Expert Systems With Applications 128 (2019) 67–80

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm

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

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

ARTICLE INFO

Article history:
Received 26 October 2018
Revised 3 March 2019
Accepted 4 March 2019
Available online 9 March 2019

Keywords:
Flowshop scheduling
Biased random-key genetic algorithm
Metaheuristics

ABSTRACT

In this paper, we advance the state of the art for solving the Permutation Flowshop Scheduling Problem with total flowtime minimization. For this purpose, we propose a Biased Random-Key Genetic Algorithm (BRKGA) introducing on it a new feature called *shaking*. With the shaking, instead to full reset the population to escape from local optima, the *shaking* procedure perturbs all individuals from the elite set and resets the remaining population. We compare results for the standard and the *shaking* BRKGA with results from the Iterated Greedy Search, the Iterated Local Search, and a commercial mixed integer programming solver, in 120 traditional instances. For all algorithms, we use warm start solutions produced by the state-of-the-art Beam-Search procedure. Computational experiments show the efficiency of proposed BRKGA, in addition to identify lower and upper bounds, as well as some optimal values, among the solutions.

1. Introduction

Scheduling problems refer to the assignment of jobs to machines – productive resources – as well as the definition of the processing start and end times for each job in each machine, in order to optimize one or more criteria. This class of problems is among the most studied in the operational research field and presents applications in several areas (Pinedo, 2016). Fuchigami and Rangel (2018) review articles published in the last 25 years and identify case studies in various productive sectors. According to the authors, scheduling problems found applications in a variety of fields, where a sequence of tasks must be executed. They include the primary, processing and assembling industries, as much as in the telecommunication and transportation industries, education, government, and among others.

One of the most important scheduling problems is the Permutation Flowshop Scheduling Problem (PFSP). This problem considers a set of n jobs to be scheduled on a set of m machines. Each job j has a processing time $p_{j\ell}$ on machine ℓ, and must be scheduled on each machine in sequence and, therefore, can be executed only one machine at time. Given a schedule S, we denote by $C_{j\ell}$ the completion time of job j on machine ℓ, and by C_j the completion time of the job j on the last machine. This work deals with the PFSP whose objective function value is given by the summation of C_j overall jobs, also known as total flow time.

Due to its NP-hard nature (Garey, Johnson, & Sethi, 1976), few exact methods have been developed for solving it to optimality. Among them are the branch-and-bound methods of Chung, Flynn, and Kirca (2002), Della Croce, Ghirardi, and Tadei (2002), Madhushini, Rajendran, and Deepa (2009), and van de Velde (1991). On the other hand, there are many heuristic approaches in the literature. Extensive comparative studies on heuristic methods were developed by Framinan, Leisten, and Ruiz-Usano (2005) and Pan and Ruiz (2013).

Regarding constructive algorithms, Pan and Ruiz (2013) present a comprehensive set of methods. This survey described the LR constructive method proposed by Liu and Reeves (2001) with $O(n^3m)$ running time. Such method was improved by Fernandez-Viagas and Framinan (2015), leading to the $O(n^2m)$ FF algorithm. Afterward, Abedinnia, Glock, and Brill (2016) and Rossi, Nagano, and Sagawa (2017) described new constructive methods which, despite some potential results on the quality of the solutions, have a higher complexity than FF thus making them inefficient for larger instances. Recently, Fernandez-Viagas and Framinan (2017) proposed a Beam-Search Constructive Heuristic (BSCH) based on the FF method by Fernandez-Viagas and Framinan (2015).

Other methods applied to PFSP are the metaheuristics that constitute a powerful tool to explore the search space by combining simple constructive and local search methods with a (pseudo) random mechanism to escape from local optima (Martí, Panos, & Resende, 2017). Pan and Ruiz (2012) analyse comparatively the best

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

https://doi.org/10.1016/j.eswa.2019.03.007
0957-4174/

performing metaheuristic methods developed so far including genetic algorithms (Tseng & Lin, 2009; 2010; Xu, Xu, & Gu, 2011; Zhang, Li, & Wang, 2009), estimation of distribution algorithms (Jarboui, Eddaly, & Siarry, 2009), discrete differential evolution algorithms (Pan, Tasgetiren, & Liang, 2008; Tasgetiren, Pan, Suganthan, & Chen, 2011), iterated greedy algorithms (Dubois-Lacoste, López-Ibáñez, & Stützle, 2011; Pan et al., 2008), iterated local searches (Dong, Huang, & Chen, 2009), among others. Additionally, Pan and Ruiz (2012) included in the analysis new iterated greedy and iterated local search methods as well as their population-based variants. A comparative evaluation showed the superiority of the new heuristics concerning the existing methods, not to mention the simple iterated search algorithms performed better than the variant based on a population scheme.

To the best of our knowledge, the state-of-the-art in heuristic methods for the permutation flowshop problem with total flowtime criterion was obtained by Fernandez-Viagas and Framinan (2017) when applying the beam-search constructive heuristic in the framework of an Iterated Local Search (ILS) proposed by Dong, Chen, Huang, and Nowak (2013). Computational experiments revealed the efficiency of the BSCH as a constructive heuristic when applied in isolation as well as the benefit of using it to generate initial solutions for a metaheuristic method.

In this context, we observe that significant progress has been achieved by applying hybrid methods based on local search and constructive algorithms. However, population-based methods seem to remain underexplored. Therefore, this paper aims to fill this gap by presenting a Biased Random-Key Genetic Algorithm (BRKGA) for minimizing total flowtime in a permutation flowshop problem. The proposed method incorporates the beam-search constructive heuristic (Fernandez-Viagas & Framinan, 2017) as well as a new feature called *shaking* to the standard BRKGA (Gonçalves & Resende, 2011) to evolve the population of solutions. Another contribution of this paper refers to the establishment of primal and dual bounds by running the related Integer Linear Program (ILP) model on a commercial solver, as much as the proof of optimality for some instances. Computational study on 120 classical benchmark instances from literature was carried out to demonstrate the effectiveness of the proposed method.

The remainder of this paper is organized as follows. Section 2 brings the formal definition and the mathematical model of the problem. Section 3 describes biased random-key genetic algorithms as well as its customization for tackling the flowshop problem with total flowtime minimization criterion. Additionally, an Iterated Greedy Search and an Iterated Local Search are presented, respectivelly, in Sections 4 and 5. In order to evaluate our proposals, extensive computational experiments were carried out. Detailed results and analysis are shown is Section 6. Concluding remarks are in Section 7.

2. Problem definition and formulation

Permutation flowshop scheduling problems require that the jobs be schedule in sequence, regardless the time they are schedule. Therefore, any job's permutation generates a valid schedule. In the permutation flowshop scheduling problems (PFSP) with total flowtime minimization criterion, a set of n jobs $J = \{1, 2, \ldots, n\}$ have to be scheduled, following the same order, on a set of m machines $M = \{1, 2, \ldots, m\}$. Each job $j \in J$ has a processing time $p_{j\ell} > 0$ on machine $\ell \in M$. Job processing interruption is not allowed and all parameters are integer numbers. Given a schedule S, we denote by $C_{j\ell}$ the completion time of job j on machine ℓ. We denote by C_j the completion time of the job j in the last machine. According to the three-field notation of Graham, Lawler, Lenstra, and Kan (1979), the problem under study can be defined as $F_m|prmu|\sum_{j=1}^{n} C_j$.

Mixed integer programming model (1a)-(1g) is based on the positional-variables formulation of Wagner (1959), and described by Stafford (1988) and Della Croce, Gupta, and Tadei (2011). In that model, let X_{ij} be the binary decision variable such that $X_{ij} = 1$ if job i is the j^{th} job of the sequence and $X_{ij} = 0$ otherwise. Let $C^{pos}_{j\ell} \geq 0$ denote the completion time of the job on the j^{th} position on machine $\ell \in M$. Assume that $C^{pos}_{0\ell} = 0$ for all $\ell \in M$.

$$\min \quad \sum_{j=1}^{n} C^{pos}_{jm} \tag{1a}$$

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

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

$$C^{pos}_{j\ell} \geq C^{pos}_{j-1,\ell} + \sum_{i=1}^{n} p_{i\ell} X_{ij} \qquad \begin{array}{l}\forall j \in \{1, \ldots, n\} \\ \forall \ell \in \{1, \ldots, m\},\end{array} \tag{1d}$$

$$C^{pos}_{j,\ell+1} \geq C^{pos}_{j\ell} + \sum_{i=1}^{n} p_{i,\ell+1} X_{ij} \qquad \begin{array}{l}\forall \ell \in \{1, \ldots, m-1\} \\ \forall j \in \{1, \ldots, n\},\end{array} \tag{1e}$$

$$C^{pos}_{j\ell} \geq 0 \qquad \begin{array}{l}\forall j \in \{1, \ldots, n\} \\ \forall \ell \in \{1, \ldots, m\},\end{array} \tag{1f}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, \ldots, n\}. \tag{1g}$$

Objective Function (1a) aims to minimize the sum of the completion times of all jobs. Constraints (1b) and (1c) ensure that each job is attributed to one position and vice-versa. Constraint (1d) prevents the overlapping of a job and its predecessor on a machine. Constraint (1e) ensures that a job cannot be processed on a machine before its completion on the preceding machine. Constraints (1f) and (1g) give the definition domain of the variables.

3. Biased random-key genetic algorithm

The Biased Random-Key Genetic Algorithm (BRKGA) has been applied to several optimization problems, as reviewed in Prasetyo, Fauza, Amer, and Lee (2015), including those in the telecommunications context (Resende, 2012), hub location (Lopes, Andrade, Queiroz, Resende, & Miyazawa, 2016; Pessoa, Santos, & Resende, 2017), facility location (Biajoli, Chaves, & Lorena, 2019), and vehicle routing (Grasas et al., 2014; Prasetyo, Putri, & Fauza, 2018). When it comes to scheduling problems, recent works tackled a bi-objective batching machine scheduling problem (Cabo, González-Velarde, Possani, & Ríos Solís, 2018), a single batch processing machine scheduling (Li & Zhang, 2018), project scheduling problems (Almeida, Correia, & Saldanha-da Gama, 2018), and a flowshop problem with stepwise objective function with payoff maximization criterion (Pessoa & Andrade, 2018). BRKGA has been used sucessfully also in large-scale real problems such as network planning (Andrade et al., 2015a) and wireless updates on connected cars (Andrade et al., 2017).

BRKGA is a population-based metaheuristic where each chromosome encodes a solution through a vector of random numbers. The biased mating of two parents is responsible for inducing the offspring inheritance from the best parent characteristics (Gonçalves & Resende, 2011). The initial population is created with p chromosomes, each one composed of n genes. A gene is a random real number (*random-key*) uniformly drawn from the interval [0,1]. A chromosome is mapped into a feasible solution by a *decoder* function which also computes the cost of such a solution. In

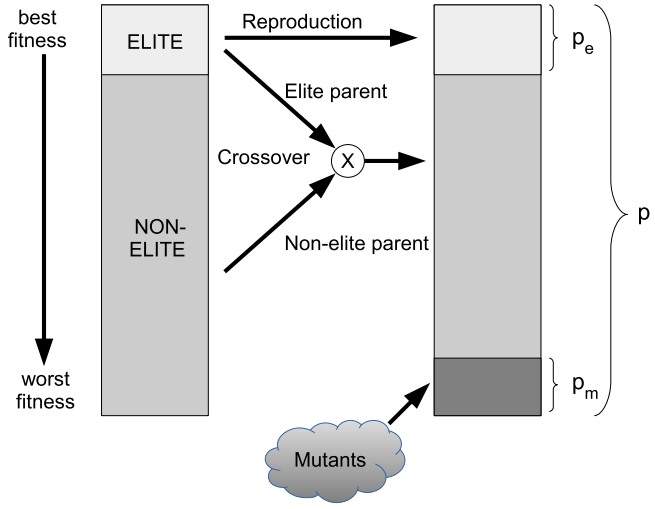

Fig. 1. BRKGA scheme for population evolution.

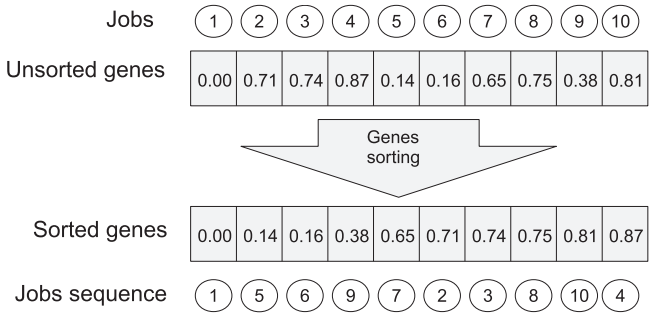

Fig. 2. A decoder for scheduling problems.

the genetic algorithm context, the cost of a solution is referred to as the *fitness of the individual* which is related to its capacity to survive through generations.

As illustrated in Fig. 1, after all individuals have their fitness value determined, the population is divided into two sets: the *elite set*, containing the fittest individuals, and the *non-elite set*, with the rest of the population. Afterward, the population evolves to a new generation in a three steps process while keeping its size. First, in the *reproduction* step, the elite set, containing p_e individuals is entirely copied to the new population. Then, the worst p_m non-elite individuals are replaced by *mutants* generated randomly in the same way that the initial population. The third step, called *crossover* or *mating*, is accomplished with the production of $p - p_e - p_m$ offsprings. The generation of an offspring requires the random choice of a parent from the elite set, and another parent from the non-elite set. Besides, a parameterized uniform crossover scheme applies a probability $\rho > 0.5$ of inheriting each gene value from the elite parent.

Once the reproduction, mutation and crossover steps are finished, each solution is given to the decoder so that the fitness of each individual is calculated, starting a new evaluation-selection-evolution cycle. The population evolves generation after generation until a stopping criterion is reached. In other words, the algorithm runs until a maximum processing time or a number of iterations is met. Following, we customize a biased random-key genetic algorithm for the flowshop problem with total flowtime minimization criterion.

3.1. Representation and decoder

A chromosome that encodes a solution for a flowshop problem is composed of n genes, i.e., a random-key for each job, as shown in Fig. 2. The decoder sorts the vector of random-keys so that the sequence of jobs related to the sorted genes corresponds to a feasible solution for the problem.

3.2. Warm-start

The standard implementation of a BRKGA creates the initial population by randomly generating all individuals. However, Reeves (1995), Zhang et al. (2009), Andrade, Miyazawa, and Resende (2013), Andrade, Toso, Resende, and Miyazawa (2015b), Lopes et al. (2016), and Pessoa and Andrade (2018) have shown the benefits of providing good solutions as chromossomes in the initial population. In this work, we used solutions from the Beam Search Constructive heuristic (BSCH – Fernandez-Viagas & Framinan, 2017) as warm-starters to BRKGA.

BSCH builds, level by level, a search tree with a partial solution in each node while solutions are constructed element by element. BSCH is parameterized so that the number of nodes explored at each level can vary from only one node to x nodes. In the case where BSCH explores only one node at each iteration, this method becomes equivalent to the simplest FF constructive method (Fernandez-Viagas & Framinan, 2015). FF builds a job sequence from scratch by appending jobs at the end of a partial solution. At each iteration, all jobs that are not in the partial schedule are (re)evaluated according to an index function that considers the completion time and the idle time originated by the placement of such a job at the end of the partial solution. BSCH behaves in a similar way as FF, but when exploring more than one node, BSCH also computes an estimate of which nodes are the most promising to be kept as exploring branches. A forecast index is used to compare different nodes by computing (additionally to the idle time and the completion time of scheduled jobs) an estimate of the contribution of unscheduled jobs to the completion time and, consequently, to the objective function. For more details, refer to Fernandez-Viagas and Framinan (2017).

3.3. Local search

The hybridization of evolutionary algorithms with local search was proposed by Moscato (1989) with the goal of improving the fitness of an individual by refining its characteristics. Indeed, local searches modify some components of an initial solution to take it to a local minimum in its neighborhood.

For flowshop problems, two methods are commonly used: *interchange* and *insert* moves (den Besten & Stützle, 2001). When the interchange move is applied, a neighbor solution is reached by swapping two jobs from their original positions. On the other hand, the insert method explores the neighborhood of a solution by moving a job to another position.

By following an iterative improvement strategy, once an initial solution is determined, the local search method is applied to it, and an improved neighbor is chosen to replace it in the next iteration. Otherwise, if no improved solution is found, the initial solution is returned as locally optimum (Hoos & Stützle, 2004). Two or more local search methods can be combined in order to better explore the solution space through movements over different neighborhoods. A procedure that orchestrates the calls is the Variable Neighborhood Search (VND – Hansen & Mladenović, 2003), which determine the order for applying the local search methods.

The interchange and insert neighborhood sizes are, respectively, $n(n-1)/2$ and $(n-1)^2$. Since exploring insert neighborhood is the most time-consuming method, we allow only one iteration on it, i.e., we only explore the first-degree neighbors and return the

best found. On the other hand, since interchange neighborhood is smaller than the insert neighborhood, we can afford to search for the local minimum.

Therefore, our local search is VND-based, where in each iteration, we search in the insert neighborhood first, and then, we explore the interchange neighborhood searching for local optima. The main VND loop stops when no improved solution is found, or maximum BRKGA time is reached.

Applying the local search for each individual of the population usually does not help to reach reasonable solutions, since it is time-expensive and impairs BRKGA to evolve the number of generations needed to good performance. Therefore, we call the local search only on the best individual of the population on each τ iterations, and the best individual value is different from the best solution found so far.

3.4. Shaking

In some situations, the diversity of the population can be compromised, i.e., when there is no difference between the fitness values of the whole elite population or when there are too many iterations without improvement. In such situations, it is usual to restart the search by resetting the whole population. However, this multi-start strategy can be too drastic because it destroys the convergence structure of the population. In other words, the full resetting destroys useful parts of solutions (block of genes) built over multiple generations of chromosome crossovers. These well-conserved parts of the genome usually appear in good solutions. To avoid such a drastic move, in this work we introduce a new BRKGA feature called *shaking*.

Instead of fully resetting the whole population, the shaking method applies random modifications to the individuals in the elite set, and resets the non-elite individuals. Therefore, we guarantee diversity on the non-elite set and preserve the convergence structure on the shaken elite set. Algorithm 1 describes the shaking method, which is controlled by a shaking intensity parameter ψ. The main loop (lines 1–6) iterates over all individuals in the elite set. Each elite individual is decoded such that a solution S is extracted from it (line 2). The inner loop (lines 3–5) applies the perturbations ψ times. In the first perturbation, the algorithm swaps the positions of a randomly chosen job i and its neighbor (line 4). The second perturbation swaps positions of two randomly chosen jobs (line 5). The algorithm then re-encodes the solution as a chromosome in the elite set (line 6). To finish, the algorithm replaces all non-elite chromosomes by random individuals (line 7), and returns the new population (line 8). Note that the shaking is done in the problem solution space rather than the BRKGA representation space. Therefore, the algorithm decodes the chromosome, applies the perturbations, and then re-encode the shaken solution to the BRKGA space. Note that this can be done without re-encoding the solution by tracking the order of the jobs in the chromosome.

Algorithm 1: Shaking.

Input : Population P where P_e is the elite set and P_{ne} is the non-elite set; shaking intensity ψ.
Output: Changed population.

1 **foreach** $ind \in P_e$ **do**
2 Let S be the solution induced by ind;
3 **for** $k \leftarrow 1$ **to** ψ **do**
4 Swap two neighboring jobs $\langle i, i+1 \rangle$ in S, where i is uniformly chosen from $[1, n-1]$;
5 Swap a pair jobs $\langle i, j \rangle$ in S, at the ith $\neq$ jth uniformly chosen random positions;
6 Encode ind according to S;
7 Replace the members of P_{ne} with random samples;
8 **return** *changed population P.*

Although we can do the shaking anytime during the evolutionary process, we chose to apply it in the following situations:

- There were many iterations (R) without improvements on the best solution of the current population. In this situation, the number of perturbations is fixed proportion of the number of jobs, i.e., $\psi = \lambda n$ where n is the number of jobs, and λ is a fixed value in the interval [0, 1];
- When the diversity of the population was compromised, i.e, there was no difference between the fitness values of the whole elite population. In this case, $\psi = \lambda n$ where λ is uniformly drawn from the interval [0.05, 0.20];
- There was also a special case when there was no improvement on the overall best solution for too many iterations ($R^\star$). For this case, the perturbation is similar to the previous one but more aggressive, selecting λ from the interval [0.5, 1.0];

In some cases, even the shaking cannot help BRKGA escape from local optima. In such situations, we use the traditional full population restart keeping the warm-start solution though. We call the full restart when the algorithm reaches $R^{\star\star}$ iterations without improvement in the best solution.

Algorithm 2 depicts the overall framework. The algorithm starts with a population of random individuals and a warm-start solution from other methods and initialize some local variables (lines 2–5). In the main loop (lines 6–31), the algorithm first evolves the population according to BRKGA rules (omitted for shortness; refer to Gonçalves & Resende, 2011 for complete evolution procedure). On every L iterations, the local search is executed (line 9). In the next blocks, the algorithm calls the shaking procedure with different intensities according to the population status. If the current solution has not been improved in the last R iterations, the algorithm performs a random shaking (line 13). At this point, if the population is too homogeneous, a weak shaking is performed (line 16). Next, the algorithm checks for updates in the best solution and performs a strong shaking if the best solution has not improved for the last $R^\star$ iterations (line 21). However, as described earlier, the shaking may be not enough. In this case, the algorithm performs a full-population reset if it reaches $R^{\star\star}$ iterations without improvements in the best solution (line 26).

4. Iterated greedy search

In order to conduct a comparative study with the proposed BRKGA, we also implemented an Iterated Greedy Search (IGS) method. Ruiz and Stützle (2007) described the first application of IGS for the Permutation Flowshop Scheduling Problem (PFSP) with makespan criterion. The superiority of the method was shown by Pan et al. (2008) when compared to the Discrete Differential Evolution method of Pan, Tasgetiren, and Liang (2007) for problems with the makespan and the total flowtime minimization criteria. IGS was also used successfully for the sequence-dependent setup times flowshop problem with makespan and weighted tardiness objectives (Ruiz & Stützle, 2008), the unrelated parallel machine scheduling (Fanjul-Peyro & Ruiz, 2010), and the blocking PFSP with makespan criterion (Tasgetiren, Kizilay, Pan, & Suganthan, 2017).

IGS builds a chain of solutions by iteratively generating new starting solutions. At each iteration, the method partially destroys a starting solution and, next, reconstructs the solution to the feasibility. Then, a local search method is applied to the incumbent solution to find an improving neighbor. After each destruction-reconstruction-local search cycle, an acceptance criterion evaluates

Algorithm 2: General BRKGA framework with Shake.

Input : Limits of the number of iterations without improvement R, $R^{\star}$, $R^{\star\star}$;number of iterations to perform a local search L;number of jobs n; stop criteria.
Output: Best solution.

```
Initialize population P with random individuals and the warm-start solution;
bestsol ← warm-start solution;
lastsol ← warm-start solution;
bestsoliter ← 1;
iter ← 1;
while a stopping criterion is not reach do
    Evolve P for one generation;
    Let cursol be the value of the best solution in P;
    if iter = 0 mod L then
        Do a local search in the best individual of P;
    if cursol = lastsol then
        stableit ← stableit + 1;
        if stableit ≥ R then
            λ ← rand([0.0, 1.0]);
            shake(P, λn);
        if elite individuals in P have the same fitness then
            λ ← rand([0.05, 0.20]);
            shake(P, λn);
    else
        stableit ← 0;
    if cursol ≥ bestsol then
        bestsoliter ← bestsoliter + 1;
        if bestsoliter = R* then
            λ ← rand([0.50, 1.00]);
            shake(P, λn);
        if bestsoliter = R** then
            Replace population P random with individuals and
            the warm-start solution;
    else
        bestsoliter ← 0;
        bestsol ← cursol;
    lastsol ← cursol;
return the best solution bestsol.
```

such solution to decide whether it will replace the initial solution to the next iteration. This loop is repeated until a stopping criterion is met (Stützle & Ruiz, 2017).

When customizing an IGS heuristic to solve a problem, five procedures must be developed: (1) a deterministic or random method to construct an initial solution; (2) a destruction procedure to remove some components of a solution; (3) a reconstruction procedure that takes the partial solution incumbent from step (2) and generate a complete feasible solution; (4) a local search to explore the neighborhood of the solution reached by the reconstruction step; and (5) an acceptance criterion which balances between the highest diversity, when any new solution is accepted, or the strongest intensification, when only improving solutions replaces the initial one to the next (2)–(4) loop iteration. Note that the strength of the destruction, given by the number of affected components, must be calibrated to reflect a trade-off between diversification and intensification.

The IGS heuristic developed for the problem tackled in this work takes as initial solution the scheduling obtained by the beam-search constructive heuristic (Fernandez-Viagas & Framinan, 2017), described in Section 3.2. The destruction procedure removes a certain number of randomly chosen jobs from a candidate solution (given a destruction parameter), generating a partial solution. Then, the removed jobs are considered for being appended at the end of the partial solution, observing the evaluation function proposed by Fernandez-Viagas and Framinan (2015) for the FF constructive method, as mentioned in Section 3.2. The local search phase applies the VND method (Hansen & Mladenović, 2003). In this way, *insert* neighborhood is explored alternately with *interchange* neighborhood, both using the best improvement search strategy. The incumbent solution is accepted only if its cost is better than the best solution found so far.

5. Iterated local search

Iterated Local Search (ILS) has been applied for diverse flowshop problems (Juan, Lourenço, Mateo, Luo, & Castella, 2014; Ribas, Companys, & Tort-Martorell, 2013; Stützle, 1998; Xu, Wu, Yin, & Lin, 2017). When it comes to total flowtime minimization criterion, the reader is refered to the works of Dong et al. (2009), Dong, Chen, and Huang (2011a) and Dong, Huang, and Chen (2011b). However their results were overcame in the recent paper of Fernandez-Viagas and Framinan (2017) which shows the benefits of incorporating the beam-search procedure into the ILS method.

Analogously to the IGS metaheuristic, presented in Section 4, the ILS generates an initial local optimum solution, which is the starting point to explore the solution space. Then, a loop coordinates an iterative process of perturbation-improvement-evaluation steps until a stopping criterion is reached (Lourenço, Martin, & Stützle, 2019).

As proposed by Fernandez-Viagas and Framinan (2017), ILS applies the beam-search procedure to construct the first solution. This is the same strategy applied in the BRKGA and IGS methods implemented in this work.

Within the cycle of perturbation-improvement-evaluation, the perturbation step is performed by swapping two randomly chosen jobs. The number of swaps determines how much characteristics of the original solution will be kept. The greater this number, the greater the diversification. Therefore, this parameter must be carefully calibrated. Improvement step is accomplished by applying the Variable Neighborhood Descent (VND) strategy for local search, as for the IGS (Section 4), exploring alternately *insert* and *interchange* neighborhoods while enhancing solutions are found. Finally, the solution obtained by the improvement procedure is compared to the current best solution. The lowest cost solution is kept as the best solution found up to this point.

6. Experimental results

6.1. Instances and computational environment

To test the proposed methodologies, we use 120 instances suggested by Taillard (1991). We warm-started all studied algorithms using the solutions from the beam-search procedure proposed by Fernandez-Viagas and Framinan (2017), which are the state-of-the-art constructive heuristic for PFSP. We used the `BSCH(n)` version which stretchs the search over "n" jobs. Since in this paper, all studied algorithms use solutions from `BSCH(n)` as incumbents, we disregard the `BSCH(n)` running time.

The computational experiments were performed in a cluster of identical machines with a Intel Xeon E5530 CPU at 2.40GHz and 64 GB of RAM running CentOS Linux. Heuristics proposed in this paper were implemented in Julia language. For `BSCH(n)`, we

used the original version provided by Fernandez-Viagas and Framinan (2017), implemented in C#. Model (1a)–(1g) was solved with IBM ILOG CPLEX 12.8 solver. To have a fair comparison among the algorithms, all implementations use a single thread. We performed experiments with stopping criterion of 5, 15, 30, and 60 min with 30 independent experiments for each instance for these algorithms.

We run CPLEX with two different setups. In the first, CPLEX uses a single thread and stops either when it finds an optimal integer solution, or it reaches the maximum time as defined for the heuristics (`CPLEX`). This setup is meant to achieve direct comparison with the other methods proposed in this paper. In the second setup, we run CPLEX for 24 wall-clock hours using 12 threads (`CPLEX-24h`). Such configuration looks to find optimal solutions or, at least, compute the best possible lower bounds. Since this configuration uses far more time and computer power per run than the other configurations, we use the results only for reporting.

6.2. Parameter settings

We performed a full-factorial design of experiments for `ILS`and `IGS`, because the number of control parameters is small. We varied the number of swap moves (for `ILS`) and the intensity of the destruction (for `IGS`) between 2 and 20. The experiments were carried out on 24 instances, each two chosen from an instance group. The experiments suggested that seven swaps moves for `ILS`and seven jobs in the destruction phase for `IGS`, enable both algorithms to achieve the best results.

Since BRKGA algorithms require a large number of parameters, we tuned such settings employing the iterated F-race and the `irace` package (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016). `irace` suggested: population size of $p = n \times 9$, where n is the number of jobs; elite size $p_e = 0.30 \times p$; number of mutants introduced at each generation $p_m = 0.22 \times p$; probability of inheriting each allele from elite parent $\rho = 0.55$. For variations with local search, the suggested number of local search iterations is 10. For variants without shaking, `irace` suggested a full population reset after 1000 iterations without improvements in the best solution. For variants where we apply shaking, `irace` suggest $R = 1000$ and $\lambda = 10$. We made $R^{\star}$ and $R^{\star\star}$ proportional to R, and `irace` suggested $R^{\star} = 5 \times R$ and $R^{\star\star} = 10 \times R$. For details in the parameters and configurations, refer to Appendix A.

6.3. Result analysis

We performed the comparison among the proposed algorithms regarding solution quality and computational effort. For solution quality, we compute the classical Relative Percentage Deviation (RPD) and associated averages. Let $\mathcal{I}$ be a set of instances. Let $\mathcal{A}$ be the set of algorithms, and assume that set $\mathcal{R}_A$ enumerates the independent runs for algorithm $A \in \mathcal{A}$ (as defined in Section 6.1, 30 runs for the heuristics and one run for CPLEX).[1] We defined C^A_{ir} as the total flowtime obtained by algorithm A in instance i on run r, and C^{best}_i as the best total flowtime found across all algorithms for instance i. The Relative Percentage Deviation (RPD) from the best solution of instance i is defined as

$$RPD^A_{ir} = \frac{C^A_{ir} - C^{best}_i}{C^{best}_i}, \quad \forall A \in \mathcal{A},\ i \in \mathcal{I},\ \text{and } r \in \mathcal{R}_A. \tag{2}$$

[1] Note that this is necessary because randomized algorithms generate a distribution of solutions (and convergence times) from runs with different (starting) random number generator states, and one must account for all these results for a proper statistical analysis.

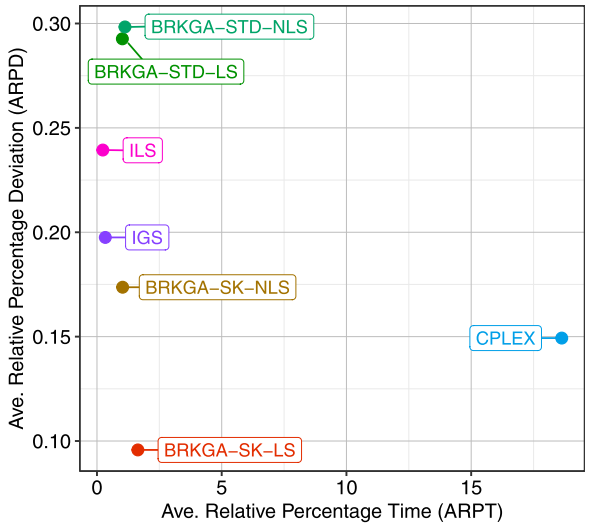

Fig. 3. Pareto frontier using the average relative percentage deviations and times.

The general Average Relative Percentage Deviation (ARPD) for algorithm A is defined as

$$ARPD_A = \frac{100}{|\mathcal{I}| \cdot |\mathcal{R}_A|} \sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R}_A} RPD^A_{ir}. \tag{3}$$

Note that we can obtain different averages by subsetting set $\mathcal{I}$, for example, per instance or groups of instances.

To measure the computational effort, we use the Average Relative Percentage Computational Time (ARPT) proposed on Fernandez-Viagas and Framinan (2015) for deterministic heuristics. However, we have slightly adapted that definition to accommodate randomized algorithms with multiple runs. First, instead of account the total running time, we use the time to best solution found in a given run, which we define as T^A_{ir}. In other words, T^A_{ir} is the time that algorithm A needs to converge on run r (find the best solution in that run) for instance i. The reason for this approach is that all experiments are carried out until the maximum time allowed, even though the algorithm may converge earlier to the best solution found in that run. We also use the results of all independent runs for all algorithms, as it is done for ARPD. With these considerations, ARPT is defined for each algorithm A as

$$ARPT_A = 1 + \frac{1}{|\mathcal{I}| \cdot |\mathcal{R}_A|} \sum_{i \in \mathcal{I}} \sum_{r \in \mathcal{R}_A} \frac{T^A_{ir} - ACT_i}{ACT_i} \tag{4}$$

where

$$ACT_i = \frac{1}{|\mathcal{A}|} \sum_{A \in \mathcal{A}} \sum_{r \in \mathcal{R}_A} \frac{T^A_{ir}}{|\mathcal{R}_A|}, \quad \forall i \in \mathcal{I}. \tag{5}$$

ACT_i is the average of T^A_{ir} for instance i considering all algorithms. Note that in Eq. (4) starts adding one to the average which guarantees that ARPT is always a positive number, allowing different transformations (such as logarithmic ones) when necessary. Therefore, while ARPD is the average deviation from the value of the best solutions, ARPT is the average deviation from the average computational time.

In terms of notations, for standard BRKGA variations (with full population reset), we used the suffix `STD`. For variations using shaking, the suffix `SK` was used. Similarly, variations using local search and variants not using local search, have the prefixes `LS` and `NLS`, respectively. For other algorithms, we used their usual acronyms.

Fig. 3 depicts the Pareto frontier among the algorithms regarding their ARPD and ARPT. Both metrics were computed considering

Table 1
Average relative percentage deviations (ARPD) and times (ARPT) per instance group. The last line shows the general ARPD and ARPT for all instances and runs.

Instance Group	BRKGA-SK-LS		BRKGA-SK-NLS		BRKGA-STD-LS		BRKGA-STD-NLS		CPLEX		IGS		ILS	
	ARPD	ARPT	ARPD	ARPT	ARPD	ARPT	ARPD	ARPT	ARPD	ARPT	ARPD	ARPT	ARPD	ARPT
20 × 5	0.00	0.06	0.00	0.13	0.02	1.51	0.02	1.63	0.00	78.57	0.00	0.01	0.00	0.01
20 × 10	0.00	0.04	0.00	0.20	0.09	2.04	0.10	2.18	0.00	45.80	0.00	0.01	0.00	0.01
20 × 20	0.00	0.03	0.00	0.14	0.09	2.62	0.12	2.27	0.00	27.94	0.00	0.01	0.00	0.01
50 × 5	0.11	1.16	0.24	1.62	0.46	0.92	0.46	0.97	0.14	4.44	0.37	0.66	0.39	0.54
50 × 10	0.28	1.31	0.52	1.60	0.86	0.24	0.86	0.29	0.58	3.78	0.61	1.47	0.77	0.99
50 × 20	0.30	1.19	0.49	1.37	0.97	0.68	0.99	0.61	0.43	3.39	0.49	1.14	0.71	0.92
100 × 5	0.06	2.92	0.17	1.13	0.19	0.45	0.20	0.49	0.07	24.35	0.17	0.08	0.19	0.08
100 × 10	0.12	2.83	0.22	1.38	0.26	0.57	0.27	0.66	0.15	9.07	0.25	0.20	0.28	0.01
100 × 20	0.21	2.43	0.35	1.75	0.44	0.58	0.45	0.73	0.37	4.30	0.39	0.30	0.47	0.03
200 × 10	0.02	2.43	0.04	1.03	0.04	0.95	0.05	1.32	0.02	7.10	0.04	0.00	0.04	0.00
200 × 20	0.03	2.60	0.05	0.81	0.05	0.90	0.06	1.37	0.03	9.97	0.05	0.01	0.05	0.01
500 × 20	0.01	2.67	0.01	1.17	0.01	0.76	0.01	0.94	0.01	4.85	0.01	0.12	0.01	0.19
General	0.10	1.64	0.17	1.03	0.29	1.02	0.30	1.12	0.15	18.63	0.20	0.33	0.24	0.23

Table 2
The average RDP (ARDP) and standard deviation for each algorithm given a maximum running time. The best results are in bold.

Algorithm	5 min.	15 min.	30 min.	60 min.
BRKGA-SK-LS	0.09 ± 0.15	**0.09 ± 0.14**	**0.08 ± 0.13**	**0.10 ± 0.15**
BRKGA-SK-NLS	0.16 ± 0.23	0.16 ± 0.23	0.16 ± 0.23	0.17 ± 0.24
BRKGA-STD-LS	0.26 ± 0.33	0.27 ± 0.35	0.27 ± 0.35	0.29 ± 0.38
BRKGA-STD-NLS	0.26 ± 0.33	0.27 ± 0.35	0.27 ± 0.36	0.30 ± 0.39
CPLEX	**0.07 ± 0.13**	0.10 ± 0.18	0.12 ± 0.20	0.15 ± 0.23
IGS	0.17 ± 0.23	0.18 ± 0.24	0.18 ± 0.24	0.20 ± 0.25
ILS	0.19 ± 0.28	0.21 ± 0.30	0.22 ± 0.30	0.24 ± 0.32

either the best-known solution in the literature or the best solution found by the algorithms for each instance. The stop criterion was 60 minutes. Note that as small the ARPD is, better is the quality of the solutions produced by the algorithm; and as small the ARPT is, faster the algorithm is to produce the solutions. Hence, the best algorithms according to ARPD and ARPT should be positioned on the lower-left of the frontier. First, note that the standard versions of BRKGA presented large ARPD when compared to the other algorithms. For the remaining heuristics, one can observe a downwards tendency in the deviation and an increase on the ARPT. ILS starts this tendency with the best ARPT but the worst ARPD. IGS presented a good ARPD improvement concerning ILS with tiny ARPT increment. BRKGA-SK-NLS follows the tendency, improving the ARPD at computational time cost. BRKGA-SK-LS show a considerable quality improvement setting apart from all other algorithms. However, its ARPT is the largest among the heuristics. CPLEX presented the second best ARDP although it uses much more time than the heuristics, as expected. Table 1 shows the ARPD and ARPT for each group of instances. Note that the last line brings the general average results depicted in the Pareto frontier.

To better assess the differences between the algorithms, we performed experiments using different stopping times. However, the parameters used for these experiments have the same values of those tuned for 60-minute runs. Fig. 4 depicts the boxplots for the RPD and Table 2 brings the RPD average and standard deviation of each algorithm and scenario. It is worth mentioning that, to compute the RPD, we use the best solution found in each scenario not the overall best-known solution for each instance. The rationale is to assess the behavior of the algorithms for unknown instances for which the bounds are also unknown. Note that BRKGA-SK-LS presents the smallest deviations consistently, although CPLEX appears to present the best results for the 5 minutes scenario. Indeed, CPLEX shows the second best results for each other scenario followed by BRKGA-SK-NLS. In the sequence, IGS depicts better results than ILS. The standard BRKGA variations presented more variability of results, reaching the worst averages.

Since the results from each algorithm are very close to each other, we applied pairwise tests using Wilcoxon rank sum test with Hommel's p-value adjusting. Considering a confidence interval of 95%, Table 3 shows the p-values for each pair of algorithms and stopping times. This table presents some interesting results. The first thing to note is the results presented by CPLEX. In the 5-min scenario, CPLEX produced significantly better solutions than the other algorithms, but BRKGA-SK-LS. Indeed, we cannot confirm the statistical difference between CPLEX and BRKGA-SK-LS for 5-min and 15-min scenarios (p-value > 0.05). However, as we allow more running time, CPLEX results diminish returns, presenting significant worse results than BRKGA-SK-LS, but significantly better results than BRKGA standard versions, ILS, and IGS. We cannot confirm the difference between CPLEX and BRKGA-SK-NLS for 15, 30, and 60-min scenarios (p-values $>$ 0.10).

These results show that CPLEX quickly found good solutions from BSCH(n) incumbent, but it got stuck in local minima. Allowing more running time enables the heuristics scape from these local minima and improve the results in the long run. Despite CPLEX results, BRKGA-SK-LS produced significantly better solutions than the other algorithms in all scenarios. The solutions of BRKGA-SK-NLS for 30 and 60-min scenarios are statistically better than those of ILSand IGS. Also, note that we cannot confirm the difference between ILSand IGSfor any scenario (p-values $>$ 0.41). BRKGA-STD-LS and BRKGA-STD-NLS presented the worse results among the algorithms.

Fig. 5 shows performance profiles for all algorithms where X axis shows the time needed to reach a target solution value while the Y axis shows the cumulative probability to reach a target solution value for the given time in the X axis. In Fig. 5a, the target values are the median solution values for each instance. For Fig. 5b, the values of the best-known solutions for each instance are used. Note that, in both average and best solution cases, IGS (purple line with inverted triangles) presented a higher probability of finding a good solution with little running time. However,

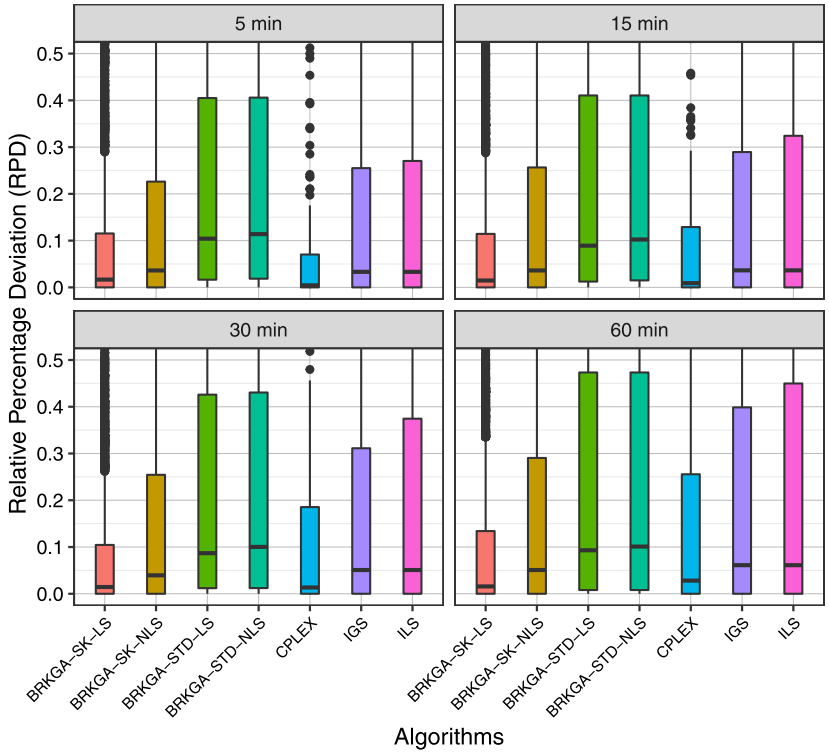

Fig. 4. Distribution of RPDs for each algorithm considering different stopping criteria.

Table 3
The *p*-values from pairwise Wilcoxon rank sum tests adjusted by Hommel's procedure. The tests use the RPD distributions of each algorithm, considering different stopping times, at 95% interval of confidence. The *p*-values in bold indicate situations where we cannot confirm the statistical difference between the algorithms. A minus sign (-) indicates that Algorithm A found solutions that are significant better than solutions found by Algorithm B. A plus sign (+) indicates the opposite case.

Pair	Stopping time			
Algorithm A / Algorithm B	5 min.	15 min.	30 min.	60 min.
BRKGA-SK-LS / BRKGA-SK-NLS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-LS / BRKGA-STD-LS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-LS / BRKGA-STD-NLS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-LS / CPLEX	**0.410**	**0.560**	0.021 −	0.001 −
BRKGA-SK-LS / IGS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-LS / ILS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-NLS / BRKGA-STD-LS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-NLS / BRKGA-STD-NLS	0.000 −	0.000 −	0.000 −	0.000 −
BRKGA-SK-NLS / CPLEX	0.000 +	**0.102**	**0.541**	**1.000**
BRKGA-SK-NLS / IGS	**1.000**	**0.356**	0.039 −	0.021 −
BRKGA-SK-NLS / ILS	**1.000**	0.001 −	0.000 −	0.000 −
BRKGA-STD-LS / BRKGA-STD-NLS	**1.000**	**1.000**	**1.000**	**1.000**
BRKGA-STD-LS / CPLEX	0.000 +	0.000 +	0.000 +	0.000 +
BRKGA-STD-LS / IGS	0.000 +	0.000 +	0.000 +	0.000 +
BRKGA-STD-LS / ILS	0.000 +	0.000 +	0.000 +	0.000 +
BRKGA-STD-NLS / CPLEX	0.000 +	0.000 +	0.000 +	0.000 +
BRKGA-STD-NLS / IGS	0.000 +	0.000 +	0.000 +	0.000 +
BRKGA-STD-NLS / ILS	0.000 +	0.000 +	0.000 +	0.000 +
CPLEX / IGS	0.000 −	0.020 −	**0.110**	**0.760**
CPLEX / ILS	0.000 −	0.007 −	0.039 −	**0.457**
IGS / ILS	**1.000**	**0.541**	**0.410**	**0.526**

both BRKGA-SK-LS (red line with circles) and BRKGA-SK-NLS (golden line with upside triangles) found better solutions in the long run in both average and best solution cases. BRKGA-STD-LS (dark green line with plus signal) and BRKGA-STD-NLS (light green line with crosses) have better probability than CPLEX (blue line with diamonds) in the average case, but they fall short for the best solutions target. ILS presented a good cumulative probability in the first minutes of optimization but then falls short for long runs. It is interesting to note CPLEX behavior. Contrasting the relatively small ARPD depicted in Fig. 4 and Table 2 with the cumulative probability distribution shown in Fig. 5b, one can argue that CPLEX finds solutions close to the best-known solutions, but it fails to reach them.

Note that IGS and ILS spent most of their time in the local searches. Considering only the best results for each algorithm, IGS iterated 23,578 ± 63,566 times on average, while ILS iter-

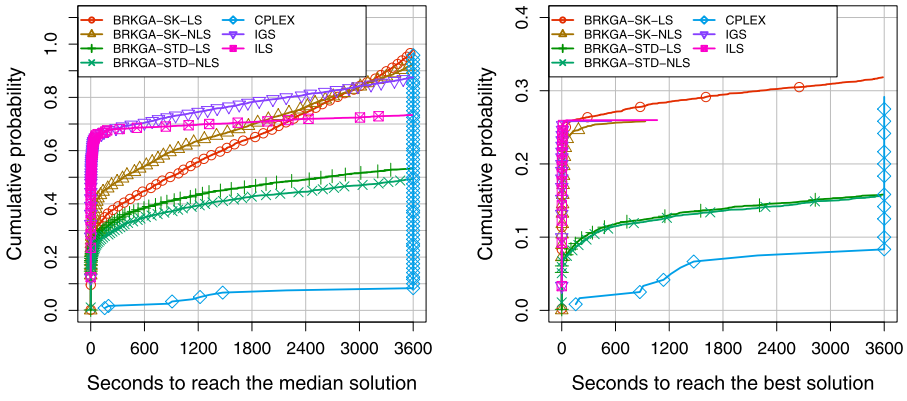

Fig. 5. Running time distributions to median and best-known solutions found. The identification marks correspond to 2% of the points plotted for each algorithm.

Table 4
Algorithm performance on instances with known and unknown optimum solutions.

Algorithm	Known optima (20 instances)					Unknown optima (100 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
BRKGA-SK-LS	20	100.00	100.00	–	–	96	96.00	18.32	0.14	0.16
BRKGA-SK-NLS	20	100.00	100.00	–	–	12	12.00	11.03	0.23	0.25
BRKGA-STD-LS	20	100.00	64.00	0.15	0.14	13	13.00	6.23	0.36	0.40
BRKGA-STD-NLS	18	90.00	64.83	0.17	0.14	11	11.00	5.77	0.37	0.40
CPLEX	20	100.00	100.00	–	–	15	15.00	15.00	0.21	0.25
IGS	20	100.00	100.00	–	–	13	13.00	11.10	0.27	0.25
ILS	20	100.00	100.00	–	–	12	12.00	11.12	0.30	0.31

ated 8,914 ± 41,221. These numbers indicate that both IGS and ILS used their diversification mechanism only a few times, leading to a premature convergence. While this behavior helps to find good and best solutions for small instances, it makes IGS and ILS to fall short in solving large and more interesting instances. BRKGA variants were able to evolve much more iterations when compare to the IGS. BRKGA-SK-LS and BRKGA-SK-NLS iterates, on average, 192,016 ± 365,730 and 362,696 ± 636,657, respectively. The standard variants went even further: BRKGA-STD-LS with 658,133 ± 1,959,411 iterations, and BRKGA-STD-NLS with 729,059 ± 2,052,841 iterations, on average. Therefore, although BRKGA variations used more time to converge, they potentially explored a much larger portion of the solution landscape, which, consequently, led to better solutions.

Table 4 summarizes the results regarding the number of good solutions found, and it is partitioned into two sections. The first group of columns (2–6) shows the performance considering 20 instances (17% of) for which solution was proven optimal (instances TA1 to TA20). 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). Note that both BRKGA-SK versions, IGSand ILSwere able to find an optimal solution in all runs, which is really remarkable for a heuristic. The STD versions missed few optimal solutions, but the proportional deviations were minimal.

The second segment of the table shows the results for instances with unknown optimum solutions. The description is similar to the first segment, but the results are compared to the best solution found by the algorithms. One can note that BRKGA-SK-LS found the best solution in almost all instances, with a clear advantage margin from the other approaches. However, note that although the other algorithms did not reach so many best solutions, the average proportional difference is less than 0.34%. BRKGA-SK-LS found a good solution in 29% of the runs. However, for the majority of the instances, it did find the best solution for at least one run. The similar behavior is also observed for the other approaches.

We improved the best-known upper bounds (Fernandez-Viagas & Framinan, 2017) for 36 of 120 instances (30%), mostly concentrated in the large and difficult ones. The average gap for the non-optimal instances was 6.54 ± 3.77% (average ± standard deviation) with minimum of 1.43, and maximum of 13.59. Depending on the application, such numbers represent an excellent compromise between solution cost and optimization time. It is interesting to notice that for 25 of those 36 instances (69%), at least one of the BRKGA variants obtained results better than IGS. For the other 11 instances (30%), both algorithms obtained the same results. Concerning the solution values, CPLEX-60min and CPLEX-24h obtained the same solution values, indicating that the extra time and computer power did not help in improving solutions. This fact may also indicate that the current solutions are very close to the optimal. However, CPLEX-24h did improve the lower bounds in all cases. On average, CPLEX-60min presented an integrality gap of 15.53 ± 28.50%, when CPLEX-24h showed an average gap of 5.24 ± 3.93%. The detailed results for all algorithms, per instance, is described on Appendix B.

Table 5 shows the average percentage difference between the incumbent solution value given by BSCH(n) and the value of the final solution. The average was computed over all 30 independent runs for each algorithm (one run per instance for CPLEX) using 60 minutes, and they are categorized by instance group shown in the first column. We highlighted the best results in bold. The last two

Table 5
Average percentage improvement concerning the warm-start solution from BSCH(n). The last line shows the percentage of runs where no improved solution was found.

Instance group	BRKGA				CPLEX	IGS	ILS
	SK-LS	SK-NLS	STD-LS	STD-NLS			
20 × 5	**−0.75**	**−0.75**	−0.73	−0.73	**−0.75**	**−0.75**	**−0.75**
20 × 10	**−0.68**	**−0.68**	−0.59	−0.59	**−0.68**	**−0.68**	**−0.68**
20 × 20	**−0.69**	**−0.69**	−0.60	−0.57	**−0.69**	**−0.69**	**−0.69**
50 × 5	**−0.56**	−0.44	−0.22	−0.22	−0.54	−0.31	−0.29
50 × 10	**−0.62**	−0.38	−0.05	−0.05	−0.33	−0.29	−0.14
50 × 20	**−1.05**	−0.86	−0.39	−0.37	−0.92	−0.85	−0.64
100 × 5	**−0.19**	−0.09	−0.06	−0.05	−0.18	−0.09	−0.07
100 × 10	**−0.21**	−0.10	−0.06	−0.06	−0.17	−0.08	−0.04
100 × 20	**−0.41**	−0.26	−0.17	−0.17	−0.24	−0.22	−0.15
200 × 10	**−0.06**	−0.04	−0.03	−0.03	−0.05	−0.03	−0.03
200 × 20	**−0.07**	−0.05	−0.05	−0.04	**−0.07**	−0.05	−0.05
500 × 20	**−0.01**	**−0.01**	**−0.01**	**−0.01**	**−0.01**	**−0.01**	**−0.01**
Average	−0.44 ± 0.53	−0.36 ± 0.50	−0.25 ± 0.43	−0.24 ± 0.42	−0.39 ± 0.51	−0.34 ± 0.50	−0.30 ± 0.48
No Impr.%	5.39	9.17	23.70	23.80	0.19	14.00	15.30

lines show the average across all instances ("Average"), and the percentage of the total runs where the algorithms could not improve the incumbent solution ("No Impr. %"). In general, the algorithms can improve the initial solution from BSCH(n). The maximum improvement was found by BRKGA-SK-LS and is about 3%, followed by IGS with 2.64%. In a few cases, the algorithms got stuck with the initial solutions. In BRKGA-SK variations, such situation happens in less than 10% of the runs, when for standard variations the no-improving rate is about 23%. IGS and ILS could not improve BSCH(n) solutions in 14% and 15.30% of the cases, respectively.

7. Conclusions

In this paper, we addressed the Permutation Flowshop Scheduling Problem (PFSP) where one wants to schedule jobs aiming to minimize the total flow time. PFSP is a fundamental problem with applications in several industries and public activities, and almost all other fields where a sequence of tasks must be executed. We presented four Biased Random-Key Genetic Algorithms (BRKGA), and an Iterated Greedy Search (IGS) procedure to solve the PFSP. We compared these approaches with the Iterated Local Search procedure (ILS), considered to be the state-of-the-art algorithm for the PFSP.

We also applied a commercial mixed integer programming solver proving the optimality for a subset of instances, and providing the lower bounds for the others. Another important contribution of this paper is the introduction of a new technique for BRKGAs called *shaking*, where instead of reseting the population entirely, we "shake" the elite population and reset all non-elite individuals.

It is challenging to find a scalable solution for PFSP. All BRKGA variations were able to explore the solution landscape very well and found a good deal of best solutions. Variants using the shaking strategy overcame the other algorithms, presenting strong results. Therefore, instead of reseting the population entirely as the standard BRKGA does, the population shaking strategy was able to introduce diversification changing the elite set without losing the optimization track completely. ILS and IGS presented good results for small instances, but they fell short in the large ones, mainly because of the little exploration these strategies used. Therefore, BRKGA algorithms pose as viable methods and strong contenders in solving permutation flowshop problems at large scale. For future works, we would like to apply the BRKGA with shaking strategy to other scheduling problems, to assess its efficacy for those scenarios.

Credit authorship contribution statement

Carlos E. Andrade: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Visualization, Supervision. **Thuener Silva:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing. **Luciana S. Pessoa:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing, Supervision.

Acknowledgements

We thank the anonymous reviewers for providing comments that improved this paper, Victor Fernandez-Viagas and Jose M. Framinan for had kindly providing the original code of the Beam-Search Constructive Heuristic, and Rakesh K. Sinha for comments in the reviewed version of this manusscript. This study was financed in part by PUC–Rio, Escola Nacional de Seguros and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) – Finance Code 001.

Appendix A. Parameter settings

BRKGA has a larger number of control parameters which makes difficult to perform a full factorial design. Therefore, we used the *iterated racing* (Birattari, Yuan, Balaprakash, & Stützle, 2010) to tune the parameters. This method consists in sampling configurations from a particular distribution, evaluating them using a statistical, and refining the sampling distribution with repeated applications of F-Race. We used the `irace` package (López-Ibáñez et al., 2016), implemented in R, for such task.

We used a budget of 3000 experiments in the tuning procedure over the 30 instances selected randomly. Each experiment was limited to one hour. To tune the BRKGA parameters, we used the following ranges: population size multiplier $px \in [1, 10]$; percentage of elite individuals $p_e \in [10, 50]$; percentage of mutants introduced at each generation $p_m \in [10, 50]$; probability on biased crossover $\rho \in [0.5, 0.8]$; and number of iterations without improvement in the best solution until population reset/shake (R) in [200, 2000]. The intensity of the shaking (λ) and the local search iterations (LS) are in [1, 50]. Both $R^{\star}$ and $R^{\star\star}$ multipliers are in [2, 15]. Table A.1 shows the suggestion made by `irace`. We picked the values of the first line of each table, rounded up to two digits for real values, and round to the next multiple of 10, in the case of integer values.

Table A.1
`irace` results for BRKGA variants.

px	% p_e	% p_m	ρ	LS	λ	R	$mR^{\star}$	$mR^{\star\star}$
9	0.30	0.23	0.55	10	10	1000	10	5
10	0.26	0.31	0.79	12	9	972	12	7
7	0.18	0.29	0.72	9	6	1208	11	4
10	0.16	0.36	0.67	28	9	1309	6	5
8	0.20	0.30	0.74	12	4	1210	7	6

Appendix B. Detailed results

Table B.2 depicts the full set of solutions compared to the best results known. The first column shows the name of the instance. In bold, we highlighted those instances that the approaches discussed in this paper improved from the results reported in Fernandez-Viagas and Framinan (2017). The second column depicts the best lower bounds found by `CPLEX-24h`, followed by the best-known solution and the percentual gap between those solutions and the lower bounds, on the third and fourth columns, respectively. A dash (–) in "Lower bound" and "Gap%" indicates that the optimal solution was found. The following columns describe the percentage deviation of the best solution found by the algorithm compared to the best solution known, where `BSCH(n)` are the deviations from the results reported on Fernandez-Viagas and Framinan (2017). A star ($\star$) indicates that the approach reached the solution. This table is also available in the CSV format in the supplementary materials for download.

Table B.1
Best results for each instance, and the percentage deviation of the best solution found for each algorithm.

Instance	Lower bound	Best solution	Gap%	BRKGA				BSCH(n)	CPLEX1h	CPLEX24h	IGS	ILS
				SK-LS	SK-NLS	STD-LS	STD-NLS					
TA1	–	14,033	–	⋆	⋆	⋆	⋆	0.18	⋆	⋆	⋆	⋆
TA2	–	15,151	–	⋆	⋆	⋆	⋆	1.76	⋆	⋆	⋆	⋆
TA3	–	13,301	–	⋆	⋆	⋆	⋆	2.71	⋆	⋆	⋆	⋆
TA4	–	15,447	–	⋆	⋆	⋆	⋆	0.84	⋆	⋆	⋆	⋆
TA5	–	13,529	–	⋆	⋆	⋆	⋆	0.72	⋆	⋆	⋆	⋆
TA6	–	13,123	–	⋆	⋆	⋆	⋆	0.21	⋆	⋆	⋆	⋆
TA7	–	13,548	–	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA8	–	13,948	–	⋆	⋆	⋆	⋆	0.62	⋆	⋆	⋆	⋆
TA9	–	14,295	–	⋆	⋆	⋆	⋆	0.38	⋆	⋆	⋆	⋆
TA10	–	12,943	–	⋆	⋆	⋆	⋆	0.22	⋆	⋆	⋆	⋆
TA11	–	20,911	–	⋆	⋆	⋆	⋆	0.22	⋆	⋆	⋆	⋆
TA12	–	22,440	–	⋆	⋆	⋆	⋆	0.61	⋆	⋆	⋆	⋆
TA13	–	19,833	–	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA14	–	18,710	–	⋆	⋆	⋆	0.07	0.44	⋆	⋆	⋆	⋆
TA15	–	18,641	–	⋆	⋆	⋆	⋆	0.52	⋆	⋆	⋆	⋆
TA16	–	19,245	–	⋆	⋆	⋆	0.02	1.81	⋆	⋆	⋆	⋆
TA17	–	18,363	–	⋆	⋆	⋆	⋆	0.93	⋆	⋆	⋆	⋆
TA18	–	20,241	–	⋆	⋆	⋆	⋆	1.82	⋆	⋆	⋆	⋆
TA19	–	20,330	–	⋆	⋆	⋆	⋆	0.48	⋆	⋆	⋆	⋆
TA20	–	21,320	–	⋆	⋆	⋆	⋆	0.08	⋆	⋆	⋆	⋆
TA21	32409.65	33,623	3.74	⋆	⋆	⋆	⋆	1.81	⋆	⋆	⋆	⋆
TA22	30439.51	31,587	3.77	⋆	⋆	⋆	⋆	0.33	⋆	⋆	⋆	⋆
TA23	32695.65	33,920	3.74	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA24	30540.21	31,661	3.67	⋆	⋆	⋆	⋆	0.24	⋆	⋆	⋆	⋆
TA25	32684.98	34,557	5.73	⋆	⋆	⋆	⋆	0.97	⋆	⋆	⋆	⋆
TA26	32040.23	32,564	1.63	⋆	⋆	⋆	⋆	0.12	⋆	⋆	⋆	⋆
TA27	32221.97	32,922	2.17	⋆	⋆	⋆	⋆	0.99	⋆	⋆	⋆	⋆
TA28	31538.05	32,412	2.77	⋆	⋆	⋆	⋆	0.94	⋆	⋆	⋆	⋆
TA29	31926.00	33,600	5.24	⋆	⋆	⋆	⋆	0.76	⋆	⋆	⋆	⋆
TA30	31477.20	32,262	2.49	⋆	⋆	⋆	⋆	0.83	⋆	⋆	⋆	⋆
TA31	63425.08	64,802	2.17	⋆	0.11	0.41	0.45	0.51	0.06	0.06	0.11	0.17
TA32	66309.69	68,051	2.63	⋆	0.12	0.44	0.47	1.43	0.08	0.08	0.20	0.32
TA33	61567.20	63,162	2.59	⋆	0.23	0.44	0.44	0.44	0.29	0.29	0.44	0.34
TA34	66458.52	68,226	2.66	⋆	0.29	0.54	0.47	0.59	0.30	0.30	0.22	0.47
TA35	67629.56	69,351	2.55	⋆	0.11	0.25	0.16	0.75	0.13	0.13	0.30	0.22
TA36	65901.21	66,841	1.43	⋆	0.22	0.25	0.25	0.34	0.19	0.19	0.18	0.28
TA37	64867.45	66,253	2.14	⋆	0.04	0.57	0.66	1.15	0.01	0.01	0.12	0.19
TA38	62908.35	64,332	2.26	0.04	0.06	0.07	0.07	0.49	0.04	0.04	0.12	0.25
TA39	61482.92	62,981	2.44	⋆	0.26	0.48	0.48	0.64	0.26	0.26	0.25	0.27
TA40	67123.46	68,770	2.45	0.09	0.21	0.34	0.35	0.63	0.18	0.18	0.49	0.52
TA41	81457.49	87,114	6.94	0.09	0.58	0.99	0.99	1.09	0.87	0.87	0.53	0.61
TA42	77769.23	82,820	6.49	⋆	0.16	1.01	1.01	1.01	0.91	0.91	0.50	0.39
TA43	75553.61	79,931	5.79	0.17	0.38	0.90	0.90	0.90	0.89	0.89	0.53	0.50
TA44	80360.36	86,446	7.57	0.15	0.42	0.73	0.73	0.73	0.72	0.72	0.46	0.58
TA45	80971.24	86,377	6.68	0.01	0.04	0.99	0.99	0.99	0.82	0.82	0.40	0.56
TA46	80746.51	86,587	7.23	0.06	0.16	0.67	0.67	0.68	0.34	0.34	0.34	0.46
TA47	83445.20	88,750	6.36	⋆	0.09	0.87	0.87	1.12	0.54	0.54	0.49	0.72
TA48	81293.29	86,727	6.68	0.09	0.24	1.06	1.39	1.55	0.72	0.72	0.58	0.82
TA49	79599.33	85,441	7.34	0.11	0.41	0.47	0.47	0.74	0.30	0.30	0.57	0.52
TA50	82597.34	87,998	6.54	⋆	0.34	0.91	0.91	0.95	0.34	0.34	0.40	0.59
TA51	110779.46	125,831	13.59	⋆	0.02	0.27	0.27	0.27	0.26	0.26	0.27	0.27
TA52	108234.50	119,247	10.17	⋆	0.15	0.52	0.52	0.57	0.44	0.44	0.08	0.25
TA53	105693.50	116,459	10.19	⋆	0.32	0.90	1.20	1.60	0.19	0.19	0.26	0.65
TA54	108682.29	120,261	10.65	0.41	0.58	0.97	1.10	2.38	1.06	1.06	0.66	0.83

(*continued on next page*)

Table B.1 (*continued*)

Instance	Lower bound	Best solution	Gap%	BRKGA				BSCH(n)	CPLEX1h	CPLEX24h	IGS	ILS
				SK-LS	SK-NLS	STD-LS	STD-NLS					
TA55	104864.01	118,184	12.70	⋆	0.38	0.86	1.03	2.22	0.62	0.62	0.50	0.67
TA56	106614.11	120,586	13.11	0.11	0.31	1.06	1.06	1.07	0.43	0.43	0.42	0.46
TA57	109049.19	122,880	12.68	0.17	0.31	0.83	0.83	1.63	0.54	0.54	0.45	0.74
TA58	108754.38	122,489	12.63	0.03	0.27	1.31	1.53	2.42	0.67	0.67	0.29	0.60
TA59	107458.13	121,872	13.41	0.11	0.23	0.54	0.57	1.26	0.42	0.42	0.36	0.54
TA60	110327.42	123,954	12.35	0.17	0.37	0.97	1.06	1.24	0.71	0.71	0.49	0.59
TA61	248085.57	253,167	2.05	⋆	⋆	⋆	⋆	0.03	⋆	⋆	0.01	0.03
TA62	237414.97	241,925	1.90	⋆	0.07	0.07	0.07	0.07	0.05	0.05	0.07	0.07
TA63	233236.38	237,832	1.97	0.17	0.38	0.40	0.40	0.41	0.32	0.32	0.39	0.40
TA64	223001.21	227,522	2.03	⋆	0.22	0.28	0.28	0.30	0.07	0.07	0.25	0.29
TA65	235251.37	240,301	2.15	0.18	0.19	0.19	0.19	0.24	0.19	0.19	0.19	0.22
TA66	227625.84	232,342	2.07	0.08	0.16	0.16	0.20	0.35	0.21	0.21	0.29	0.29
TA67	235913.31	240,366	1.89	0.18	0.43	0.96	1.06	1.57	0.28	0.28	0.62	0.85
TA68	225998.50	230,945	2.19	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
TA69	242577.86	247,526	2.04	⋆	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
TA70	238147.21	242,933	2.01	0.10	0.26	0.26	0.26	0.26	0.22	0.22	0.26	0.24
TA71	283005.20	298,385	5.43	0.41	0.77	1.21	1.23	1.36	1.09	1.09	0.87	1.14
TA72	259208.82	273,674	5.58	⋆	0.05	0.05	0.05	0.06	⋆	⋆	0.05	0.05
TA73	271813.41	288,114	6.00	0.19	0.31	0.34	0.34	0.36	0.31	0.31	0.34	0.34
TA74	283451.60	301,044	6.21	0.30	0.34	0.36	0.36	0.36	0.34	0.34	0.36	0.36
TA75	267821.37	284,233	6.13	⋆	0.02	0.02	0.02	0.02	⋆	⋆	0.02	0.02
TA76	253800.63	269,686	6.26	0.58	0.66	0.80	0.80	0.81	0.66	0.66	0.78	0.81
TA77	264572.14	279,463	5.63	0.35	0.79	0.70	0.90	1.65	0.68	0.68	0.92	1.08
TA78	273381.62	290,908	6.41	0.15	0.22	0.24	0.24	0.24	0.22	0.22	0.24	0.24
TA79	286021.91	301,970	5.58	0.31	0.43	0.43	0.45	0.52	0.42	0.42	0.45	0.45
TA80	276762.89	291,283	5.25	0.16	0.19	0.34	0.34	0.35	0.21	0.21	0.29	0.34
TA81	322937.70	365,463	13.17	0.83	1.15	1.28	1.30	1.36	1.23	1.23	1.17	1.30
TA82	331942.97	372,449	12.20	0.16	0.29	0.31	0.31	0.31	0.29	0.29	0.31	0.31
TA83	329667.79	370,027	12.24	0.63	0.79	1.08	1.09	1.20	0.96	0.96	1.05	1.12
TA84	332064.91	372,393	12.14	0.66	0.98	1.12	1.08	1.19	1.11	1.11	0.97	1.12
TA85	328577.91	368,915	12.28	0.36	0.39	0.44	0.44	0.44	0.43	0.43	0.41	0.44
TA86	327315.43	370,908	13.32	0.59	0.86	0.95	0.96	1.25	0.96	0.96	0.98	1.04
TA87	331099.33	373,408	12.78	0.42	0.57	0.68	0.68	0.68	0.68	0.68	0.63	0.67
TA88	339644.64	384,525	13.21	0.36	0.57	0.61	0.60	1.53	0.65	0.65	0.65	0.65
TA89	330702.73	374,423	13.22	0.64	1.08	1.55	1.53	1.89	1.63	1.63	0.99	1.50
TA90	339754.76	379,296	11.64	0.52	0.81	1.10	1.09	1.52	0.95	0.95	0.66	1.19
TA91	998469.90	1,041,023	4.26	⋆	0.04	0.05	0.05	0.14	0.03	0.03	0.05	0.06
TA92	988625.43	1,028,828	4.07	⋆	⋆	0.01	0.01	0.01	0.01	0.01	0.01	0.01
TA93	1001693.83	1,042,357	4.06	⋆	0.07	0.07	0.08	0.11	0.06	0.06	0.07	0.08
TA94	984205.61	1,025,564	4.20	⋆	0.06	0.05	0.06	0.33	0.01	0.01	0.02	0.12
TA95	985397.30	1,028,963	4.42	⋆	0.03	0.04	0.04	0.04	0.03	0.03	0.04	0.02
TA96	957895.26	998,340	4.22	⋆	0.05	0.06	0.06	0.09	0.06	0.06	0.06	0.06
TA97	1000477.26	1,042,570	4.21	⋆	⋆	⋆	⋆	0.01	⋆	⋆	⋆	⋆
TA98	1000307.85	1,035,915	3.56	⋆	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
TA99	975347.06	1,015,280	4.09	⋆	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
TA100	978537.50	1,021,865	4.43	⋆	0.03	0.04	0.04	0.04	0.03	0.03	0.04	0.04
TA101	1102941.86	1,219,341	10.55	⋆	0.13	0.20	0.19	0.37	0.02	0.02	0.15	0.18
TA102	1115381.71	1,233,161	10.56	⋆	0.04	0.06	0.06	0.07	0.06	0.06	0.03	0.06
TA103	1130764.69	1,259,605	11.39	⋆	⋆	0.01	0.01	0.02	0.01	0.01	0.01	0.01
TA104	1119826.64	1,228,027	9.66	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA105	1105327.45	1,215,854	1.00	⋆	0.02	0.09	0.08	0.33	0.08	0.08	0.06	0.06
TA106	1108726.36	1,218,757	9.92	⋆	0.04	0.04	0.04	0.06	0.04	0.04	0.04	0.03
TA107	1123430.72	1,234,330	9.87	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA108	1123984.67	1,240,105	10.33	⋆	0.02	0.02	0.02	0.04	0.02	0.02	0.02	0.02
TA109	1098798.57	1,220,058	11.04	⋆	⋆	0.01	0.01	0.07	0.01	0.01	0.01	⋆
TA110	1127270.58	1,235,113	9.57	⋆	⋆	0.02	0.02	0.03	0.01	0.01	0.02	0.02
TA111	6191253.65	6,558,109	5.93	⋆	⋆	⋆	⋆	0.01	⋆	⋆	⋆	⋆
TA112	6301558.79	6,679,339	6.00	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA113	6261525.71	6,624,644	5.80	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA114	6281508.02	6,646,006	5.80	⋆	⋆	⋆	0.01	0.06	⋆	⋆	⋆	⋆
TA115	6240001.85	6,587,110	5.56	⋆	0.01	⋆	0.03	0.04	0.01	0.01	0.01	0.01
TA116	6269214.53	6,603,291	5.32	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	⋆
TA117	6223303.96	6,602,685	5.67	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA118	6277413.77	6,629,065	5.60	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
TA119	6203828.97	6,587,638	6.19	⋆	⋆	⋆	0.01	0.02	0.01	0.01	0.01	⋆
TA120	6272934.31	6,623,849	5.59	⋆	⋆	⋆	0.02	0.04	0.01	0.01	⋆	0.01

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2019.03.007.

References

Abedinnia, H., Glock, C. H., & Brill, A. (2016). New simple constructive heuristic algorithms for minimizing total flow-time in the permutation flowshop scheduling problem. *Computers & Operations Research, 74*, 165–174. doi:10.1016/j.cor.2016.04.007.

Almeida, B. F., Correia, I., & Saldanha-da Gama, F. (2018). A biased random-key genetic algorithm for the project scheduling problem with flexible resources. *TOP, 26*(2), 283–308. doi:10.1007/s11750-018-0472-9.

Andrade, C. E., Byers, S. D., Gopalakrishnan, V., Halepovic, E., Majmundar, M., Poole, D. J., et al. (2017). Managing massive firmware-over-the-air updates for connected cars in cellular networks. In *Proceedings of the 2nd ACM international workshop on smart, autonomous, and connected vehicular systems and services*. In *CarSys '17* (pp. 65–72). ACM. doi:10.1145/3131944.3131953.

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 15th annual conference on genetic and evolutionary computation*. In *GECCO'13* (pp. 463–470). New York, NY, USA: ACM. doi:10.1145/2463372.2463434.

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. doi:10.1016/j.asoc.2015.04.016.

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. doi:10.1162/EVCO_a_00138.

den Besten, M., & Stützle, T. (2001). Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. In *Mic'2001–4th metaheuristics international conference* (pp. 545–549).

Biajoli, F. L., Chaves, A. A., & Lorena, L. A. N. (2019). A biased random-key genetic algorithm for the two-stage capacitated facility location problem. *Expert Systems with Applications, 115*, 418–426. doi:10.1016/j.eswa.2018.08.024.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Springer Berlin Heidelberg. doi:10.1007/978-3-642-02538-9_13.

Cabo, M., González-Velarde, J. L., Possani, E., & Ríos Solís, Y. A. (2018). Bi-objective scheduling on a restricted batching machine. *Computers and Operations Research, 100*, 201–210. doi:10.1016/j.cor.2018.07.004.

Chung, C.-S., Flynn, J., & Kirca, O. (2002). A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems. *International Journal of Production Economics, 79*(3), 185–196. doi:10.1016/S0925-5273(02)00234-7.

Della Croce, F., Ghirardi, M., & Tadei, R. (2002). An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. *European Journal of Operational Research, 139*(2), 293–301. doi:10.1016/S0377-2217(01)00374-5.

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

Dong, X., Chen, P., & Huang, H. (2011a). An improved iterated local search algorithm for the permutation flowshop problem with total flowtime. *Advances in automation and robotics. Lecture Notes in Electrical Engineering*: Vol. 122 LNEE.

Dong, X., Chen, P., Huang, H., & Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research, 40*(2), 627–632. doi:10.1016/j.cor.2012.08.021.

Dong, X., Huang, H., & Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research, 36*(5), 1664–1669. doi:10.1016/j.cor.2008.04.001.

Dong, X., Huang, H., & Chen, P. (2011b). Study on iterated local search algorithm for permutation flowshop problem with total flowtime objective. *Applied informatics and communication. Communications in Computer and Information Science*: Vol. 225 CCIS.

Dubois-Lacoste, J., López-Ibáñez, M., & Stützle, T. (2011). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research, 38*(8), 1219–1236. doi:10.1016/j.cor.2010.10.008.

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. doi:10.1016/j.ejor.2010.03.030.

Fernandez-Viagas, V., & Framinan, J. M. (2015). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research, 53*, 68–80. doi:10.1016/j.cor.2014.08.004.

Fernandez-Viagas, V., & Framinan, J. M. (2017). A beam-search-based constructive heuristic for the PFSP to minimise total flowtime. *Computers & Operations Research, 81*, 167–177. doi:10.1016/j.cor.2016.12.020.

Framinan, J. M., Leisten, R., & Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research, 32*(5), 1237–1254. doi:10.1016/j.cor.2003.11.002.

Fuchigami, H. Y., & Rangel, S. (2018). A survey of case studies in production scheduling: Analysis and perspectives. *Journal of Computational Science, 25*, 425–436. doi:10.1016/j.jocs.2017.06.004.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research, 1*(2), 117–129.

Gonçalves, J. F., & Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics, 17*(5), 487–525. doi:10.1007/s10732-010-9143-1.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of discrete mathematics: 5* (pp. 287–326). Elsevier.

Grasas, A., Ramalhinho, H., Pessoa, L. S., Resende, M. G., Caballé, I., & Barba, N. (2014). On the improvement of blood sample collection at clinical laboratories. *BMC Health Services Research, 14*(1), 12. doi:10.1186/1472-6963-14-12.

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

Hoos, H. H., & Stützle, T. (2004). *Stochastic local search: Foundations & applications*. Elsevier / Morgan Kaufmann.

Jarboui, B., Eddaly, M., & Siarry, P. (2009). An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Computers & Operations Research, 36*(9), 2638–2646. doi:10.1016/j.cor.2008.11.004.

Juan, A. A., Lourenço, H. R., Mateo, M., Luo, R., & Castella, Q. (2014). Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues. *International Transactions in Operational Research, 21*(1), 103–126.

Li, X., & Zhang, K. (2018). Single batch processing machine scheduling with two-dimensional bin packing constraints. *International Journal of Production Economics, 196*, 113–121. doi:10.1016/j.ijpe.2017.11.015.

Liu, J., & Reeves, C. R. (2001). Constructive and composite heuristic solutions to the P//ΣC_i scheduling problem. *European Journal of Operational Research, 132*(2), 439–452. doi:10.1016/S0377-2217(00)00137-5.

Lopes, M. C., Andrade, C. E., Queiroz, T. A., Resende, M. G. C., & Miyazawa, F. K. (2016). Heuristics for a hub location-routing problem. *Networks, 68*(1), 54–90. doi:10.1002/net.21685.

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. doi:10.1016/j.orp.2016.09.002.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 129–168)). Cham: Springer International Publishing. doi:10.1007/978-3-319-91086-4_5.

Madhushini, N., Rajendran, C., & Deepa, Y. (2009). Branch-and-bound algorithms for scheduling in permutation flowshops to minimize the sum of weighted flowtime/sum of weighted tardiness/sum of weighted flowtime and weighted tardiness/sum of weighted flowtime, weighted tardiness and weighted earliness of jobs. *Journal of the Operational Research Society, 60*(7), 991–1004. doi:10.1057/palgrave.jors.2602642.

Martí, R., Panos, P., & Resende, M. (2017). *Handbook of Heuristics*. Springer International Publishing.

Moscato, P. (1989). On evolution, search, optimization, GAs and martial arts: Toward memetic algorithms. *Tech. Rep*. California Inst. Technol.

Pan, Q.-K., & Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research, 222*(1), 31–43. doi:10.1016/j.ejor.2012.04.034.

Pan, Q.-K., & Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research, 40*(1), 117–128. doi:10.1016/j.cor.2012.05.018.

Pan, Q.-K., Tasgetiren, M., & Liang, Y.-C. (2007). *A discrete differential evolution algorithm for the permutation flowshop scheduling problem* (pp. 126–133). doi:10.1145/1276958.1276976.

Pan, Q.-K., Tasgetiren, M. F., & Liang, Y.-C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering, 55*(4), 795–816. doi:10.1016/j.cie.2008.03.003.

Pessoa, L. S., & Andrade, C. E. (2018). Heuristics for a flowshop scheduling problem with stepwise job objective function. *European Journal of Operational Research, 266*(3), 950–962. doi:10.1016/j.ejor.2017.10.045.

Pessoa, L. S., Santos, A. C., & Resende, M. G. (2017). A biased random-key genetic algorithm for the tree of hubs location problem. *Optimization Letters, 11*(7), 1371–1384. doi:10.1007/s11590-016-1082-9.

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

Prasetyo, H., Fauza, G., Amer, Y., & Lee, S. H. (2015). Survey on applications of biased-random key genetic algorithms for solving optimization problems. In *Industrial engineering and engineering management (IEEM), 2015 IEEE international conference on* (pp. 863–870). IEEE.

Prasetyo, H., Putri, A. L., & Fauza, G. (2018). Biased random key genetic algorithm design with multiple populations to solve capacitated vehicle routing problem with time windows. In *Aip conference proceedings: 1977* (p. 020052). AIP Publishing.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & operations research, 22*(1), 5–13. doi:10.1016/0305-0548(93)E0014-K.

Resende, M. G. C. (2012). Biased random-key genetic algorithms with applications in telecommunications. *TOP, 20*(1), 130–153. doi:10.1007/s11750-011-0176-x.

Ribas, I., Companys, R., & Tort-Martorell, X. (2013). An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. *International Journal of Production Research, 51*(17), 5238–5252.

Rossi, F. L., Nagano, M. S., & Sagawa, J. K. (2017). An effective constructive heuristic for permutation flow shop scheduling problem with total flow time criterion. *The International Journal of Advanced Manufacturing Technology, 90*(1–4), 93–107. doi:10.1007/s00170-016-9347-0.

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. doi:10.1016/j.ejor.2005.12.009.

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. doi:10.1016/j.ejor.2006.07.029.

Stafford, E. (1988). On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *Journal of the Operational Research Society, 39*(12), 1163–1174. doi:10.1057/jors.1988.193.

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

Stützle, T., & Ruiz, R. (2017). Iterated greedy. In R. Martí, P. Panos, & M. G. C. Resende (Eds.), *Handbook of Heuristics* (pp. 1–31)). Cham: Springer International Publishing. doi:10.1007/978-3-319-07153-4_10-1.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing, 17*(4), 443–455. doi:10.1016/S0167-8191(05)80147-4.

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. doi:10.1016/j.cor.2016.07.002.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P. N., & Chen, A. H. (2011). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences, 181*(16), 3459–3475. doi:10.1016/j.ins.2011.04.018.

Tseng, L.-Y., & Lin, Y.-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 198*(1), 84–92. doi:10.1016/j.ejor.2008.08.023.

Tseng, L.-Y., & Lin, Y.-T. (2010). A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics, 127*(1), 121–128. doi:10.1016/j.ijpe.2010.05.003.

van de Velde, S. L. (1991). Minimizing the sum of the job completion times in the two-machine flow shop by lagrangian relaxation. *Annals of Operations Research, 26*(1–4), 257–268.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly, 6*(2), 131–140.

Xu, J., Wu, C.-C., Yin, Y., & Lin, W.-C. (2017). An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times. *Applied Soft Computing, 52*, 39–47.

Xu, X., Xu, Z., & Gu, X. (2011). An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems With Applications, 38*(7), 7970–7979. doi:10.1016/j.eswa.2010.12.075.

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. doi:10.1016/j.ejor.2008.04.033.