (2015), is the following:

$$\xi' = \frac{(n-k-2)}{a} IT'_{j,k} + AT'_{j,k} \tag{3}$$

with

$$IT'_{j,k} = \sum_{i=2}^{m} \frac{m \cdot \max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i - b + k(m - i + b)/(n-2)} \tag{4}$$

$$AT'_{j,k} = C_{m,j} \tag{5}$$

where, k is the total of jobs in the partial solution, so that $IT'_{j,k}$ is the weighted idle time between an unscheduled job j and the job placed at the kth position (or the last job in the partial solution) and $AT'_{j,k}$ is the makespam of the solution, if job j is appended. Parameters a and b are applied to equilibrate the weights of both idle time and makespan in the index function.

In this work, we apply a small modification in Eq. (4) in order to consider the job release dates in the first machine, since this characteristic can contribute to the idle time. So, we included $i = 1$ in the summation and, in this case, the expression $\max\{C_{i-1,j} - C_{i,[k]}, 0\}$ is replaced by $\max\{r_j - C_{i,[k]}, 0\}$. For $i = 2 \ldots m$ the original expression $\max\{C_{i-1,j} - C_{i,[k]}, 0\}$ is kept.

IECT (Iterated Earliest Completion Time): Following the ideas of NEH and ECT methods, we developed a new iterative constructive method. A job sequence is built from scratch and the first added job is the one with the smallest value of E_j, as defined for the ECT method. After that, at each iteration, the earliest possible completion time of all jobs that are not in the partial schedule are reevaluated by considering the completion time, on each machine, of the last job added in the schedule.

Algorithm 1 describes the IECT method.

Algorithm 1: Constructive heuristic IECT.

```
1  $L \leftarrow \{1, \ldots, n\}$;
2  foreach $j \in L$ do
3      $E_j \leftarrow r_j + \sum_{\ell=1}^{m} p_{j\ell}$;
4  $j^* \leftarrow \mathrm{argmin}\{E_j : j \in L\}$;
5  $S \leftarrow J_{j^*}$;
6  $L \leftarrow L \setminus \{j^*\}$;
7  while $L \neq \emptyset$ do
8      foreach $j \in L$ do
9          $E_{j1} \leftarrow \max\{r_j, C_{j^*,1}\} + p_{j1}$;
10         for $\ell \leftarrow 2, \ldots, m$ do
11             $E_{j\ell} \leftarrow \max\{C_{j^*,\ell}, E_{j,\ell-1}\} + p_{j\ell}$;
12     $j^* \leftarrow \mathrm{argmin}\{E_{jm} : j \in L\}$;
13     $S \leftarrow S \| J_{j^*}$;
14     $L \leftarrow L \setminus \{j^*\}$;
15 return $S$;
```

The list L of jobs indices is initialized at line 1. The earliest possible completion time is computed at lines 2 and 3 for all jobs. The index j^* of the job with the earliest possible completion time is identified at line 4. Then, solution S is initialized with J_{j^*} as the first job in the sequence (line 5), and index j^* remove from the job list (line 6). The main loop at lines 7 to 14 adds one job at a time to the sequence, until all jobs are scheduled. For that, the loop at lines 8 to 11 calculates the earliest possible completion time of all jobs indexed in L. Line 9 considers the first machine and calculates the earliest possible completion time E_{j1} of job J_j by adding its processing time to the maximum value between its release date and the completion time on machine 1 of the last job in the schedule. Remaining machines are treated by the loop at lines 10 and 11. Line 11 computes the earliest possible completion time of job J_j on machine M_ℓ by adding its processing time to the maximum value between its predecessor on the machine under consideration and its own completion time on the preceding machine. At lines 12 to 14, respectively, the job with the smallest possible completion time in the last machine is identified, concatenated to the partial schedule, and its index is removed from L. Note that $S \| J_{j^*}$ denotes the concatenation of job J_{j^*} at the end of S. Finally, solution S is returned on line 15. IECT has time complexity of $O(n^2 m)$.

3. Local search methods

Local search methods attempt to find cost-improving solutions by exploring the neighborhood of an initial solution. If none is found, then the search returns the initial solution as a local minimum. Otherwise, if an improving solution is found, it is made the new initial solution, and the procedure repeats itself (Talbi, 2009). We use here the *best improvement strategy*: the new initial solution is the best one in the neighborhood of the previous solution.

3.1. Neighborhoods

The two neighborhoods explored in this work are based on those standard ones studied by den Besten and Stützle (2001) for scheduling problems. Recall that any feasible left-shifted schedule of $F|r_j, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$ can be represented as a sequence.

- In the *interchange* neighborhood, a neighbor sequence is obtained by swapping two jobs from their original positions. The neighborhood size is thus $n(n-1)/2$.
- The *insert* neighborhood attempts to move a job to a new position that improves the solution cost. For each job, $n-1$ positions can be verified, therefore the whole neighborhood contains $O(n^2)$ solutions.

For both methods, the cost of each solution is verified in $\Theta(nm)$ steps which make the local search an $O(n^3 m)$ method.

3.2. Variable neighborhood descent

Attempting to improve the results obtained by the previous methods, we consider the use of metaheuristics. These are high-level procedures that coordinate simple heuristics, such as local search, to find better quality solutions than those obtained by the individual application of a heuristic. Glover and Kochenberger (2003) present the classical metaheuristics found in the literature and their applications in diverse domains.

The metaheuristic Variable Neighborhood Descent (VND) establishes an order in which the local search neighborhoods can be combined in order to better explore the solution space (Hansen & Mladenović, 2003). Starting from a solution generated by the constructive algorithm, a VND exploits the first neighborhood until a local optimum is found. This solution becomes the initial solution for the next neighborhood, and the sequence continues until the last neighborhood is reached. If, during the investigation in the local search algorithm, an improving solution is found, then the procedure returns to the first neighborhood. For details about VND, refer to Hansen and Mladenović (2003).

4. Hybrid Iterated Local Search

A common technique to solve complex problems is to use hybrid methods than combines different kinds of algorithms, making up for shortcomings of one method with special characteristics of the other. The *hybrid metaheuristic* concept refers to the combination of one or more algorithmic ideas from different metaheuristics and sometimes even from outside the traditional field of metaheuristics. Raidl (2006) surveys the meaningful hybridization approaches presenting a unified view of hybrid metaheuristics. Moreover, Raidl and Puchinger (2008) discuss promising combinations of exact algorithms and metaheuristics to solve combinatorial optimization problems.

Iterated Local Search (ILS) is a metaheuristic which builds a sequence of solutions by iteratively applying perturbations and local search methods from an initial solution. Lourenço, Martin, and Stützle (2003) describe the main parts of the ILS and review some applications of this method. Applications of ILS for flowshop problems can be found in den Besten and Stützle (2001), Lourenço et al. (2003), Dong et al. (2009), and Stützle (1998).

A basic ILS algorithm is built with four components: (1) a method to generate an initial solution; (2) a perturbation method, to allow the ILS to escape local optima; (3) a local search procedure; and (4) an acceptance criterion, to decide if a solution will replace the current one to the next iteration. The method to generate an initial solution can be either a randomized method or a greedy constructive heuristic. Both can be combined with a local search. Once an initial solution is created, the method execute steps (2)–(4) in a loop until a stopping condition is reached. Perturbations are applied to the current local minimum and modify a number of solution elements. The higher the number of modified elements, the greater the strength of the perturbation, which must be carefully studied in order to avoid random restarts or falling back into previous solutions just visited. The perturbation method together with the local search brings a new solution that replaces the current one to the next loop iteration if an acceptance criterion is met. This component may range between a strong intensification criterion, by only accepting improving solutions, and a strong diversification, by accepting any new solution, not taken its cost into consideration.

Algorithm 2 describes an ILS for the flowshop scheduling problem with delivery dates and cumulative payoffs. The initial solution is generated by a constructive heuristic (line 1) followed by the application of a VND described in lines 3–8. The VND starts checking if the current solution S is better than the best solution S^*. If it is the case, we set the current solution as the best one (lines 4 and 5). Note that function val(S) returns the payoff of solution S. Following, the algorithm performs local searches in the insert and interchange neighborhoods (lines 6 and 7), as described in Section 3.1. If no improvement in S is found, the VND stops, other another iteration is performed. In the main ILS loop (lines 9–19), the perturbation step (lines 10–12) consists on swap neighbor jobs (line 11) and two random chosen jobs (line 12 ×) sequentially for a given ρ number of iterations. For the local search step, we choose between either simple local searches in the *Insert* and *Interchange* neighborhoods, or a VND algorithm similarly to lines 3–8. However, the chosen order for the neighborhoods may be different. Section 8.4 evaluates the effects of switch the order of these neighborhoods. If solution S has a better payoff than solution S^*, we accept S as the best current solution (line 19). The stop criteria can be maximum running time, maximum number of iterations, or yet a target value for the solution.

Algorithm 2: Iterated local search for $F|r_j, perm|\sum_{j=1}^{n}\mathcal{F}(C_j)$.

```
1  Let $S$ be an initial solution obtained by a constructive
   heuristic;
2  $S^* \leftarrow S$;
3  repeat
4      if val($S$) > val($S^*$) then
5          $S^* \leftarrow S$;
6      Perform a local search in the Insert neighborhood in $S$;
7      Perform a local search in the Interchange neighborhood in
       $S$;
8  until val($S$) = val($S^*$);
9  while a stopping criterion is not reached do
10     for $i \leftarrow 1$ to $\rho$ do
11         Swap two neighbor jobs $(J_i, J_{i+1})$ in $S$, where $i$ is
           uniformly chosen from $[1, n-1]$;
12         Swap a pair of jobs $(J_i, J_j)$ in $S$, at the $i$th and $j$th
           uniformly chosen random positions;
13     Considering $S$ as initial solution, apply one of the
       following:
       (1) local search Insert neighborhoods;
14     (2) local search Interchange neighborhood;
15     (3) a VND using a combination of the Insert and
       Interchange neighborhoods;
16     (4) a VND using a combination of the Interchange and
       Insert neighborhoods;
17     if val($S$) > val($S^*$) then
18         $S^* \leftarrow S$;
19 return $S^*$;
```

5. Hybrid Iterated Greedy Search

Iterated Greedy Search (IGS) explores the solution space from a greedy or randomized initial solution. Then, the algorithm iteratively proceeds partial destruction followed by a greedy reconstructing of solutions, which are finally submitted to local search methods. A locally optimum solution replaces the preceding one according to a Simulated Annealing-like acceptance criterion (Nikolaev & Jacobson, 2010).

The first applicaton of the Iterated Greedy Search (IGS) heuristic in the context of scheduling problems was proposed by Ruiz and Stützle (2007). Despite its similarities to the Iterated Local Search, IGS has been proved to be more efficient when solving flowshop problems. Recently, Fernandez-Viagas et al. (2017) showed the best performance of IGS among diverse heuristics and metaheuristic methods for the permutation flowshop with makespam minimization criterion. Other scheduling problems successfully tackled by IGS are, for example, the blocking flowshop scheduling problem with makespan criterion (Tasgetiren, Kizilay, Pan, & Suganthan, 2017), the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives (Ruiz & Stützle, 2008), and the unrelated parallel machine scheduling (Fanjul-Peyro & Ruiz, 2010).

An IGS for the flowshop scheduling problem with delivery dates and cumulative payoffs is described by Algorithm 3. From lines 1 to 8, IGS proceeds the same operations then ILS to build an initial locally optimum solution. The main loop of the IGS heuristic controls the destruction and construction procedures, as well as the acceptance of new solutions. Firstly, we set S' with the current locally optimal solution S, in line 10, for a future comparison. In the *destruction* loop (lines 11–13), d jobs are randomly chosen to be removed from S. Each of these jobs is included in a list L. Line 14 proceeds the *construction* step by applying the IECT heuristic (1) taking into account the reduced solution S as its starting point (instead of building a solution from scratch). Jobs from L are considered according to their earliest possible completion time, as described in Section 2, for reinsertion in S (line 14). A local search method applied on S according to the choice made in lines 16–19, creates the locally optimum solution S''. Lines 20–23 update the initial solution for the next main loop iteration, by considering the Simulated Annealing-like acceptance criterion based on the parameter T (temperature). If applicable, line 25 updates S^* the best solution that has found so far. As in the ILS algorithm, a stop criterion is verified to determine the end of the procedure execution.

6. Biased Random-Key Genetic Algorithm

Genetic algorithms have been largely applied for flowshop scheduling problems (Chen, Vempati, & Aljaber, 1995; Onar, Öztayşi, Kahraman, Yanık, Şenvar, 2016; Reeves, 1995; Ruiz, Maroto, & Alcaraz, 2006; Zhang, Li, & Wang, 2009). The Biased Random-Key Genetic Algorithm (BRKGA) is an elitist method that has been

Algorithm 3: Iterated Greedy Search for $F|r_j, perm|\sum_{j=1}^{n}\mathcal{F}(C_j)$.

```
1  Let $S$ be an initial solution obtained by constructive heuristic;
2  $S^* \leftarrow S$;
3  repeat
4      if $\text{val}(S) > \text{val}(S^*)$ then
5          $S^* \leftarrow S$;
6      Perform a local search in the Insert neighborhood in $S$;
7      Perform a local search in the Interchange neighborhood in $S$;
8  until $\text{val}(S) = \text{val}(S^*)$;
9  while a stopping criterion is not reached do
10     $S' \leftarrow S$;
11     for $i \leftarrow 1$ to $d$ do
12         Remove a job $J_r$ from $S$, where $r$ is uniformly chosen,
           without repetition, from $[1, n]$;
13         Include $J_r$ in a list $L$ of removed jobs;
14     Apply a constructive heuristic to complete $S$ with jobs of $L$;
15     Considering $S$ as an initial solution, apply one of the
       following methods to obtain $S''$ :
       (1) local search Insert neighborhoods;
16     (2) local search Interchange neighborhood;
17     (3) a VND using a combination of the Insert and
       Interchange neighborhoods;
18     (4) a VND using a combination of the Interchange and
       Insert neighborhoods;
19     if $\text{val}(S'') > \text{val}(S')$ then
20         $S \leftarrow S''$;
21     else if $random \le \exp((val(S'') - val(S'))/T)$ then
22         $S \leftarrow S''$;
23     if $\text{val}(S'') > \text{val}(S^*)$ then
24         $S^* \leftarrow S''$;
25 return $S^*$;
```

shown to be useful to a wide variety of combinatorial problems (Andrade, Miyazawa, & Resende, 2013; Andrade, Resende, Karloff, & Miyazawa, 2014; Andrade et al., 2015a; Andrade, Toso, Resende, & Miyazawa, 2015b; Gonçalves & Resende, 2011b; Stefanello, Aggarwal, Buriol, Gonçalves, & Resende, 2015), machine learning (Caserta & Reiners, 2016), and in hybridizations with other mixed integer programming techniques for find feasible solutions for problems in such task is difficult (Andrade, Ahmed, Nemhauser, & Shao, 2017). BRKGA is able to overcome other classic genetic algorithms such as the RKGA as shown by (Gonçalves, Resende, & Toso, 2014). The BRKGA was proposed by Gonçalves and de Almeida (2002) and Ericsson, Resende, and Pardalos (2002), and works with a pool of solutions $\mathcal{P}$ called *population*. The population is partitioned into two groups according to a performance metric, usually referred as *fitness*. The *elite set* $\mathcal{P}_e$ compounds the solutions that are the best of the whole population. The *non-elite set* is formed by the remaining individuals $(\mathcal{P} \setminus \mathcal{P}_e)$. On each evolutionary iteration, the elite set is copied to the next generation population $\mathcal{P}^{\mathcal{N}}$. A small percentage of random solutions, called *mutants*, is added to $\mathcal{P}^{\mathcal{N}}$. For the remaining positions, the algorithm creates offsprings using the biased uniform crossover between a randomly chosen individual from the elite set and a randomly chosen individual from the non-elite set, i.e., with probability ρ, a component is taken from the elite individual, otherwise the component is take from the non-elite individual. In the BRKGA, each individual is represented by a vector $v \in [0, 1]^n$, for a given n. A solution is obtained applying a function $f : [0, 1]^n \to \mathcal{S}$ that maps a vector v to a solution in the space $\mathcal{S}$ which is the space of solutions of the problem to be solved. Function f is called *decoder* and usually also computes the fitness of the individual. For a detailed description, see Gonçalves and Resende (2011a).

Algorithm 4 describes a decoder for the Flowshop Scheduling Problem with Delivery Dates and Cumulative Payoffs.

Algorithm 4: Decoder.

```
   Input: Vector $v \in [0, 1]^n$, max. num. iterations $\tau$.
   Output: Feasible schedule and cumulative payoff.
1  Let $\pi$ to be a permutation of jobs induced by the
   non-increasing order of correspondent keys in $v$;
2  Build a schedule $S$ such the jobs are ordered as in $\pi$;
3  $best\_payoff \leftarrow 0$;
4  $iteration \leftarrow 0$;
5  while $\text{val}(S) > best\_payoff$ and $iteration < \tau$ and the
   maximum time is not reached do
6      $best\_payoff \leftarrow \text{val}(S)$;
7      Perform local search on Insert neighborhood;
8      Perform local search on Interchange neighborhood;
9      $++iteration$;
10 Rewrite the chromosome keys according to $S$;
11 return $S, \text{val}(S)$
```

We assume that each job $J_i \in \mathcal{J}$ is associated with an allele/key v_i for the chromosome $v \in [0, 1]^n$ where $n = |\mathcal{J}|$. In line 1, the jobs are sorted according the value of their respective keys. Using this order, the algorithm builds a schedule on line 2. The loop between lines 5 and 9, applies a local search in the *Insert* neighborhood (line 7), then in the *Interchange* neighborhood (line 9). The loop stops either the payoff does not improve, or it reaches the maximum number of local search iterations τ, or yet the maximum time is reached. Note that when $\tau = 0$, no local search is performed, and when $\tau = \infty$, a full local search is carried out (if enough time is allowed to do so). The schedule S may be changed by the local searches and, therefore, we have to rewrite the chromosome to reflect the correct order of the jobs (line 10). This operation guarantees that the chromosome reflects the correct solution during the mating process. Note that the payoff is used as the performance measurement (fitness) during the evolutionary process. Note that during the local search, we can apply the local search first on the *Interchange* neighborhood and then in the *Insert* neighborhood. However, such order does not interfere in the quality of the solutions, as shown in Section 8.2.

To improve the convergence rate, we start the BRKGA population with a solution from one of the constructive methods. Such solutions allow BRKGA jump-start with good job sub-sequences that usually take part in the best solutions.

In our implementation, we used the island model (Whitley, Rana, & Heckendorn, 1998), commonly find in other genetic algorithm variations. In the island model, several populations are evolved independently and exchange their best individuals every given number of generations. This strategy improves the variability of individuals, usually speeding up convergence, and reduces the risk that the algorithm will get stuck in local optima. However, if such situation happens for a given number of iterations, we reset the population keeping the best solution found so far. Section 7.4 describes the parameter tuning for BRKGA with and without local search in the decoder.

7. Experimental settings

7.1. Instances generation

We randomly generated two groups of 150 test instances for the problem addressed in this work. The first group contains instances with 100 jobs which must be processed on 2 machines. In the second group, the instances have 200 jobs with the same conditions as before. Processing times are random integers from a uniform distribution in the interval [1, 100]. The number of delivery dates varies in {1, 2, 3, 5, 7, 10}. Delivery dates for each instance are defined as follows. Let C be the makespan of a schedule obtained by using Johnson's rule (Brucker, 2004), while ignoring release dates. Let $D_1 = \lfloor(\alpha \times C)/K\rfloor$ be the first delivery date, where the parameter $\alpha \in \{0.1, 0.3, 0.5, 0.7, 1.0\}$. The other delivery dates are computed as: $D_k = k \times D_1$ for $1 < k \leq K$. Following Seddik et al. (2013), the release date of each job is picked randomly in one of the intervals $r_k = [D_{k-1}, D_{k-1} + R \times D_{k-1})$, with $D_0 = 0$, $k = 1, \ldots, K$, and R is a parameter taking values in {0.1, 0.3, 0.5, 0.7, 1.0}. The choice of the interval is also random (each with probability $1/K$).

The instance naming follows the pattern `k<K>n<N>a<α>r<R>`. For example, instance `k2n100a0.1r0.7` has $N = 100$ jobs; $K = 2$ delivery dates; parameter α, used to give the first delivery date, is $\alpha = 0.1$; and $R = 0.7$ is the same parameter R used to define the release date of each job. The instances can be found on https://doi.org/10.17632/xyrw89wcjt.1.

7.2. Initial bounds

In order to have some reference values to evaluate the proposed heuristics, the MILP formulation was solved with a commercial solver. In this way, we obtained lower bounds, as well as upper bounds provided by a linear relaxation.

An alternative upper bound on the $F2|r_j, perm|\sum_{j=1}^{n} \mathcal{F}(C_j)$ is obtained by considering the relaxed problem where the first machine has infinite capacity (i.e., all jobs can be performed in parallel). This relaxed problem is equivalent to the single machine problem where the release date of job J_i is equal to $r_i + p_{i1}$ and its processing time is equal to p_{i2}, for all $i \in \{1, \ldots, n\}$. By adapting the algorithm presented in Seddik et al. (2013), we can efficiently solve this relaxed version of the problem in $O(Kn\log n)$ time. We refer to the upper bound obtained from this relaxation as "Uncapacitated Upper Bound."

7.3. Computational environment

The computational experiments were performed in a cluster of identical machines with a Intel Xeon E5530 CPU at 2.40GHz and 32 GB of RAM running CentOS Linux release 7.2.1511. Each run was limited to a single processor. Heuristics were implemented in C++ and compiled with GNU GCC 5.2.0. The ILP formulation was solved with IBM ILOG CPLEX 12.6.2 solver and the function STEP embedded in the OPL language was used to implement the objective function. CPLEX stopped when an optimal integer solution was found, or when the CPU time limit of three hours is reached. For ILS, IGS, and BRKGA, the stopping criterion is also three hours.

7.4. Parameter settings

Since each strategy has control parameters as input, we tuned them as we shortly describe following. For details, refer to Appendix A in the supplementary materials.

Among the heuristics, only FF requires parameter setting. We performed a full factorial design of experiments using values suggested by Fernandez-Viagas and Framinan (2015). The Wilcoxon U test reveals that we cannot affirm significant differences among the parameter levels, at 95% confidence level. Therefore, we picked $a = 4$ and $b = 0$, since these levels resulted in the best average and best median deviation.

For ILS and IGS, we performed a full factorial design of experiments, since the number of control parameters for both algorithms is small. For ILS, these experiments reviewed that a small number of swap moves led to a better performance than applying a strong perturbation. Therefore, we chose to apply 8 swap moves for the perturbation step. For IGS, the experiments shown that 2 removals (followed by the respective insertions) for the perturbation step result in the best results. However, the results do not differ significantly in quality or running time for different levels of the temperature factor. Therefore, we chose to use 0.2.

For BRKGA, due to the large number of control parameters, we performed the parameter tuning using the `irace` package (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016). Note that the results of the decoding can vary substantially within the choice of the maximum number of iterations τ. Since we are interested in compare the best results independently value of τ, we performed two separately tunings. First, we tuned for BRKGA without local search (BRKGA-NLS) where τ is fixed to zero. `irace` suggested: population size of $p = 5000$; elite size $p_e = 0.34 \times p = 1700$; number of mutants introduced at each generation $p_m = 0.11 \times p = 550$; probability of inheriting each allele from elite parent $\rho = 0.74$; one population; and reset interval of 300 iterations without improvements in the best solution. Allowing `irace` to tune τ (BRKGA-LS), the package suggested: population size of $p = 150$; elite size $p_e = \lfloor 0.11 \times p\rfloor = 16$; number of mutants introduced at each generation $p_m = \lfloor 0.11 \times p\rfloor = 16$; probability of inheriting each allele from elite parent $\rho = 0.71$; maximum number of local search iterations $\tau = 1$; two populations which exchange one individual on each 50 iterations; and reset interval of 350 iterations without improvements in the best solution. Note that, since BRKGA-NLS has not local search, it needs more variability in the population compared to the BRKGA-LS, as suggested by the size of its population, and smaller inheritance probability.

8. Computational experiments

8.1. Instance analysis

To assess the difficult in solving the instances, we try to solve each instance using CPLEX for three hours. Since CPLEX is a very mature solver that includes sophisticated several heuristics and exact methods, the hardness results are probably similar in other frameworks.

In three hours, CPLEX is able to prove the optimality for 17 instances (5.67% of the instances). Among them, the unique 200-job instance for which an optimum solution was found is `k1n200a1.0r0.1`. For the other instances, the average integrality gap is 174.10 ± 339.58, with minimum 0.0025 and maximum 2187.36. For 173 instances (57.67%), the gap is larger than 100%. For 25 instances (8.33%) no solution was found in 3 hours (such solutions have 200 jobs). In other hand, for 18 instances (6%), the gap is less than 1%.

Fig. 2 depicts the distribution of the integrality gaps considering each generation parameter individually. In Fig. 2(a), the gaps became large as K does. This fact is expected since more delivery dates usually means more decision points to be made. In Fig. 2(b), the gaps decrease in function of the size of α. Note that for small α's, the delivery times are close to the release times leading to tidy regions for optimal solutions. In other hand, the gaps for large α's do no differ significantly. The R levels appears do not affect the hardness of the instances as suggest the overlapping notches in Fig. 2(c).

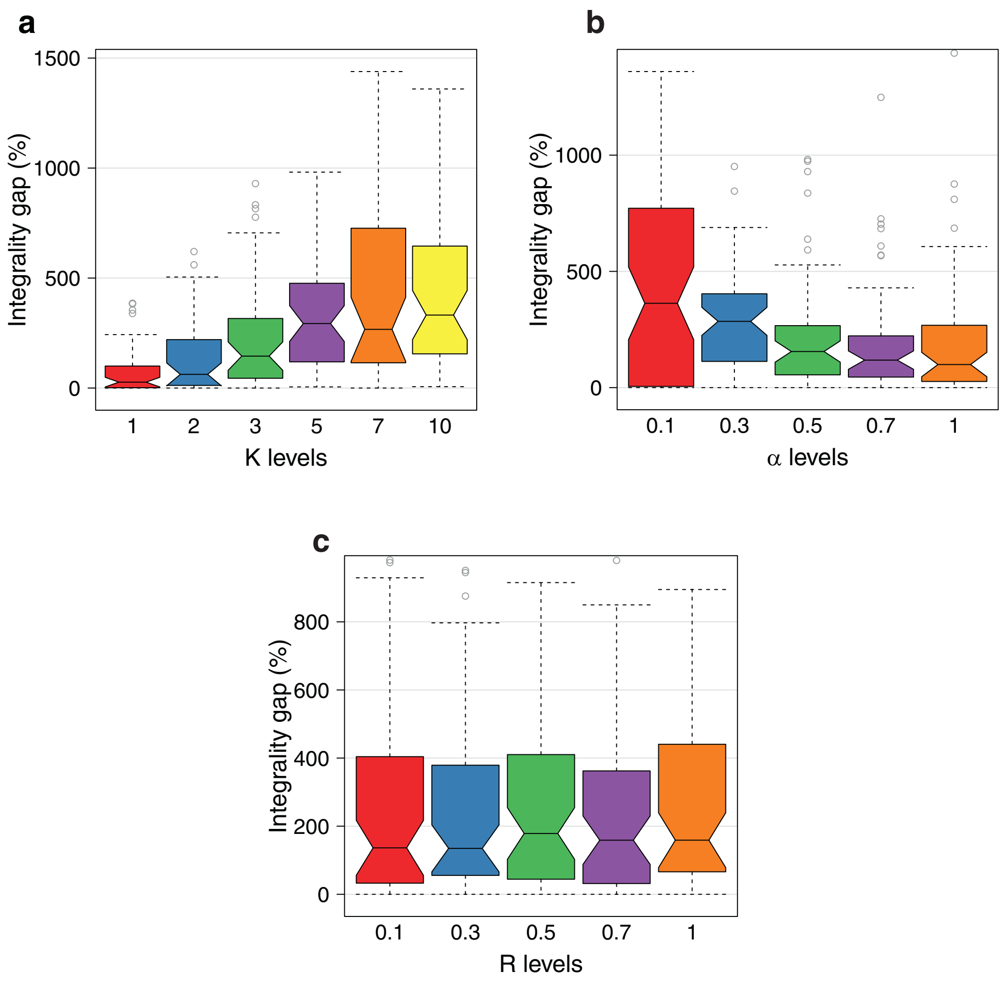


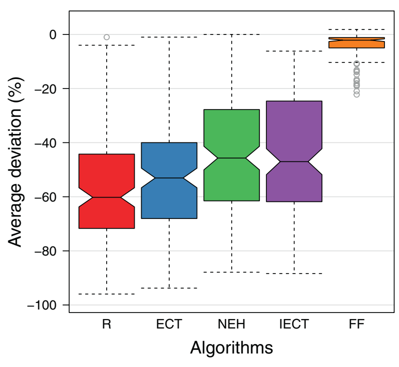
Fig. 2. Integrality gap for CPLEX limited to three hours runs. Gaps larger than 400% were omitted.

Since an optimum solution was found only for few instances and, for the remaining, the average integrality gap is relatively large, we rerun CPLEX for 24 hours for the 30 instances with 100 jobs that presented the largest gap. The objective is to assess the difficult to reduce the gap and prove optimality. Table 1 shows the comparison between gaps for 3-hour and 24-hour experiments. The first column describes the instance. The second and third columns bring the 3-hour and 24-hour gaps, respectively. The fourth column shows the absolute difference between the gaps, and the last column the proportional improvement in the gap for 24-hour experiments. The lines are sorted from the largest to the smaller 24-hour gap. The last line shows the averages of the columns. In general, CPLEX is able to improve the gap in 43.62% when allowed to run for 24 hours. For the last eight instances in the table, the improvement is notable, although the optimality could not be proved, even for instance `k5n100a0.1r0.7`. In other hand, for 19 of 30 instances, the gaps could not be improved below 100%. Particularly, instance `k7n100a0.1r0.1` show itself very hard since the integrality gap is very large even for 24-hour run. In face of these results, we can argue that this problem is very hard to solve, at least, from the perspective of exact algorithms.

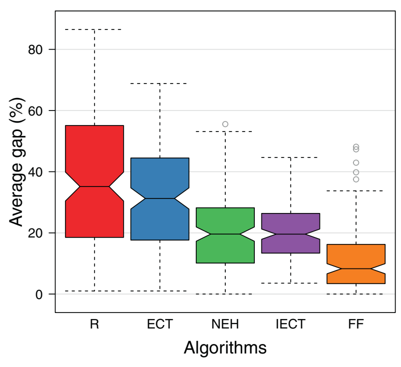
8.2. Constructive and local search methods

First, we evaluate the constructive methods described in Section 2 since they are used as subroutines in the other more sophisticated algorithms. For each instance, we compute the relative percentage deviation between the solution found with the considered method and the CPLEX solution value. Since CPLEX was not able to find feasible solution for several 200-job instances, we only used 100-job instances in these experiments. For a given instance, if val_{cplex} is the value of the CPLEX solution and val_{alg} is the value of the solution given by an algorithm for the same instance, then $(val_{alg} - val_{cplex})/val_{cplex}$ is the percentage of the deviation for the solution of the algorithm compared to CPLEX solution. We report the average percentage deviation ($AvgDev(\%)$) across all the instances. Note that negative AvgDev(%) values mean that, on average, the method finds worse solutions in comparison with those obtained by CPLEX. Besides, we identify the best values of upper bound obtained over the CPLEX linear relaxation ($CPLEX_{LR}$) and the relaxation described in Section 7.2 ($UncapUB$). Let $BestUB = \min(CPLEX_{LR}, UncapUB)$. We compute the integrality gap $(val_{alg} - BestUB)/BestUB$, whose average, over all the instances, is reported as $AvgGap(\%)$.

Fig. 3 depicts the box plots for the deviation percentage and the gap percentage. The results show that methods R and ECT, which builds a solution by just following some sorting criterion, yield the worst average relative deviation (−57.53% and −52.49%, respectively) and average gap (36.79% and 30.71%, respectively) varying widely. However, the average processing times for R and ECT are about one millisecond. The quality of the solutions obtained by NEH and IECT methods are similar. The average of the difference between the deviations is 5.25 ±9.46, minimum of 0.00 (no difference), and maximum of 54.55. However, the NEH complexity makes its average processing time to reach 0.34 second while IECT average processing time is less than one millisecond. FF obtained



(a) Deviation percentage.



(b) Gap percentage.

Fig. 3. Deviation and gap of solutions of the respective constructive heuristics when compared to CPLEX solutions.

Table 1
Integrality gap presented by CPLEX for three and 24-hour experiments. We only consider the 30 100-job instances whose gaps were the worst in three hours. The instances are order by the worst gap in 24-hour experiments.

Instance	3 hours	24 hours	Diff.	% Imp.
k7n100a0.1r0.1	616.96	608.33	8.53	1.38
k7n100a0.3r1.0	303.58	303.44	0.04	0.01
k10n100a0.3r0.7	362.37	301.66	60.61	16.73
k7n100a0.3r0.7	280.65	271.43	9.11	3.24
k10n100a0.3r0.3	330.68	264.92	65.65	19.85
k10n100a0.3r0.1	311.34	263.56	47.78	15.34
k10n100a0.3r1.0	250.95	241.38	9.47	3.77
k5n100a0.3r1.0	296.72	221.37	75.34	25.39
k5n100a0.3r0.7	325.99	214.96	110.93	34.03
k10n100a0.5r0.7	232.65	213.04	19.50	8.38
k10n100a0.5r1.0	232.43	198.47	33.96	14.61
k10n100a0.5r0.5	221.31	196.47	24.83	11.22
k5n100a0.3r0.1	253.44	192.95	60.48	23.86
k7n100a0.3r0.1	252.91	152.65	100.26	39.64
k10n100a0.7r0.7	168.55	146.34	22.11	13.12
k10n100a0.5r0.1	181.97	142.42	39.44	21.68
k7n100a0.3r0.3	192.30	131.26	61.03	31.74
k10n100a0.7r0.1	163.49	131.24	32.14	19.67
k7n100a0.5r0.7	161.33	130.41	30.92	80.83
k3n100a0.5r1.0	162.50	112.17	50.32	30.96
k5n100a0.5r0.7	183.23	75.82	107.41	58.61
k10n100a0.3r0.5	174.19	73.10	100.99	58.00
k5n100a0.3r0.5	289.51	22.22	267.29	92.32
k3n100a0.3r0.3	216.21	10.98	205.22	94.91
k2n100a0.3r1.0	181.05	7.81	173.13	95.68
k3n100a0.3r0.5	251.80	7.05	244.74	97.19
k10n100a0.1r0.5	369.57	5.78	363.69	98.43
k5n100a0.3r0.3	302.36	5.38	296.88	98.21
k5n100a0.1r0.5	355.92	1.24	354.68	99.65
k5n100a0.1r0.7	336.84	0.01	336.83	99.99
Average	265.40	154.93	–	43.62

Table 2
Difference in median location for cost distributions for all instances, for the constructive heuristics. The main diagonal shows the median of the results of each algorithm. We use a confidence interval of 95% and omit p-values are less than 0.05.

	R	ECT	NEH	IECT	FF
R	−60.21	−5.48	−13.01	−12.92	−55.77
ECT		−53.05	−7.73	−7.63	−48.75
NEH			−45.70	0.00	−40.95
IECT			1.00	−47.02	−42.87
FF					−2.13

the best results with deviation −4.18% and gap 4.21% which is ten times better than NEH and IECT. For all instances, FF took less than one millisecond to run. Worth to note that the maximum running time for R, ECT, and IECT is less than four milliseconds. NEH takes longer than the previous algorithms to run: in seconds, we have minimum of 0.28, maximum of 0.47, and average and standard deviation of 0.34 ± 0.04.

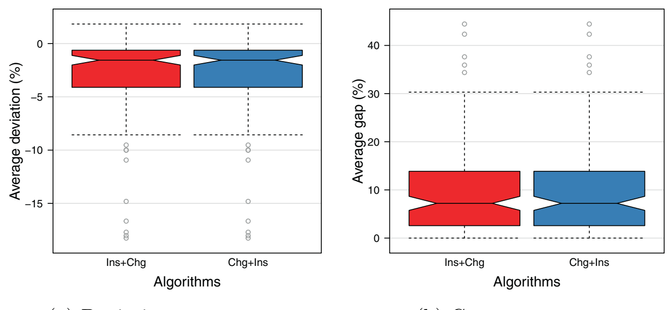
To confirm our conclusions, we tested the normality of these distributions using the Shapiro–Wilk test and applied the Mann–Whitney–Wilcoxon U test, considered more effective than the *t*-test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover, 1980; Fay & Proschan, 2010). For all tests, we assume a confidence interval of 95%. Except for IECT, the *p*-values for all algorithms' distributions are under 0.05 indicating that we can reject the normality hypotheses and, therefore, the U test should be applied. As several statistical tests were performed, we used a *p*-value correction procedure based on false discovery rate (FDR) to minimize the number of false positives (Type I error) (Benjamini & Hochberg, 1995).

Table 2 show U test results for each pair of algorithms at 95% of confidence level. The structure of these tables is as follows. Each row and column is indexed by one algorithm. Each element in the diagonal (bold) is the median of the percentage deviation for the corresponding algorithm. The upper-right diagonal elements are medians of the differences in location statistics for each pair of algorithms. Larger negative differences indicates that the "row algorithm" has its location statistics worse than the "column algorithm". The bottom-left diagonal elements are the *p*-values of each test. We omit all $p < 0.05$ values, that indicate that the difference is statistically significant for those pairs. We also omitted confidence intervals since for all tests the values lie in these intervals and they are very narrow. Note that since $p > 0.05$, we cannot affirm significant differences between NEH and IECT, as observed earlier. However, NEH and IECT are significantly better than R and ECT pairwise. Finally, FF presented the best values with clear margin from the other approaches.

Now, we combine Interchange and Insertion neighborhoods in a VND method, so that two VND variants are created by changing the order of the search in the neighborhoods. The initial solution is computed with the FF constructive algorithm, since this method has obtained the best results. Fig. 4 depicts the proportional deviation and gap distribution for both variants. Note that the order in which local search operators are applied does not affect so much the solution quality. Indeed, the U test reviews that those differences are not significant ($p = 1$). The running time for both variants are similar: minimum: 0.04, average: 0.20 ± 0.13, maximum: 0.80 (in seconds).

8.3. BRKGA variations

By controlling the maximum number of local search iterations τ, we can obtain variations in the BRKGA. The first variation,



(a) Deviation percentage. (b) Gap percentage.

Fig. 4. Deviation and gap for solutions of VND algorithm when compared to CPLEX solutions.

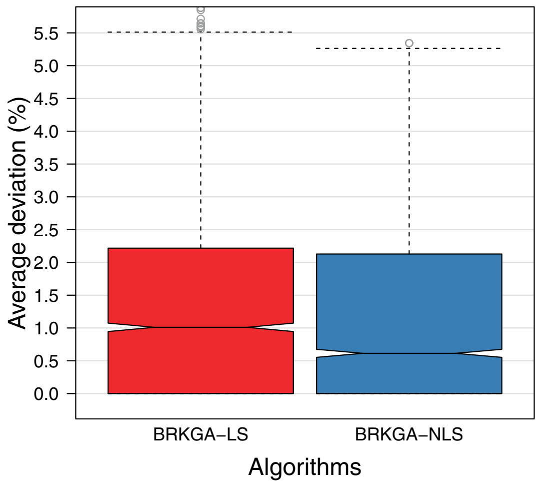


Fig. 5. Percentile deviation of the results of BRKGA variations from the best results found for each instance.

where $\tau = 0$ and it is called BRKGA-NLS, the optimization relies only in the evolutionary mechanism and the decoder only is responsible to build and evaluate the solution. The second variation, where $\tau > 0$ and it is called BRKGA-LS, the decoder applies a VNS-like local search where τ limits the number of local search iterations. Therefore, BRKGA-LS has an additional intensification step.

Fig. 5 depicts the average deviation from the best results obtained among the BRKGA variations. Surprisingly BRKGA-NLS presented less variation in the results than BRKGA-LS. This difference is not statistically significant at 95% confidence level, according the U test. On average, BRKGA-LS presented 1.51 $\pm$ 1.82% of deviation from the best results BRKGA-NLS resulted on 1.37 $\pm$ 1.88% of deviation. Indeed, BRKGA-LS took much more time (5421 $\pm$ 3941 seconds) than BRKGA-NLS (1193 $\pm$ 2228 seconds) to converge to a good solution. The reason is that the local search procedure uses the most time of the algorithm, even the solutions are improved. BRKGA-LS only performs 13 $\pm$ 20 iterations on average, which reduces the influence of the learning effect of BRKGA evolution. In contrast, BRKGA-NLS evolves about 8305 $\pm$ 17,330 iterations.

8.4. Comparison analysis

Since each algorithm has some variations, we picked the best variations to compare in this section. We used the results from ILS-Ins+Chg, and, for shortness we call it only by ILS in this section. Note that IGS applied a local search first in the Insert than in the Interchange neighborhoods. This design was deliberately chosen as it was for the ILS method. We use the BRKGA-NLS variation (without local search) due to its superior results when compared to BRKGA-LS. For shortness, we call BRKGA-NLS only by BRKGA in this section. For each strategy, we consider two variations such the initial solution is created using (1) IECT heuristic and (2) the FF heuristic. We append the respective suffixes in the name of the strategy.

Fig. 6 depicts the box plot of the percentile deviation of the results of each algorithm with respect to the best solutions found for each instance. CPLEX presented much worse results when compared to the other approaches.

Among the heuristics, BRKGA-FF presented the smallest deviation followed by ILS-FF. In general, strategies using FF presented better results than the ones using IECT.

To confirm these results, in Table 3 describes the results for U test applied to each pair of algorithms. The description of this table is similar to Table 2. However, negative numbers in the upper-right diagonal indicates that the “row algorithm” has less deviation with relation to the best results than the “column algorithm”, i.e., negative numbers shows evidence that the “row algorithm” is better than the “column algorithm”. According to these tests, all algorithms differ significantly since no p-value is larger than 0.05 for a confidence interval of 95%, except BRKGA-IECT and IGS-FF, since p-value is 0.21. BRKGA-FF presented the best results by far with an average deviation of 0.62 $\pm$ 1.87. All other algorithms presented a median deviation of at least 2%. CPLEX could not present good results in these experiments obtaining an average deviation of 10.29 $\pm$ 22.38.

Table 4 reports the performance of the algorithms considering the instances partitioned into two sets. The first column is the name of the algorithm. The first group of columns (2–6) shows the performance considering 66 instances (22% of) for which an solution was proven optimal. There, column “# OPT” represents the number of instances for which the algorithm found an optimal solution; column “% Opt” shows a percentage of the number of optimal solutions found; and column “% Run” shows a percentage of the number of runs on which the algorithm found an optimal solution. The two columns under label “Prop. diff.” show, respectively, the average of the proportional difference between the optimal solution value and the achieved value (%), and its corresponding standard deviation (σ). BRKGA-FF was able to find almost all optimal solutions and presented small deviation from ones it could not find the optimal solution. Also note that strategies using IECT could not find an optimal solution. However, average proportional deviation of all algorithms is similar, except CPLEX and IGS-FF.

The second group of columns (7–11) of Table 4 reports the performance of the algorithms on the 234 instances (78% of) where no solution was proven optimal. It follows the same structure of columns 2–6 but instead of comparing the algorithms with the optimum solution values, we compared them using the best known solutions. Again, BRKGA-FF excelled in its results when compared to the other strategies. Both variations of IGS presented slightly better results when compared with results for optimal solutions.

Fig. 7 shows performance profiles for all algorithms where X axis shows the time needed to reach a target solution value (in log scale), while the Y axis shows the cumulative probability to reach a target solution value for the given time in the X axis. Each algorithm is characterized by a different performance profile curve made up of (time, cumulative probability) pairs, one for each execution of the algorithm on a particular instance. The target values are the best solution for each instance. The blue line with circles represents the empirical cumulative probability of the BRKGA-FF, which presents the largest probability among all approaches in finding the best solutions in very few minutes. Note that BRKGA-FF can find about 30% of the best solutions in about 15 minutes, and it takes about 2 hours to find more than 40% of such solutions. After BRKGA, only ILS-FF (orange line with crossed squares) was able to overcome the 10% mark in about 75 minutes. On the supplementary materials, one can find the detailed results for each instance.

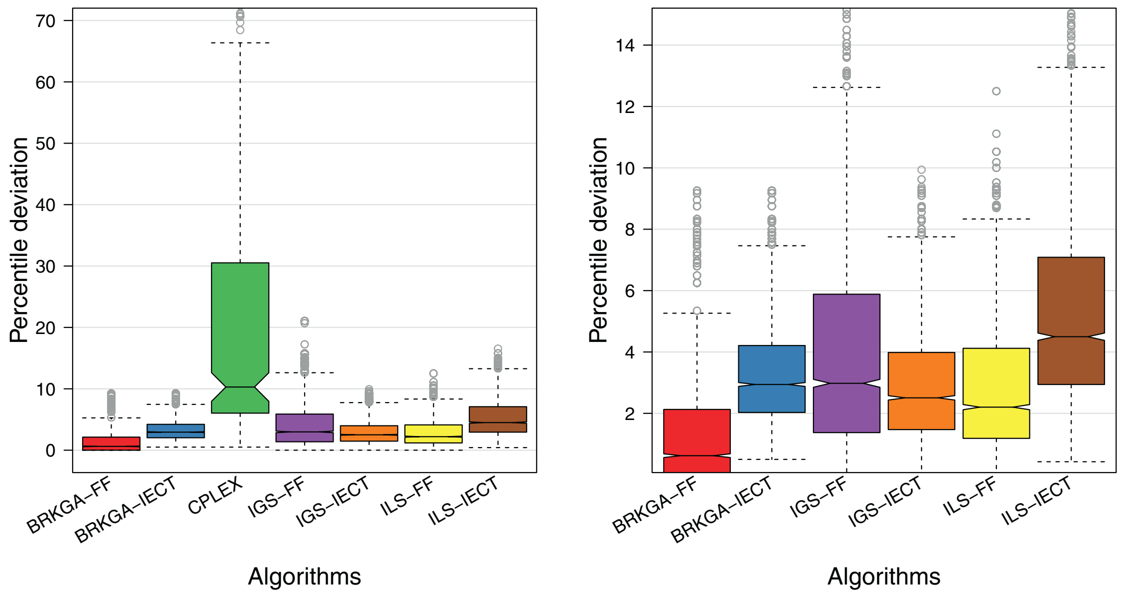


Fig. 6. Percentile deviation of the results of the algorithms considering the best results found for each instance.

Table 3
Difference in median location for gap distributions for all instances. The main diagonal shows the median of the gaps of each algorithm. We use a confidence interval of 95% and omit *p*-values are less than 0.05.

	BRKGA-FF	BRKGA-IECT	CPLEX	IGS-FF	IGS-IECT	ILS-FF	ILS-IECT
BRKGA-FF	0.62	−1.98	−9.09	−1.98	−1.54	−1.26	−3.43
BRKGA-IECT		2.94	−7.18	−0.07	0.40	0.62	−1.54
CPLEX			10.29	7.06	7.56	7.75	5.69
IGS-FF		0.21		2.98	0.47	0.69	−1.36
IGS-IECT					2.51	0.21	−1.93
ILS-FF						2.20	−2.12
ILS-IECT							4.50

Table 4
Algorithm performance on instances with known and unknown optimum solutions. BRKGA stands for BRKGA-NLS here.

Algorithm	Known optima (66 instances)					Unknown optima (234 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
BRKGA-FF	61	92.42	39.24	3.04	2.67	201	85.90	32.99	2.03	1.73
BRKGA-IECT	0	0.00	0.00	3.46	2.56	0	0.00	0.00	3.22	1.57
CPLEX	0	0.00	0.00	9.75	15.24	0	0.00	0.00	24.38	23.13
IGS-FF	1	1.52	1.52	4.87	4.68	7	2.99	2.99	4.41	3.39
IGS-IECT	0	0.00	0.00	3.47	2.63	7	2.99	0.64	2.89	1.75
ILS-FF	3	4.55	2.42	3.60	3.08	17	7.26	4.66	2.98	2.01
ILS-IECT	0	0.00	0.00	4.80	3.31	0	0.00	0.00	5.26	2.89

9. Concluding remarks

In this work, we introduced a new permutation flowshop problem with a stepwise job objective function as well as its mathematical model and a benchmark instances for this problem. For this problem, this work proposed a new constructive heuristic, heuristics based on a hybrid Iterated Local Search (ILS) and iterated greed search (IGS), and a biased random-key genetic algorithm (BRKGA). Both methods are somehow combined with a VND algorithm. Additionally, relaxations were applied to provide upper bounds for the problem.

Extensive experiments on 300 test instances showed that BRKGA using warmup solution obtained with the FF heuristic was able to produce excellent results when compared to other strategies. Indeed, FF heuristic presented itself as a valuable method in creating initial solutions that are improved afterwards through the high level strategies. It is interesting to note that, while the order of neighborhoods in the VND in the local search step of the ILS does not have effect in the quality of the results, the VND is necessary to improve the results when compare to a simple local search in the Insert neighborhood.

We would like to point out that that optimal solutions were found for 66 out of 300 instances, but the primal-dual gap for the rest of the instances varies up to 8%. Therefore, this work open a new research avenue where further investigations would be interesting. For example, structural properties of the problem could be identified aiming to design better upper bounds and to provide a more accurate validation of the heuristic methods quality.

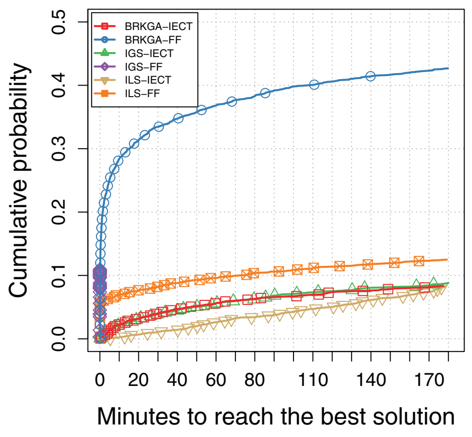


Fig. 7. Running time distributions to best solutions found. The identification marks correspond to 2% of the points plotted for each algorithm. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

10. Disclaimer

All presented information does neither reflect nor imply actual business cases for AT&T and/or associated companies. They are generalized data used to assess the performance of the algorithms only.

Acknowledgment

The authors wish to thank the anonymous reviewers for providing comments that improved this paper. This research was partially supported by PNPD/Capes and PUC-Rio. The first author would like thank Safia Kedad-Sidhoum for providing the initial motivation for the production of this work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2017.10.045.

References

Andrade, C. E., Ahmed, S., Nemhauser, G. L., & Shao, Y. (2017). A hybrid primal heuristic for finding feasible solutions to mixed integer programs. *European Journal of Operational Research, 263*(1), 62–71.

Andrade, C. E., Miyazawa, F. K., & Resende, M. G. C. (2013). Evolutionary algorithm for the *k*-interconnected multi-depot multi-traveling salesmen problem. In *Proceedings of the fifteenth annual conference on genetic and evolutionary computation. GECCO'13* (pp. 463–470). New York, NY, USA: ACM.

Andrade, C. E., Resende, M. G. C., Karloff, H. J., & Miyazawa, F. K. (2014). Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the sixteenth conference on genetic and evolutionary computation. GECCO'14* (pp. 405–412). New York, NY, USA: ACM.

Andrade, C. E., Resende, M. G. C., Zhang, W., Sinha, R. K., Reichmann, K. C., Doverspike, R. D., & Miyazawa, F. K. (2015a). A biased random-key genetic algorithm for wireless backhaul network design. *Applied Soft Computing, 33*, 150–169.

Andrade, C. E., Toso, R. F., Resende, M. G. C., & Miyazawa, F. K. (2015b). Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions. *Evolutionary Computation, 23*, 279–307.

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological), 57*(1), 289–300.

den Besten, M., & Stützle, T. (2001). Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. In *Proceedings of the fourth metaheuristics international conference (MIC2001)* (pp. 545–549).

Brucker, P. (2004). *Scheduling algorithms*. SpringerVerlag.

Caserta, M., & Reiners, T. (2016). A pool-based pattern generation algorithm for logical analysis of data with automatic fine-tuning. *European Journal of Operational Research, 248*(2), 593–606.

Chalghoumi, S., Mrad, M., & Ladhari, T. (2013). A new lower bound for minimising the total completion time in a two-machine flow shop under release dates. In *Proceedings of the fifth international conference on modeling, simulation and applied optimization (ICMSAO)* (pp. 1–4). IEEE.

Chen, C.-L., Vempati, V. S., & Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research, 80*(2), 389–396.

Conover, W. J. (1980). *Practical nonparametric statistics*. 2nd ed.: John Wiley & Sons.

Curry, J., & Peters, B. (2005). Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. *International Journal of Production Research, 43*(15), 3231–3246.

Della Croce, F., Gupta, J., & Tadei, R. (2000). Minimizing tardy jobs in a flowshop with common due date. *European Journal of Operational Research, 120*, 375–381.

Della Croce, F., Gupta, J., & Tadei, R. (2011). A matheuristic approach for the total completion time two-machines permutation flow shop problem. *EvoCOP*, 38–47.

Detienne, B., Dauzère-Pérès, S., & Yugma, C. (2011). Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling, 14*(6), 523–538.

Dong, X., Huang, H., & Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers and Operations Research, 36*(5), 1664–1669.

Ericsson, M., Resende, M. G. C., & Pardalos, P. M. (2002). A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization, 6*(3), 299–333.

Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research, 207*(1), 55–69.

Fay, M. P., & Proschan, M. A. (2010). Wilcoxon–Mann–Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys, 4*, 1–39.

Fernandez-Viagas, V., & Framinan, J. M. (2015). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers and Operations Research, 53*, 68–80.

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research, 257*(3), 707–721.

Framinan, J. M., Gupta, J. N., & Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society, 55*(12), 1243–1255.

Glover, F., & Kochenberger, G. (2003). *Handbook of metaheuristics*. Kluwer: Academic Publishers.

Gonçalves, J. F., & de Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics, 8*(6), 629–642.

Gonçalves, J. F., & Resende, M. G. C. (2011a). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*, 487–525.

Gonçalves, J. F., & Resende, M. G. C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization, 22*(2), 180–201.

Gonçalves, J. F., Resende, M. G. C., & Toso, R. F. (2014). An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional, 34*, 143–164.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 4*, 287–326.

Gupta, J. N., & Stafford, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research, 169*(3), 699–711.

Hansen, P., & Mladenović, N. (2003). Variable neighborhood search. In F. Glover, & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 145–184). Kluwer Academic Publishers.

Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics (NRL), 1*(1), 61–68.

Ladhari, T., & Rakrouki, M. A. (2009). Heuristics and lower bounds for minimizing the total completion time in a two-machine flowshop. *International Journal of Production Economics, 122*(2), 678–691.

Liu, J., & Reeves, C. R. (2001). Constructive and composite heuristic solutions to the $p//\Sigma c_i$ scheduling problem. *European Journal of Operational Research, 132*(2), 439–452.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.

Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated local search. In F. Glover, & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 321–353). Kluwer Academic Publishers.

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

Nikolaev, A. G., & Jacobson, S. H. (2010). Simulated annealing. In *Handbook of metaheuristics* (pp. 1–39). Springer.

Onar, S. c., Öztayşi, B., Kahraman, C., Yanık, S., & Şenvar, Ö. (2016). *A literature survey on Metaheuristics in production systems* (pp. 1–24). Cham: Springer International Publishing. doi:10.1007/978-3-319-23350-5-1.

Pan, Q.-K., & Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers and Operations Research, 40*(1), 117–128.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

Potts, C. (1985). Analysis of heuristics for two-machine flow-shop sequencing subject to release dates. *Mathematics of Operations Research, 10*(4), 576–584.

Raidl, G. (2006). A unified view on hybrid metaheuristics. In F. Almeida, M. Aguilera, C. Blum, J. M. Vega, M. Pérez, A. Roli, & M. Sampels (Eds.), *Hybrid metaheuristics*. In *Lecture notes in computer science: 4030* (pp. 1–12). Heidelberg: Springer Berlin.

Raidl, G., & Puchinger, J. (2008). Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. B. Aguilera, A. Roli, & M. Sampels (Eds.), *Hybrid metaheuristics* (pp. 31–62). Springer.

Rakrouki, M., Kooli, A., Chalghoumi, S., & Ladhari, T. (2017). A branch-and-bound algorithm for the two-machine total completion time flowshop problem subject to release dates. *Operational Research*.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers and Operations Research, 22*(1), 5–13.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494.

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega, 34*(5), 461–476.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.

Ruiz, R., & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research, 187*(3), 1143–1159.

Seddik, Y., Gonzales, C., & Kedad-Sidhoum, S. (2013). Single machine scheduling with delivery dates and cumulative payoffs. *Journal of Scheduling, 16*(3), 313–329.

Seddik, Y., Gonzales, C., & Kedad-Sidhoum, S. (2015). Performance guarantees for a scheduling problem with common stepwise job payoffs. *Theoretical Computer Science, 562*, 377–394.

Stefanello, F., Aggarwal, V., Buriol, L. S., Gonçalves, J. F., & Resende, M. G. C. (2015). A biased random-key genetic algorithm for placement of virtual machines across geo-separated data centers. In *Proceedings of the 2015 on genetic and evolutionary computation conference. GECCO'15* (pp. 919–926). New York, NY, USA: ACM.

Stützle, T. (1998). *Applying iterated local search to the permutation flow shop problem*. University of Technology: Darmstadt.

Talbi, E.-G. (2009). *Metaheuristics – from design to implementation*. Wiley.

Tasgetiren, M. F., Kizilay, D., Pan, Q.-K., & Suganthan, P. (2017). Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Computers and Operations Research, 77*, 111–126.

Vallada, E., Ruiz, R., & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers and Operations Research, 35*(4), 769–780.

Whitley, D., Rana, S., & Heckendorn, R. B. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology, 7*, 33–47.

Zhang, Y., Li, X., & Wang, Q. (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research, 196*(3), 869–876. http://www.sciencedirect.com/science/article/pii/S0377221708003925.