European Journal of Operational Research 257 (2017) 707–721

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Invited Review

A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation

Victor Fernandez-Viagas[a], Rubén Ruiz[b,*], Jose M. Framinan[a]

[a] Industrial Management, School of Engineering, University of Seville, Ave. Descubrimientos s/n, Seville E41092, Spain
[b] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021 València, Spain

A R T I C L E I N F O

Article history:
Received 25 October 2015
Accepted 28 September 2016
Available online 8 October 2016

Keywords:
Scheduling
Flowshop
Heuristics
Metaheuristics
Computational evaluation

A B S T R A C T

The permutation flowshop problem is a classic machine scheduling problem where n jobs must be processed on a set of m machines disposed in series and where each job must visit all machines in the same order. Many production scheduling problems resemble flowshops and hence it has generated much interest and had a big impact in the field, resulting in literally hundreds of heuristic and metaheuristic methods over the last 60 years. However, most methods proposed for makespan minimisation are not properly compared with existing procedures so currently it is not possible to know which are the most efficient methods for the problem regarding the quality of the solutions obtained and the computational effort required. In this paper, we identify and exhaustively compare the best existing heuristics and metaheuristics so the state-of-the-art regarding approximate procedures for this relevant problem is established.

1. Introduction

The flowshop is a common manufacturing layout where n jobs have to be processed on m machines, with each job following the same route at the machines. The so-called flowshop scheduling problem involves the determination of the sequence of jobs at each machine. When the sequence of jobs is the same for all machines, the problem is denoted as Permutation Flowshop Scheduling Problem (PFSP in the following). The PFSP is one of the most studied problems in the Operations Research literature (e.g., see the reviews by Framinan, Gupta, and Leisten (2004); Reza Hejazi and Saghafian (2005); Ruiz and Maroto (2005)).

In the related literature, the minimization of makespan, $C_{\max}$, (also denoted as maximum completion time or maximum flowtime) has been commonly chosen by researchers as the objective to optimize in the PFSP (e.g., see Fernandez-Viagas and Framinan (2015b); Framinan and Leisten (2006); Leisten and Rajendran (2014); M'Hallah (2014); Vallada and Ruiz (2010) or Sun, Zhang, Gao, and Wang (2011) for other objectives in the PFSP). According to the notation of Pinedo (2012), this problem is denoted as $Fm|prmu|C_{\max}$. Since (Rinnooy Kan, 1976) showed the problem to be NP-complete for more than two machines, most researchers have focused on implementing approximate methods to find good solutions without excessive computation times.

There has been a vast number of papers published with algorithms and procedures. Ruiz and Maroto (2005) carried out an exhaustive review and computational evaluation of the heuristics and metaheuristics published until 2004 for the PFSP to minimize makespan. A total of 18 heuristics and 7 metaheuristics were implemented and tested under the same conditions. Among them, two of these methods turned out to be the most efficient ones: the NEH heuristic (Nawaz, Enscore Jr., & Ham, 1983) was clearly the most efficient among the constructive heuristics for the problem, and the Iterated Local Search (Stütze, 1998) presented itself as the most efficient metaheuristic for the problem.

Since the publication of the work by Ruiz and Maroto (2005), more than 100 new algorithms have been proposed in the literature over the last 10 years. Some of these methods – such as the iterated greedy (IG) of Ruiz and Stützle (2007) – have improved the best existing algorithms in Ruiz and Maroto (2005). However, the new state-of-the-art remains unclear due to the lack of a homogeneous framework to conduct the comparison among algorithms. More specifically, the following problems can be detected:

- Many algorithms are compared under different conditions:
  - Tested under different computer conditions (different programming languages and/or different computers, operating systems, etc.).

* Corresponding author.
E-mail addresses: vfernandezviagas@us.es (V. Fernandez-Viagas), rruiz@eio.upv.es (R. Ruiz), framinan@us.es (J.M. Framinan).

http://dx.doi.org/10.1016/j.ejor.2016.09.055
0377-2217/

  - Comparison of algorithms with different CPU time usages.
  - Use of different benchmarks (see Section 2).
- Many algorithms are compared in a non-conclusive way:
  - Lack of comparison against the state-of-the-art (e.g. without comparing with the iterated greedy proposed by Ruiz & Stützle (2007)).
  - Among the several runs performed in each instance to increase the power of the results, the best runs are used instead of the average for some algorithms.
- New advances in the evaluation of the algorithms:
  - A more extensive benchmark of instances has been recently proposed by Vallada, Ruiz, and Framinan (2015). This testbed can be used to establish statistical differences among algorithms in a sound way, differently from what can be done with older benchmarks (such as those by Carlier (1978); Demirkol, Mehta, and Uzsoy (1998); Heller (1960); Reeves (1995); Taillard (1993); Watson, Barbulescu, Whitley, and Howe (2002)).
  - A new indicator has been proposed by Fernandez-Viagas and Framinan (2015a) to measure the CPU requirements of the algorithms in relative terms. This indicator improves the deficiencies of the most common indicator (i.e., average CPU time) for the evaluation of efficient heuristics.
- Finally, a special effort should be made when comparing efficient heuristics against the best metaheuristics under the same stopping criterion since the CPU time required by some heuristics is relatively high in comparison with some metaheuristics.

As a conclusion, a new review and evaluation of the approximate methods for the $Fm|prmu|C_{\max}$ problem is pertinent and may serve firstly to establish a clear picture of the state-of-the-art within this important problem, and secondly, to give indications of possible avenues for future research. This twofold objective is the goal of our research.

The remainder of the paper is as follows: in Section 2, heuristics and metaheuristics published in the literature from Ruiz and Maroto (2005) are analysed and summarised. The most promising ones are chosen to be evaluated and compared. A description of the evaluation and comparison is carried out in Section 3. Computational results of the comparisons between heuristics and metaheuristics are described in Section 4. Finally, in Section 5 conclusions are discussed and some indications and ideas for future research are shown.

2. Background

The problem under consideration is the permutation flowshop scheduling problem to minimise the maximum completion time or makespan. The problem consists of the determination of the sequence of n jobs which achieves the minimal makespan when all jobs are processed (in the order indicated by the sequence) on the m machines of a shop. The following additional hypotheses are usually assumed for the PFSP:

- Processing times, denoted as p_{ij} where $i = 1, \ldots, m$ and $j = 1, \ldots, n$, are known and deterministic.
- No preemption is allowed.
- Release times are set to 0.
- Sequence-dependent set-up times are considered insignificant.
- Sequence-independent setup times are considered as non-anticipatory, and therefore can be added to the processing time of the jobs on the machines.
- Transportation times can be considered either insignificant or constant.
- Each job can be processed by at most one machine at the same time.
- Each machine can process only one job at the same time.
- Unlimited in-process inventory is considered.
- All machines are available on the whole scheduling horizon.

As mentioned in the previous section, the NP-hard nature of the problem has led the vast majority of research towards the proposal of approximate solutions, usually classified either as heuristics or metaheuristics. The division between heuristics and metaheuristics is ambiguous and different classifications have been proposed in the literature (see e.g., Zanakis, Evans, and Vazacopoulos (1989); Zäpfel, Braune, and Bögl (2010)). For an in-depth classification of the $Fm|prmu|C_{\max}$ problem, we refer to Framinan et al. (2004). However, in this paper we use the same division as in Ruiz and Maroto (2005), where heuristics and metaheuristics are analysed separately. There, heuristics (constructive and improvement ones) naturally stop when the procedure is finished whereas metaheuristics typically stop after a given number of iterations or elapsed CPU time. This fact naturally leads to perform different computational experiments in Section 4, since the efficiency of the metaheuristics can be compared by running them for the same CPU time whereas heuristics should be compared by means of a Pareto-efficient frontier using the quality of the solution and the CPU time as indicators. In order to maintain the readability of the paper, the same division is considered when analysing the state-of-the-art in this section.

2.1. Heuristics

Traditionally divided into constructive and improvement types, heuristics have been extensively developed for $Fm|prmu|C_{\max}$ either to yield a good solution in less CPU time or to find a seed sequence for metaheuristics. Since the computational evaluation of Ruiz and Maroto (2005), several constructive heuristics have been proposed in the literature, most of them variants of the NEH heuristic by Nawaz et al. (1983). This heuristic consists of two phases:

1. First, jobs are ordered according to an initial order (decreasing sum of processing times).
2. The first job is removed from the initial order and placed in a partial sequence, initially without any job. Next, following this order, each job is removed and tried to be inserted in each possible position of the partial sequence. The position that minimizes the makespan is chosen for the job. The procedure is repeated $n-1$ times until all jobs are placed in the partial sequence.

The computational complexity of the NEH is $O(n^3m)$. However, the method proposed by Taillard (1990) (denoted as Taillard's acceleration in the following) reduces its original complexity to $O(n^2m)$.

The different variants of the NEH heuristic can be unified using the following notation formed by three fields: $NEH(a|b|c)$ where the fields a, b and c are defined by:

- a: Initial order used by the NEH. In the computational evaluation, the following sorting criteria have been considered:
  - rand: Jobs are randomly ordered. This order is used by Ribas, Companys, and Tort-Martorell (2010) in RAER and RAER-di heuristics as comparison heuristics.
  - SD: Non decreasing sum of processing times (original order of the NEH) of the jobs. This order is used by the following heuristics: NEHR (Ribas et al., 2010), NEHR-di(Ribas et al., 2010), NEH (Nawaz et al., 1983), NEH-di (Ribas et al., 2010), NEH1 (Kalczynski & Kamburowski, 2007) and NEH1-di (Ribas et al., 2010).
  - AD: sum of the mean and deviation of the processing times (proposed by Dong, Huang, & Chen (2008)).

Table 1
Summary of heuristics.

Heuristic	NEH Notation	Paper
RAER	*NEH*(*rand*|*RCT*|*d*)	Ribas et al. (2010)
RAER-di	*NEH*(*rand*|*RCT*|*di*)	Ribas et al. (2010)
KKER	*NEH*(*KK*1|*RCT*|*d*)	Ribas et al. (2010)
KKER-di	*NEH*(*KK*1|*RCT*|*di*)	Ribas et al. (2010)
NEHR	*NEH*(*SD*|*RCT*|*d*)	Ribas et al. (2010)
NEHR-di	*NEH*(*SD*|*RCT*|*di*)	Ribas et al. (2010)
NEMR	*NEH*(*NM*|*RCT*|*d*)	Ribas et al. (2010)
NEMR-di	*NEH*(*NM*|*RCT*|*di*)	Ribas et al. (2010)
NEH	*NEH*(*SD*|*FS*|*d*)	Nawaz et al. (1983)
NEH-di	*NEH*(*SD*|*FS*|*di*)	Ribas et al. (2010)
NEH1	*NEH*(*SD*|*TBKK*1|*d*)	Kalczynski and Kamburowski (2007)
NEH1-di	*NEH*(*SD*|*TBKK*1|*di*)	Ribas et al. (2010)
NEHKK1	*NEH*(*KK*1|*TBKK*2|*d*)	Kalczynski and Kamburowski (2008)
NEHKK1-di	*NEH*(*KK*1|*TBKK*2|*di*)	Ribas et al. (2010)
NEHKK2	*NEH*(*KK*2|*TBKK*3|*d*)	Kalczynski and Kamburowski (2009)
NEHD	*NEH*(*AD*|*DHC*|*d*)	Dong et al. (2008)
NEHD-di	*NEH*(*AD*|*DHC*|*di*)	Ribas et al. (2010)
NEHFF	*NEH*(*AD*|*FF*|*d*)	Fernandez-Viagas and Framinan (2014)
CL_{WTS}	*NEH*(*SD*|*FS*|*d*) with a backward shift mechanism in the insertion phase	Ying and Lin (2013)
CL_{WOTS}	*NEH*(*SD*|*RCT*|*d*) with a backward shift mechanism in the insertion phase	Ying and Lin (2013)
NEHI	Best of several runs of $NEH(-\|-\|-)$	Vasiljevic and Danilovic (2015)
FRB1	Similar to the *NEH*(*SD*|*FS*|*d*) including a local search method in the insertion phase	Rad et al. (2009)
FRB2	Similar to the *NEH*(*SD*|*FS*|*d*) including a local search method in the insertion phase	Rad et al. (2009)
FRB3	*NEH*(*SD*|*FS*|*d*) including a local search method in the insertion phase	Rad et al. (2009)
$FRB4_k$	*NEH*(*SD*|*FS*|*d*) including a local search method in the insertion phase	Rad et al. (2009)
FRB5	*NEH*(*SD*|*FS*|*d*) including a local search method in the insertion phase	Rad et al. (2009)

  - NM: order proposed by Nagano and Moccellin (2002) and used in NEMR and NEMR-di heuristics by Ribas et al. (2010).
  - KK1: Sorting criterion proposed by Kalczynski and Kamburowski (2008). This initial order is applied in NEHKK1 (Kalczynski & Kamburowski, 2008) and NEHKK1-di (Ribas et al., 2010) heuristics.
  - KK2: Sorting criterion proposed by Kalczynski and Kamburowski (2009) in NEHKK2 heuristic.
- *b*: Once a job is selected for insertion in all positions of a partial sequence, the same makespan can be obtained for several positions causing ties in each iteration. These ties have a great influence on the performance of the constructive heuristics (see Kalczynski & Kamburowski (2007)). In the original proposal, the first slot (denoted as FS) for which the minimum makespan is achieved is kept as the best sequence. This *b* field then defines the type of tie-breaking mechanism implemented in the NEH. The following mechanisms have been considered: TBKK1, proposed by Kalczynski and Kamburowski (2007); TBKK2, proposed by Kalczynski and Kamburowski (2008); TBKK3, proposed by Kalczynski and Kamburowski (2009); DCH, proposed by Dong et al. (2008); RCT, proposed by Ribas et al. (2010); and the FF, proposed by Fernandez-Viagas and Framinan (2014).
- *c*: This field is associated with the reversibility property of the problem (see Ribas et al. (2010)). It establishes that the makespan of the permutation $\Pi := (\pi_1, \ldots, \pi_n)$ in instance I (instance formed by n jobs and m machines with processing times equal to p_{ij}) is the same as the makespan of the reverse permutation $\Pi' := (\pi_n, \ldots, \pi_1)$ in instance I' (instance formed by n jobs and m machines with processing times equal to $p'_{ij} = p_{m-j+1,i}$). Therefore, the value d indicates that the NEH is applied on the direct instance I whereas i is used when the algorithm is applied on the inverse instance I'. Accordingly, *di* indicates that both the direct and the inverse are used, and the best sequence is retained.

This notation has been employed to classify the different variants of the original NEH heuristic –which can be denoted as *NEH*(*SD*|*FS*|*d*) in our notation– proposed in the literature. These are summarized in Table 1.

Among the heuristics proposed, some of them – i.e., NEH1, NEHKK1, NEHKK2, NEHD and NEHFF by Dong et al. (2008); Kalczynski and Kamburowski (2007, 2008, 2009) and Fernandez-Viagas and Framinan (2014) respectively – maintain the original complexity of $O(n^2m)$. Other variants with a greater complexity have been proposed by Ribas et al. (2010), see Table 1 (the heuristics implemented in this research are indicated in bold, see Section 3).

Two different variants with a greater complexity have been proposed by Ying and Lin (2013) and are denoted as CL_{WOTS} and CL_{WTS}. In CL_{WOTS}, a new mechanism (denoted as backward shift mechanism) is added to the traditional insertion phase of the NEH. This mechanism increases the sequences to be evaluated in each iteration by means of a movement of the jobs of the partial sequence. When the tie-breaking mechanism of Ribas et al. (2010) is added to the CL_{WOTS}, the heuristic is denoted as CL_{WTS}

Furthermore, 10 heuristics that also modify the insertion phase of the NEH algorithm have been proposed by Rad, Ruiz, and Boroojerdian (2009). These heuristics are denoted as: FRB1, FRB2, FRB3, $FRB4_2$, $FRB4_4$, $FRB4_6$, $FRB4_8$, $FRB4_{10}$, $FRB4_{12}$ and FRB5. Among then, the FRB1 heuristic is statistically outperformed by several heuristics (e.g., $FRB4_2$ and $FRB4_4$) with shorter average CPU times. Finally, Vasiljevic and Danilovic (2015) proposed a constructive NEH-based heuristic, NEHI, which also considers different interpretations for the ties in the initial order of the NEH.

2.2. Metaheuristics

As explained in Section 1, numerous metaheuristics have been published in the literature since 2004. A summary of them is shown in Tables 2 and 3, where the metaheuristics implemented in this research are indicated in bold (see Section 3). The first, second, third and fourth columns indicate the year of publication, the bibliographical reference, the type of metaheuristic and the acronym (maintaining the same acronym as in the original papers) respectively. The fifth column shows the papers proposing metaheuristics that outperform the referenced one. In the sixth column, the benchmark(s) used for the computational evaluation are shown (the following notation is used: T1, Taillard (1993); T2,

Table 2
Summary of metaheuristics I.

Year	Ref.	Algorithm	Notation	Outperformed by	Testbed	*ARPD*(Taillard)	Coding Lang.	Parameter *t*
2004	Ying and Liao (2004)	AC	ACS	Ahmadizar (2012); Eksioglu et al. (2008); Liu and Liu (2013); Saravanan, Noorul Haq, Vivekraj, and Prasad (2008)	T1	1.4[d]	C++	608.11
2004	Solimanpur et al. (2004)	Neuro-TS	EXTS	Eksioglu et al. (2008)	T2	0.245	C++	2884.85
2004	Rajendran and Ziegler (2004)	AC	PACO	Ruiz et al. (2006)[b], Ruiz and Stützle (2007)[a], Pan et al. (2008); Pan et al. (2008)[c], Ahmadizar (2012); Laha and Chakraborty (2009); Li and Yin (2012); Tzeng and Chen (2012); Zhang and Wu (2014); Zobolas, Tarantilis, and Ioannou (2009)[a], Chen, Tzeng, and Chen (2015); Tseng and Lin (2009)	T2	0.71[d]	–	–
2004	Rajendran and Ziegler (2004)	AC	M-MMAS	Ruiz et al. (2006)[b], Ruiz and Stützle (2007)[a], Pan et al. (2008); Pan et al. (2008)[c], Ahmadizar (2012); Laha and Chakraborty (2009); Li and Yin (2012); Tzeng and Chen (2012); Zobolas et al. (2009)[a], Tseng and Lin (2009)	T2	0.80[d]	–	–
2004	Low, Yeh, and Huang (2004)	SA	LWK-SA1	Li and Yin (2013a, 2013b); Xie, Zhang, Shao, Lin, and Zhu (2014)	T2,D	0.853	–	331.8
2004	Etiler, Toklu, Atak, and Wilson (2004)	GA	GA	Liao, Tseng, and Luarn (2007)[a]	O	–	Pascal	–
2004	Nearchou (2004b)	SA	Hybrid SAA	–	T2	0.893	Pascal	–
2004	Nearchou (2004c)	Hybrid SA	Hybrid SAA	–	T2	1.081	Pascal	134.06
2004	Nearchou (2004a)	GA	GA	Zhang, Sun, Zhu, and Yang (2008)[a], Zhang, Ning, and Ouyang (2010a)[a], Zhang and Sun (2009); Zhang, Zhang, and Liang (2010b)	T2	1.84[d]	Pascal	–
2006	Agarwal, Colak, and Eryarsoy (2006)	ALA	NEH-ALA	Nagano, Ruiz, and Lorena (2008)[a], Laha and Chakraborty (2009)	T1,C,R,H	1.514	Visual Basic	7907.6
2006	Ruiz et al. (2006)	GA	GA_RMA	Ruiz et al. (2006)[b], Ruiz and Stützle (2007)[a], Pan et al. (2008); Pan et al. (2008)[c], Nagano et al. (2008)[a], Chen and Hsieh (2014); Li and Yin (2012); Zhang et al. (2008)[a]	T1	1.12[d], 1.09[d], 1.02[d]	Delphi	30, 60, 90
2006	Ruiz et al. (2006)[b]	GA	HGA_RMA	Ruiz and Stützle (2007)[a], Pan et al. (2008); Pan et al. (2008)[c], Chen et al. (2015); Tzeng and Chen (2012)	T1	0.55[d], 0.47[d], 0.45[d]	Delphi	30, 60, 90
2006	Onwubolu and Davendra (2006)	DE	DE	Nagano et al. (2008)[a], Qian et al. (2008)	O	–	C++	–
2006	Nowicki and Smutnicki (2006)	SS	**MSSA**	–	T2	>0.054	–	1139.33
2006	Lian, Gu, and Jiao (2006)	PSO	SPSOA	Chang and Chen (2014); Chang, Chen, Tiwari, and Iquebal (2013); Chang, Huang, and Ting (2011); Hsu, Chang, and Chen (2015); Zhang et al. (2010b)	T2	3.002	–	–
2006	Prabhaharan, Khan, and Rakesh (2006)	GRASP	GRASP	–	T2,C,R	19.09	–	–
2006	Huang and Wang (2006)	ILS	LS	–	C,R	–	C	
2007	Ruiz and Stützle (2007)	IG	**IG_RS**$_{LS}$	Pan et al. (2008); Pan et al. (2008)[c], Fernandez-Viagas and Framinan (2014)[b,a]	T1	0.44[d]	Delphi	60
2007	Tasgetiren, Liang, Sevkli, and Gencyilmaz (2007)	PSO	PSO_{spv}	Pan et al. (2008)[b], Chen, Chang, Cheng, and Zhang (2012); Tasgetiren et al. (2007)[b], Jarboui et al. (2008); Liao et al. (2007)[a], Marinakis and Marinaki (2013)	T2	4.01[d]	C	21.1
2007	Tasgetiren et al. (2007)[b]	PSO	PSO_{vns}	Pan et al. (2008); Pan et al. (2008)[c], Chen et al. (2015); Li and Yin (2013a, 2013b); Lin, Gao, Li, and Zhang (2015); Liu, Wang, and Jin (2007); Liu, Yin, and Gu (2014); Liu and Liu (2013); Tseng and Lin (2009); Tzeng and Chen (2012); Xie et al. (2014)	T2,W	0.47[d]	C	264.64
2007	Liu et al. (2007)	MA-PSO	PSOMA	Chang et al. (2013); Li and Yin (2013a); Lin et al. (2015); Liu, Gao, and Pan (2011); Liu, Ma, Ma, and Li (2013); Liu et al. (2014); Liu and Liu (2013)	C,R	–	Matlab	–
2007	Liao et al. (2007)	PSO	PSO	Dasgupta and Das (2015); Li and Yin (2013b); Zheng and Yamashiro (2010)	T2,D	2.409[d]	C++	111.68
2008	Eksioglu et al. (2008)	TS	**3XTS**	–	T1	0.15[e]	C++	196.93
2008	Saravanan et al. (2008)	SS	SS	–	T1	1.57[e]	C++	59.9
2008	Nagano et al. (2008)	GA	CGALS	Liu and Liu (2013)	T1	1.02[d]	Delphi	60
2008	Framinan and Pastor (2008)	GA	BDS	–	T2	0.64	–	7350
2008	Jarboui et al. (2008)	PSO	CPSO	Pan et al. (2008); Pan et al. (2008)[b], Pan et al. (2008)[c], Chen et al. (2012)	T2	3.4[d]	C++	3.42
2008	Jarboui et al. (2008)[b]	PSO	CPSO-PNEH	–	T2	0.59[d]	C++	19.12
2008	Jarboui et al. (2008)[c]	PSO	**H-CPSO**	Pan et al. (2008); Pan et al. (2008)[c]	T2	0.45[d]	C++	229.34
2008	Chang, Chen, Fan, and Chan (2008)	GA	ACGA	Chang et al. (2011); Hsu et al. (2015)	R	–	–	–
2008	Qian et al. (2008)	DE	HDE	Li and Yin (2012, 2013a, 2013b); Zheng and Yamashiro (2010)	C,R	–	Delphi	–
2008	Lian, Gu, and Jiao (2008)	PSO	NPSO	Zhang et al. (2010a)[a], Zhang et al. (2008)[a], Kuo et al. (2009); Liu et al. (2014); Zhang and Sun (2009)	T2	1.323	–	–
2008	Yagmahan and Yenisey (2008)	AC	ACS	Zhang et al. (2008)[a], Li and Yin (2012)	R	–	VP	–
2008	Pan et al. (2008)	IG	**IG**$_{RIS}$	Fernandez-Viagas and Framinan (2014)[a]	T1	0.33[d]	C++	30
2008	Pan et al. (2008)[b]	DE	DDE	Pan et al. (2008)[a], Pan et al. (2008)[c,a], Hsu et al. (2015)	T1	1.05[d]	C++	30
2008	Pan et al. (2008)[c]	DE	**DDE**$_{RLS}$	–	T1	0.32[d]	C++	30
2008	Zhang et al. (2008)	PSO	IPSO	Li and Yin (2012)	T1	0.76[d]	C++	120

Notation: AC, Ant Colony Algorithm; TS, Tabu Search; ALA, Adaptive learning approach; GA, Genetic Algorithm; IG, Iterated Greedy; ILS, Iterated local search; DE, Differential Evolution; SS, Scatter Search; DF, Discrete Firefly; BCA, Bee colony algorithm; PSO, Particle Swarm Optimization Algorithm; SA, Simulated Annealing; CS, Cuckoo Search; NN, Neural Network; EA, Evolutionary algorithm; EDA, Estimation of Distribution Algorithm; PA, Population based Algorithm;

[a] Compared under the same conditions;
[b] In case of a paper proposing two methods, it is used to distinguish the second one from the first one;
[c] In case of a paper proposing three methods, it is used to distinguish the third one from the first and second one;
[d] *ARPD* taken directly from the paper;
[e] *ARPD* corrected by 0.565.

Table 3
Summary of metaheuristics II.

Year	Ref.	Algorithm	Notation	Outperformed by	Testbed	*ARPD* (Taillard)	Coding Lang.	Parameter *t*
2009	Chang, Hsieh, Chen, Lin, and Huang (2009)	GA	ACEGA	–	R	–	–	–
2009	Laha and Chakraborty (2009)	Hybrid	PSA	–	T2	1.049	C	–
2009	Zobolas et al. (2009)	GA with VNS	NEGA_{VNS}	Tzeng and Chen (2012), Chen et al. (2015)	T1	0.468	C++	112.93
2009	Zhang and Sun (2009)	PSO	ATPPSO	Li and Yin (2013b), Li and Yin (2013a), Liu et al. (2014)	T2	1.269	–	–
2009	Kuo et al. (2009)	Hybrid PSO	HPSO	Liu et al. (2014)	T2	0.760	C	555.04
2009	Tseng and Lin (2009)	Hybrid GA	Hybrid GA	Dasgupta and Das (2015)	T2	0.63***	C++	199.79
2009	Rajkumar and Shahabudeen (2009)	GA	IGA	–	C,R	–	C	–
2010	Sayadi, Ramezanian, and Ghaffari-Nasab (2010)	DF	Discrete Firefly	–	D	–	Matlab	–
2010	Zheng and Yamashiro (2010)	DE	QDEA	Xie et al. (2014), Li and Yin (2012), Li and Yin (2013b)	C,R,D	–	–	–
2010	Zhang et al. (2010b)	PSO	L-CDPSO	Li and Yin (2013b), Li and Yin (2013a)	T2	0.409	–	–
2010	Zhang et al. (2010a)	PSO	I-ATTPSO	–	T2	1.331	–	–
2010	Haq, Ramanan, Shashikant, and Sridharan (2010)	NN-GA	ANN-GA-RIPS	–	T2	2.519	–	–
2010	Chang, Chen, Fan, and Mani (2010)	GA	ACGA	Chang et al. (2013), Hsu et al. (2015)	R	–	–	–
2011	Chang et al. (2011)	GA	HGIA	Chang et al. (2013), Hsu et al. (2015)	T2,R	1.16	C++	–
2011	Liu et al. (2011)	Hybrid PSO	PSO-EDA_PI	Liu et al. (2013), Chang et al. (2013)	C,R,W	–	Matlab	–
2011	Ramanan, Sridharan, Shashikant, and Haq (2011)	NN	ANN-GA	–	T2	2.34**	–	–
2012	Chen et al. (2012)	GA	Self-Guided GA	Pan et al. (2008)',Hsu et al. (2015)	T2	1.85**	Java	–
2012	Tzeng and Chen (2012)	EDA with AC	$\mathbf{EDA}_{\text{ACS}}$	–	T1	0.572**, 0.508**, 0.463**	C	30, 60, 90, 200
2012	Li and Yin (2012)	BCA	CDABC	–	C,R,T1,D	0.62**	Matlab	120
2012	Ahmadizar (2012)	AC	NACA	–	T1	0.582**	C++	108.92
2013	Liu and Liu (2013)	Hybrid BCA	HDABC	–	T2,R	0.48**	C++	42.52
2013	Marinakis and Marinaki (2013)	PSO	PSOENT	Zobolas et al. (2009)	T1	1.65**	Fortran	104.42
2013	Liu et al. (2013)	MA-PSO	MPSOMA	–	C,R	–	Matlab	–
2013	Li and Yin (2013b)	DE-MA	ODDE	Xie et al. (2014)	C,R,T2,D	0.400	Matlab	–
2013	Li and Yin (2013a)	CS	**HCS**	–	C,R,T2,D	0.401	Matlab	–
2013	Chang et al. (2013)	EA	BBEA	Hsu et al. (2015)	T2,R	1.75**	–	–
2014	Xie et al. (2014)	Hybrid	HTLBO	–	C,R,D	–	C++	–
2014	Zhang and Wu (2014)	PSO	**PSO**	–	T2	0.39**	C++	–
2014	Fernandez-Viagas and Framinan (2014)	IG	$\mathbf{IG}_{\text{RIS}}\mathbf{(TB_{FF})}$	–	T1	0.461, 0.385, 0.353**	C#	30, 60, 90
2014	Fernandez-Viagas and Framinan (2014)'	IG	$\mathbf{IG\_RS}_{\text{LS}}\mathbf{(TB_{FF})}$	–	T1	0.461, 0.376, 0.350**	C#	30, 60, 90
2014	Chen and Hsieh (2014)	SA	SEASA	–	T1	0.94**	–	35
2014	Liu et al. (2014)	Hybrid DE	L-HDE	–	T2,C,R	0.750	Matlab	–
2014	Chang and Chen (2014)	EDA	BBEDA	Hsu et al. (2015)	T2,R	1.420**	–	–
2015	Hariharan and Golden Renjith Nimal (2014)	GA-SS	HGSS	–	D	–	C++	–
2015	Chen et al. (2015)	PA	HLBS	–	T1	0.45**, 0.38**, 0.35**, 0.30**	C	30, 60, 90, 200
2015	Dasgupta and Das (2015)	CS	DISCS	–	T2,W	2.84**	Matlab	10.88
2015	Hsu et al. (2015)	EA	LMBBEA	–	T2,R	0.89	–	–
2015	Lin et al. (2015)	EA	HBSA	Qian et al. (2008)	C,R	–	–	–

non-complete set of instances of Taillard (1993); R, Reeves (1995), C, Carlier (1978); D, Demirkol et al. (1998); W, Watson et al. (2002); H, Heller (1960); O, Other set of instances). The seventh column shows the *ARPD* values of the metaheuristics when tested on Taillard's benchmark (Taillard, 1993). Average Relative Percentage Deviation values of algorithm *j* are denoted as $ARPD_j$ and are calculated as follows:

$$ARPD_j = \frac{\sum_{\forall i} RPD_{i,j}}{I} \quad (1)$$

where *I* is the number of instances for which the *RPD* (Relative Percentage Deviation) values are obtained (i.e., the testbed size). RPD_{ij} is the relative percentage deviation obtained by algorithm *j* when applied to instance *i* and is typically calculated as follows:

$$RPD_{i,j} = \frac{C_{\max,i,j} - Best_i}{Best_i} \cdot 100 \quad (2)$$

where $C_{\max,i,j}$ is the makespan of the algorithm *j* in instance *i* and $Best_i$ is the upper bound (best solution known) for that instance. When the raw makespan value for each instance is given in the paper, the *ARPD* is computed again using (2) and the best known value for those instances (see on-line materials) in order to have a common reference. Otherwise, the *ARPD* values of the paper are reported. Note that these papers could have used different upper bounds ($Best_i$) and the values are therefore only approximations. For papers using the same upper bounds as in Taillard (1993), a factor of 0.565 is added to correct the ARPDs. This value is the difference in *ARPD* between the actual upper bounds and the upper bounds of Taillard (1993).

The eighth and ninth columns indicate the programming languages used for coding the algorithms as well as the raw speed of the processors used for the evaluation. Finally, the average CPU time on Taillard's instances as a function of the size of the problem (i.e. *n* and *m*) is calculated, when possible, in the last column in order to analyse the CPU requirements of the algorithms. This value is expressed in terms of t_j for metaheuristic *j*, a variable traditionally used in the literature to measure its stopping criterion as $n \cdot m \cdot t_j/2$ milliseconds (see e.g. Ruiz & Stützle (2007)). When t_j is not indicated and/or other stopping criteria are used, t_j is calculated as follows:

$$t_j = \sum_{\forall i} t_{ij}$$

and

$$t_{ij} = \frac{2 \cdot CPU_{ij}}{n_i \cdot m_i}$$

where CPU_{ij} is the CPU time in milliseconds required by algorithm *j* in instance *i*. n_i and m_i and the number of jobs and machines in instance *i*. Therefore, t_{ij} balances the CPU time with the size of the problem, and t_j – average of t_{ij} – can be seen as an indicator of the average CPU time requirements of an algorithm, since, given an instance, n_i and m_i are constants for different algorithms.

For clarity, when a paper proposes several metaheuristics, these methods are included in the table as long as they are used as reference metaheuristics in other papers. Otherwise, only the best one among the reported results is selected. The language used to code the algorithms has been included in the table since languages can result in much greater differences than those caused by the use of varying computer characteristics. This is a well studied phenomenon, mainly in the computer science field. A deep comparison of this effect can be found in Nanz and Furia (2015).

In view of the tables, the need for a new review and computational evaluation – already discussed in Section 1 – is confirmed, as there are very few papers whose methods are directly compared with the state-of-the-art algorithms (i.e., the IG_RS_{LS} by Ruiz & Stützle (2007)). Most of them are directly compared with metaheuristics of the same type (i.e. papers proposing PSO metaheuristics are compared with other PSO metaheuristics). Additionally, among all analysed metaheuristics, only 9 papers (less than 10%) explicitly state that the metaheuristics are compared using the same conditions. Finally, there is no homogeneity in the set of instances used to compare the methods. Most metaheuristics (56) are tested in Taillard's benchmark, although only 20 of these use all 120 instances of the testbed. The rest of the testbeds used were mainly Reeves' (23 times) and Carlier's (15 times). From this literature review, the current state-of-the-art is far from easy to identify.

3. Computational evaluation

In this section, the procedure followed to evaluate the algorithms is described. A total of 31 algorithms have been recoded in C# (using Microsoft Visual Studio Professional 2013 and the .NET Framework 4.5.1). All experiments have been carried out on a computational cluster formed by 30 blade servers. Each server contains two Intel XEON E5420 processors running at 2.5 gigahertz and 16 gigabytes of RAM memory. However, the specific tests are performed on virtual machines running on this cluster. Each virtual machine runs Microsoft Windows 7 64 bit operating system and has one virtual processor and 2 gigabytes of RAM. Several benchmarks have been used (see e.g., Carlier (1978); Demirkol et al. (1998); Heller (1960); Reeves (1995); Taillard (1993); Watson et al. (2002)) in the literature to perform comparisons between algorithms. Among them, the most extended one is the benchmark from Taillard (1993) which includes 120 instances with 12 different sizes of instance combining the values $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, with 10 instances for each size. More recently, Vallada et al. (2015) proposed a more exhaustive symmetric benchmark which contains 240 instances (denoted as VRF instances) for all the combinations of parameters $n \in \{100, 200, 300, 400, 500, 600, 700, 800\}$ and $m \in \{20, 40, 60\}$. This benchmark was shown to have more discriminant power than that of Taillard (1993). In this paper, both benchmarks are used to compare the algorithms.

When comparing heuristics, there is a trade-off between the quality of the solution and the computational effort required. Traditionally, the quality of the solution is measured by the *ARPD* – defined as in Eq. (1)–, and the computational effort by the Average CPU time (denoted as *ACPU*) which can be defined as follows:

$$ACPU_j = \frac{\sum_{\forall i} CPU_{i,j}}{I} \quad (3)$$

where, as usual, *I* is the number of instances and $CPU_{i,j}$ is the CPU time (in seconds) required by algorithm *j* in instance *i*.

Since each constructive heuristic has a different value of *ACPU* and *ARPD*, assessing the efficiency of the heuristics is not trivial. In a similar problem, Fernandez-Viagas and Framinan (2015a) established that the use of the previous indicators presents several problems since *ARPD* is a dimensionless indicator and *ACPU* is heavily instance- and instance-size-dependent (e.g. the last ten largest instances of the $Fm|prmu|\Sigma C_j$ problem contribute more than 88% to the average CPU time indicator). In order to avoid these problems, Fernandez-Viagas and Framinan (2015a) defined $ARPT'_j$ as the average relative percentage time consumed by algorithm *j* as follows:

$$ARPT'_j = \frac{\sum_{\forall i} RPT_{i,j}}{I} \quad (4)$$

where $RPT_{i,j}$ (relative percentage computation time obtained by algorithm *i* for instance *j*) is calculated as

$$RPT_{i,j} = \frac{CPU_{i,j} - ACT_i}{ACT_i} \quad (5)$$

and ACT_i can be computed as

$$ACT_i = \frac{\sum_{\forall i} CPU_{i,j}}{J} \quad (6)$$

where J is the number of considered heuristics.

Despite its dimensionless nature, $ARPT'$ can be higher than or equal to -1 and therefore, it can yield negative values. As a result, we suggest a small modification of $ARPT'$, denoted as $ARPT$ in the following in order to be able to show graphics in logarithmic scale ($ARPT > 0$). More specifically, $ARPT$ is defined as follows:

$$ARPT_j = ARPT'_j + 1 \quad (7)$$

ARPT represents, on average for all instances, the number of times that the CPU time of each heuristic is larger than the mean CPU time across all heuristics. Values close to 0 indicate very fast heuristics (as compared with the rest of heuristics) while high values indicate slow heuristics.

In this paper, we use the *ARPD* indicator to measure the quality of the solutions and both *ARPT* and *ACPU* indicators to measure the computational effort of the algorithms. Note that, despite the problems when using the *ACPU* indicator to compare algorithms, it is included in the evaluation in order for one to be able to reproduce the original comparisons of the authors since all reviewed and implemented heuristics consider the *ACPU* indicator. By means of these two indicators, let us denote a method as efficient in terms of *ARPT* (*ACPU*) when there is no other method with both less *ARPD* and less *ARPT* (*ACPU*).

Regarding the algorithms implemented in the computational evaluation, numerous algorithms have been proposed in the literature since the last computational evaluation of (Ruiz & Maroto, 2005). As a matter of fact, the number of metaheuristics is staggering and new proposals do not cease to appear. Therefore, only a selected number of them have been implemented with a cutoff date of December 2014.

Among the heuristics of Section 2.1, the FRB1 heuristic has been statistically improved by several heuristics (e.g., $FRB4_6$, $FRB4_8$) in the same paper. Additionally, the tie-breaking mechanisms of (Dong et al., 2008), (Kalczynski & Kamburowski, 2007), (Kalczynski & Kamburowski, 2008) as well as the original one of (Nawaz et al., 1983) are statistically outperformed by the tie-breaking mechanism proposed by (Fernandez-Viagas & Framinan, 2014) and therefore, heuristics NEHD, NEH1, NEHKK1 and NEH are removed from the analysis. A total of 19 remaining heuristics, are reimplemented here under the same conditions. They are: RAER, RAER-di, KKER, KKER-di, NEHR, NEHR-di, NEMR, NEMR-di, NEH-di, NEH1-di, NEHKK1-di, NEHKK2, NEHD-di, NEHFF, CL_{WTS}, FRB2, FRB3, $FRB4_k$ ($k \in \{2, 4, 6, 8, 10, 12\}$) and FRB5 (indicated in bold in Table 1). Note that, although the recent heuristic NEHI was initially discarded due to the fact that it was available online after December 2014, it also seems to be clearly inefficient according to the *ARPD* and average computational times (around 25 times greater than the original NEH) shown in that paper (as compared to $FRB4_{10}$ or $FRB4_{12}$ for example). Note that there are two possible interpretations of *RCT*, the idle-time- based tie-breaking mechanism proposed by Ribas et al. (2010). The authors state that this mechanism can be implemented in $O(n^2m^2)$. However, as explained in Fernandez-Viagas and Framinan (2014), it can be implemented in $O(n^3m)$ if the idle time between jobs is calculated only for the ties. Thereby, the complexity is $O(E \cdot n^2m)$ due to the need to evaluate a complete sequence for each iteration E times. Clearly, since the maximum number of tie-breaks is the number of jobs in the partial sequence, the complexity of this interpretation is $O(n^3m)$. In this paper, this latter interpretation is employed as it yields a lower computational effort for the benchmark of (Taillard, 1993), i.e. the constant affecting the complexity of the original interpretation is higher than that of the second one for each instance of the testbed.

Regarding metaheuristics, the decision about which ones to select is not trivial due to the large amount of existing methods. More precisely, only algorithms fulfilling the two following requirements are considered:

- $ARPD < 0.4$ (on T1 or T2, see Table 2) or
- $ARPD < 0.6$ and t parameter ≤ 90 (on T1).

In other words, we are demanding that for a metaheuristic to be selected it either has to have a good solution quality ($ARPD < 0.4$), or a reasonable solution quality in short-medium computational times ($ARPD < 0.6$ and t parameter ≤ 90). 12 metaheuristics fulfil these requirements: EXTS by Solimanpur, Vrat, and Shankar (2004); HGA_RMA by Ruiz, Maroto, and Alcaraz (2006); MSSA by Nowicki and Smutnicki (2006); IG_RS_{LS} by Ruiz and Stützle (2007); IG_{RIS} by Pan, Tasgetiren, and Liang (2008); DDE_{RLS} by Pan et al. (2008); 3XTS by Eksioglu, Eksioglu, and Jain (2008); EDA_{ACS} by Tzeng and Chen (2012); PSO by Zhang and Wu (2014); $IG\_RS_{LS}(TB_{FF})$ by Fernandez-Viagas and Framinan (2014); $IG_{RIS}(TB_{FF})$ by Fernandez-Viagas and Framinan (2014). Among them, EXTS and HGA_RMA, are discarded since they are outperformed in statistically and/or sound comparisons by Eksioglu et al. (2008) and Ruiz and Stützle (2007), respectively. Additionally, the H-CPSO algorithm by Jarboui, Ibrahim, Siarry, and Rebai (2008) has been implemented due to its promising results despite being outperformed by Pan et al. (2008) under different stopping criteria and conditions. Metaheuristic HCS by Li and Yin (2013a) has also been included in the comparison since the *ARPD* is very close to 0.4 and has not been shown to be outperformed by any other metaheuristic. Finally, we include the TSAB tabu search algorithm by Nowicki and Smutnicki (1996) in the comparisons, given its excellent performance and the fact that it was not included in the last computational evaluation by Ruiz and Maroto (2005). The reason behind this omission is explained in (Ruiz & Stützle, 2007) which is mainly the difficulty in reproducing the results of the TSAB algorithm. As a matter of fact, we had to contact the authors of the method, which kindly provided the source code used for checking our reimplementation. Hence, a total of 12 metaheuristics have been chosen (indicated in bold in Tables 2 and 3).

Note that all selected algorithms are implemented and tested under the same conditions which means:

- Using the same computer. This means same processor speed, bus speed, memory speed and size, etc.
- Using the same programming language.
- Using the same operating system.
- Using the same libraries and common functions.
- Using the same stopping criteria for the metaheuristics.

When reimplementing the algorithms, doubts relating to the implementation were transmitted to the corresponding authors of the papers. All questions were successfully answered by the authors with the exception of (Zhang & Wu, 2014), where no answer was received after several tries. Other specifics considered in order to carry out a fair comparison of the algorithms are the following:

- The order of the instances was randomly chosen in the experiments to avoid systematic errors in the tests.
- The algorithms to be run in each instance are similarly randomized.
- For each instance, ten independent runs were performed for each heuristic to better fit the required CPU time (the average CPU time is kept).
- For each instance, five independent runs were carried out for each metaheuristic keeping the average values.

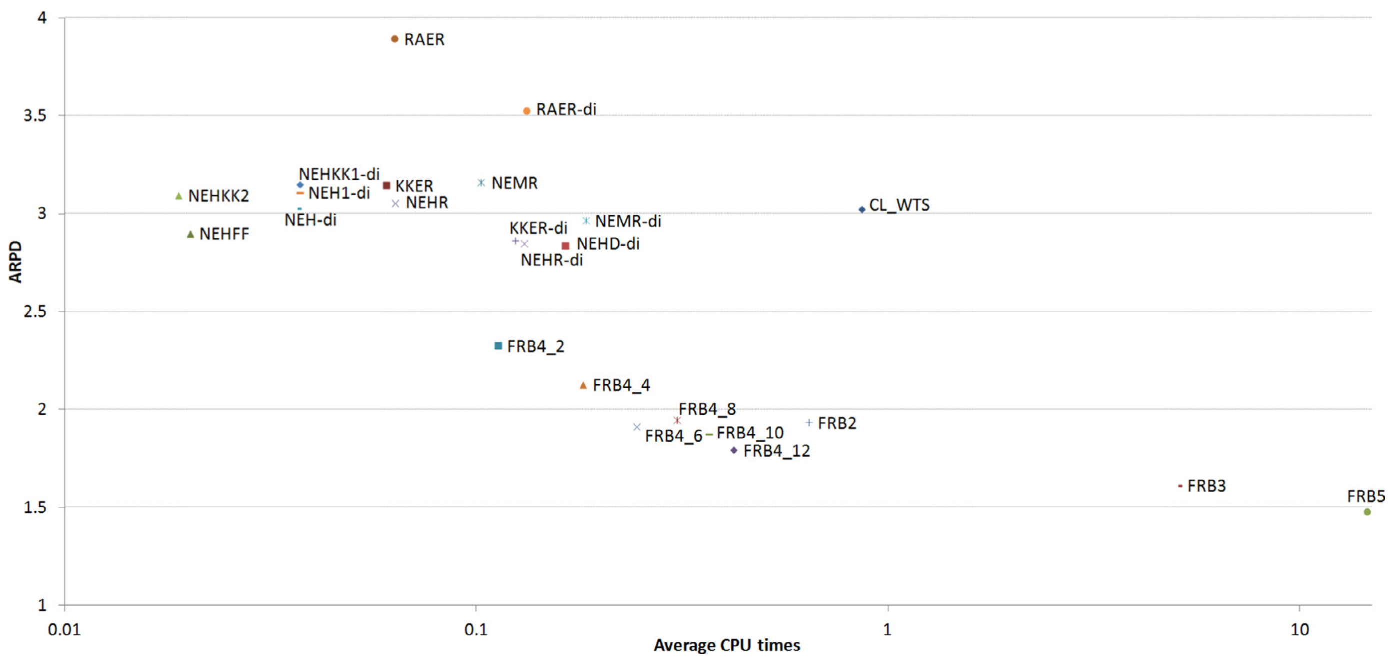

Fig. 1. *ARPD* vs. *ACPU* of heuristics in logarithmic scale on Taillard's instances.

Note that even recently published, this computational evaluation follows many of the practices highlighted in (Kendall et al., 2016). The results of these experiments – that have required a total CPU time effort of 393.03 days – are presented in the next section.

4. Computational results

4.1. Constructive and improvement heuristics

The 19 heuristics implemented in this evaluation are first compared under the classic benchmark set of Taillard with 120 instances. The detailed results in terms of *ARPD*, *ACPU* and *ARPD*, ordered by problem size, are presented as on-line materials. The overall results are summarised in Table 4. The second, third and fourth columns represent the *ARPD*, *ACPU* and *ARPT* values for each algorithm in the set of instances of Taillard. *ARPD* values range from 3.89 (RAER heuristic) to 1.48 (FRB5 improvement heuristic) while *ARPT* values range from 0.02 to 7.23. Results are graphically shown in Figs. 1 and 2 where the y-axis represents the *ARPD* for each heuristic and x-axis, respectively, represents *ACPU* and *ARPT* in logarithmic scale. Although results obtained for the different time indicators are, in general, similar, there are also differences in the performance of the heuristics. Therefore, considering *ACPU* as a measure of the computational effort as compared to *ARPT*, $FRB4_2$ is faster than KKER-di, NEHR-di and RAER-di in addition to the CL_{WTS} being slower than the FRB2 heuristic. According to indicators *ARPD* and *ARPT*, the efficient heuristics are NEHKK2, NEHFF, NEHR-di (this last one would not be efficient considering *ARPD* and *ACPU*), $FRB4_2$, $FRB4_4$, $FRB4_6$, $FRB4_{10}$, $FRB4_{12}$, FRB3 and FRB5 (shown with a black circle in Fig. 2). To be able to compare heuristics with different stopping criteria, they are grouped into clusters with similar *ARPT* values (see Fig. 2). Then, the heuristics of each cluster are compared with the best heuristic in terms of *ARPD* of that cluster, i.e. NEHFF, $FRB4_2$, $FRB4_4$, $FRB4_6$ and $FRB4_{12}$, respectively, for clusters 1, 2, 3, 4, and 5. The hypotheses to statistically check the efficiency of the heuristics are shown in Table 5, ordered by these clusters of heuristics. Since each heuristic is based on the original NEH algorithm and the same set of instances is used, the hypotheses of independence (denoted by $H_{0,\,t,\,i}$) of the random variables (*RDI*) can be rejected (see third and fourth columns in Table 5). The non-parametric Friedman two-way analysis of variance for paired samples is used to check the statistical significance of the differences among the heuristics in each cluster (being the null hypothesis –denoted $H_{0,\,t,\,f}$– that there are no differences). Additionally, to establish the significance of the differences between the best heuristic of the cluster and the rest, the non-parametric Wilcoxon signed-rank test in a post-hoc analysis is employed (being $H_{0,t,w}$ the corresponding null hypothesis). Results are shown in Table 5. Assuming a level of confidence of 0.95, several $H_{0,t,w}$ null hypotheses of the NEHFF heuristic (Cluster 1) have not been rejected (see e.g., NEHFF vs NEHR or NEHFF vs NEH-di).

Table 4
Summary of heuristics.

Algorithm	Taillard			VRF		
	ARPD	ACPU	ARPT	ARPD	ACPU	ARPT
NEHKK2	3.09	0.02	0.12	3.21	0.47	0.02
NEHFF	2.90	0.02	0.13	2.95	0.46	0.02
NEH-di	3.03	0.04	0.20	3.18	0.91	0.04
NEH1-di	3.11	0.04	0.20	3.15	0.91	0.04
NEHKK1-di	3.15	0.04	0.20	3.19	0.93	0.04
RAER	3.89	0.06	0.20	3.46	0.88	0.04
NEHR	3.05	0.06	0.21	3.16	0.93	0.04
KKER	3.15	0.06	0.21	3.15	0.93	0.04
NEMR	3.16	0.10	0.31	3.22	1.64	0.07
RAER-di	3.53	0.13	0.40	3.33	1.71	0.07
NEHR-di	2.85	0.13	0.40	3.02	1.82	0.07
KKER-di	2.86	0.12	0.42	3.00	1.79	0.07
NEHD-di	2.84	0.16	0.48	2.86	2.06	0.08
$FRB4_2$	2.33	0.11	0.48	2.57	2.81	0.13
NEMR-di	2.97	0.18	0.52	3.05	2.53	0.10
$FRB4_4$	2.13	0.18	0.68	2.31	4.65	0.20
CL_{WTS}	3.02	0.86	0.73	3.11	26.63	0.68
$FRB4_6$	1.91	0.25	0.89	2.17	6.42	0.28
$FRB4_8$	1.95	0.31	1.06	2.07	8.09	0.35
$FRB4_{10}$	1.87	0.37	1.20	1.97	9.87	0.43
$FRB4_{12}$	1.79	0.42	1.34	1.94	11.42	0.49
FRB2	1.93	0.64	1.68	1.74	37.97	1.40
FRB3	1.61	5.08	3.61	1.32	198.31	4.34
FRB5	1.48	14.59	7.23	1.04	753.56	14.36

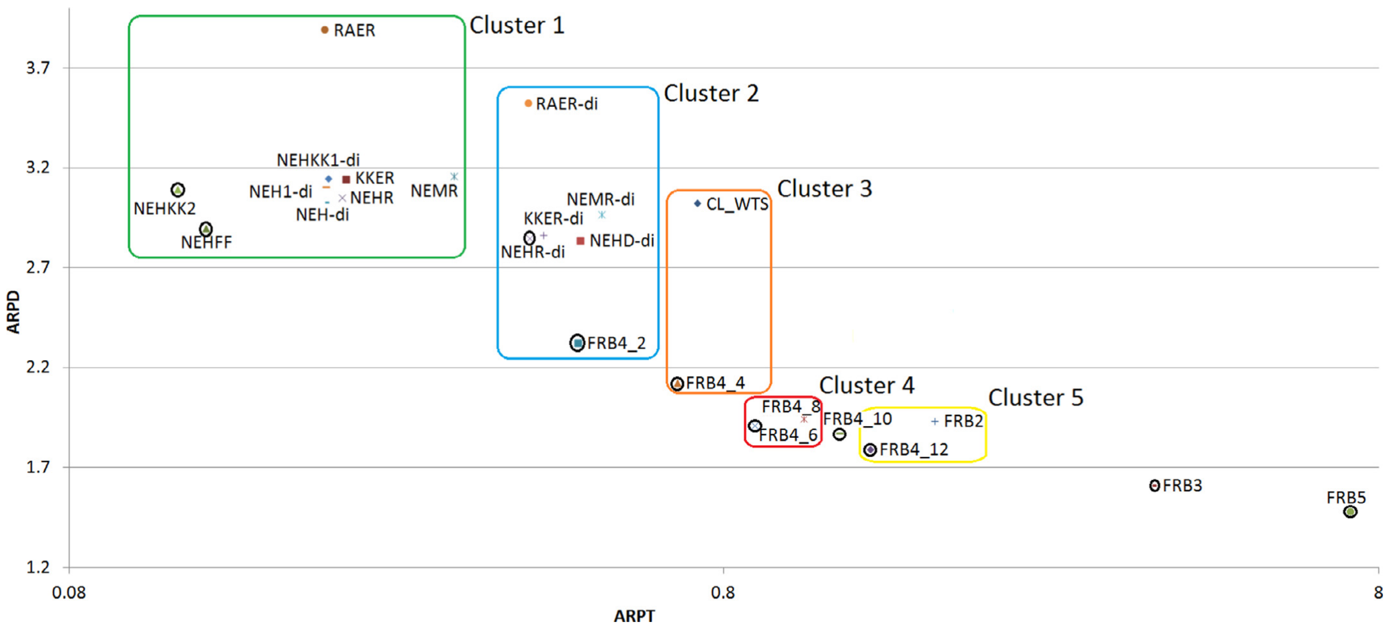

Fig. 2. *ARPD* vs. *ARPT* of heuristics in logarithmic scale on Taillard's instances.

Table 5
Hypotheses, analysis of dependence and Friedman two-way analysis on Taillard's instances.

Clusters	Comparison	Analysis of dependence		Friedman	Wilcoxon
		correlation	Sig.	Sig.	Sig.
Cluster 1 (green)	NEHFF vs NEHKK2	0.891	0.000	0.000	0.015
	NEHFF vs NEH-di	0.923	0.000		0.054
	NEHFF vs NEHKK1-di	0.895	0.000		0.001
	NEHFF vs NEHR	0.893	0.000		0.055
	NEHFF vs NEH1-di	0.910	0.000		0.021
	NEHFF vs KKER	0.884	0.000		0.010
	NEHFF vs NEMR	0.869	0.000		0.006
	NEHFF vs RAER	0.830	0.000		0.000
Cluster 2 (blue)	$FRB4_2$ vs RAER-di	0.842	0.000	0.000	0.000
	$FRB4_2$ vs NEHR-di	0.880	0.000		0.000
	$FRB4_2$ vs KKER-di	0.877	0.000		0.000
	$FRB4_2$ vs NEHD-di	0.860	0.000		0.000
	$FRB4_2$ vs NEMR-di	0.864	0.000		0.000
Cluster 3 (orange)	$FRB4_4$ vs CL_{WTS}	0.868	0.000	0.000	0.000
Cluster 4 (red)	$FRB4_6$ vs $FRB4_8$	0.937	0.000	0.604	–
Cluster 5 (yellow)	$FRB4_{12}$ vs FRB2	0.927	0.000	0.107	–

Additionally, there is not enough statistical evidence to state that $FRB4_6$ and $FRB4_{12}$ outperform $FRB4_8$ and FRB2, respectively.

A similar Pareto set is found when the heuristics are compared under the new set of instances VRF of (Vallada et al., 2015). Average results are shown in Table 4. The last three columns represent the *ARPD*, *ACPU* and *ARPT* of each heuristic in that set of instances. Clearly, heuristics of complexity $O(n^3m)$ (CL_{WTS}, FRB2, FRB3 and FRB5) need proportionally more computational effort since this set of instances considers higher values of n and m than in Taillard's instances. This increase in the computational effort also results in a decrease in the *ARPD* of the heuristics with the exception of CL_{WTS}. Results are graphically shown in Fig. 3 comparing *ARPD* vs. *ACPU*, and in Fig. 4 comparing *ARPD* vs. *ARPT*. In terms of *ARPD* and *ARPT*, efficient heuristics are shown with a black circle in Fig. 4. Note that regarding the NEH-based heuristics of (Ribas et al., 2010) with direct and inverse approach, the best *ARPD* is now found by the NEHD-di heuristic instead of the NEHR-di. In order to compare the heuristics, we group them according to their *ARPT* (see Fig. 4) and perform the same Friedman two-way analysis of variance to identify the differences among the heuristics in each cluster (being $H_{0,v,f}$ the corresponding null hypothesis), since hypotheses of independence ($H_{0,v,i}$) can be rejected again). In a post-hoc analysis, a non-parametric Wilcoxon signed-rank test is applied to establish the statistical significance of the differences between the best heuristic of each cluster ($H_{0,v,w}$ being the null hypothesis). Note that heuristics $FRB4_k$ are not compared together as they are the same heuristic with a different input parameter. Results are shown in Table 6. Each p-value is 0.000 and all $H_{0,v,f}$ and $H_{0,v,w}$ hypotheses are rejected. Thus, according to *ARPD* and *ARPT*, there is no statistical reason to affirm that the NEHFF, $FRB4_k$, FRB2, FRB3, FRB5 heuristics are not efficient within each cluster.

4.2. Metaheuristics

In Section 3, 12 metaheuristics were defined as the most promising according to the results shown in their papers. In this section, these metaheuristics are compared using the sets of instances of (Taillard, 1993) and (Vallada et al., 2015). Each metaheuristic is stopped using the same stopping criterion based on CPU time. More specifically, three different stopping criteria are applied, $t \cdot n \cdot m/2$ milliseconds with $t \in \{30, 60, 90\}$, which depends on the number of jobs and machines. Results are shown in Table 7.

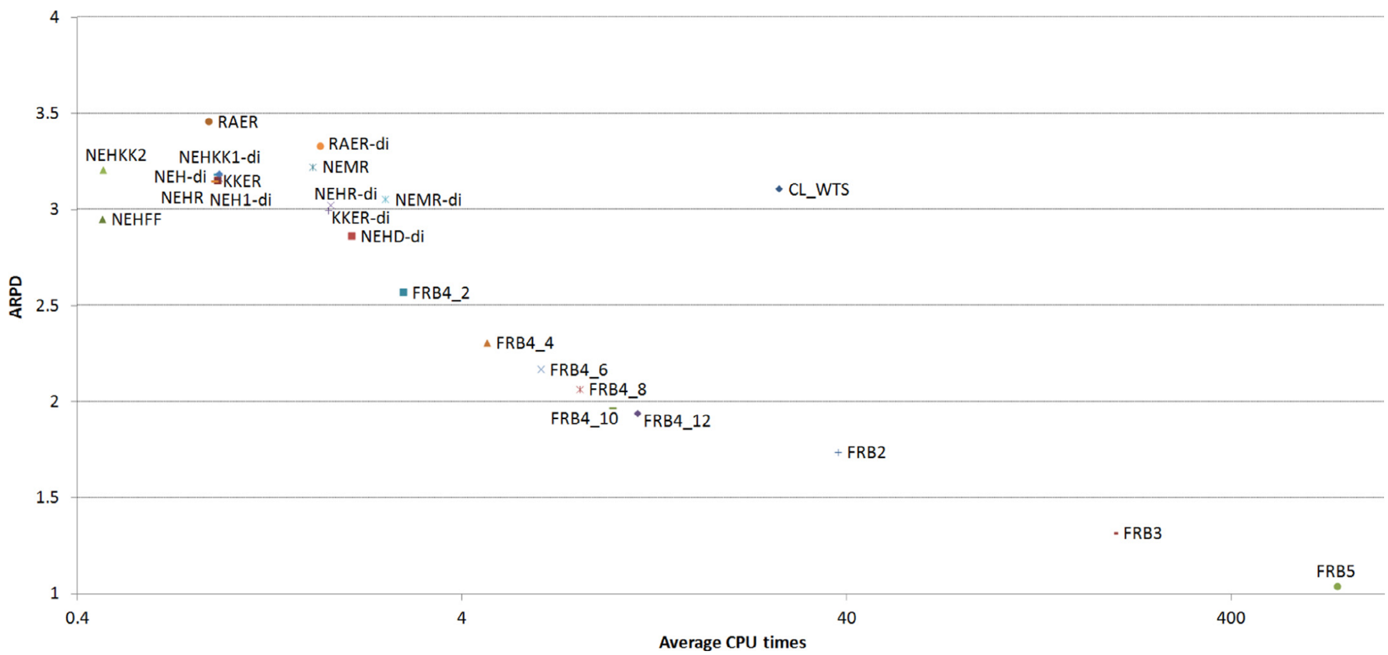

Fig. 3. *ARPD* vs. *ACPU* of heuristics in logarithmic scale on VRF instances.

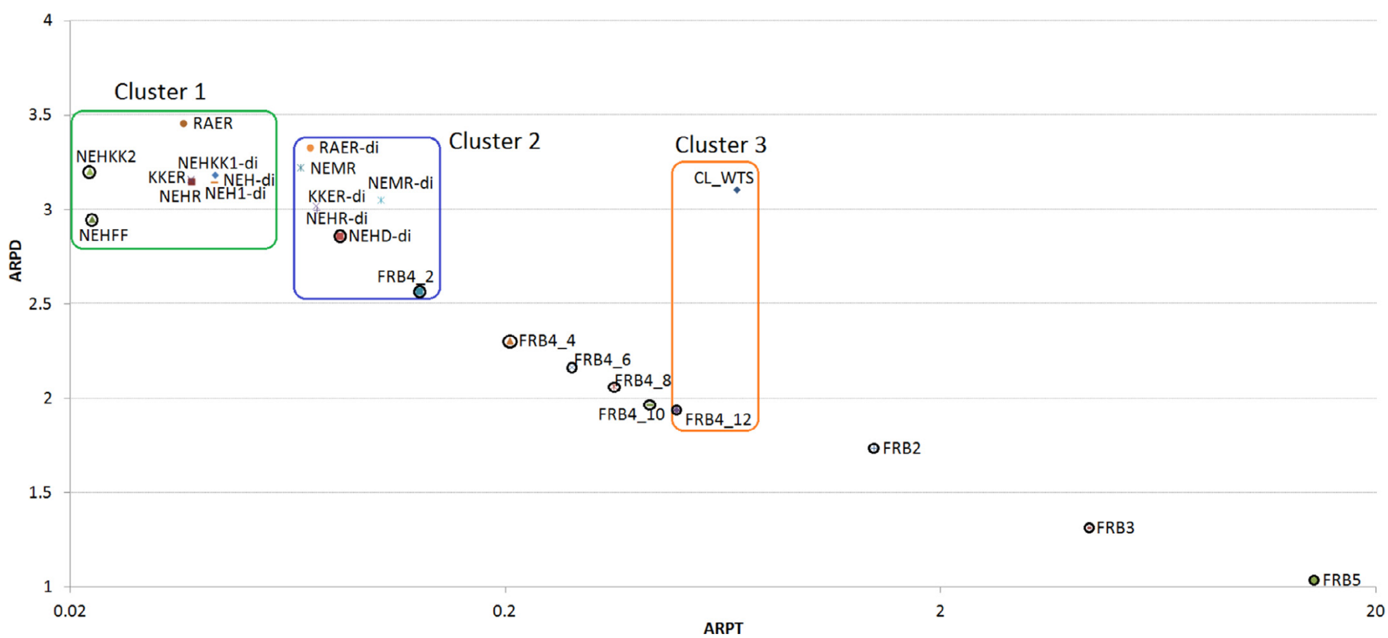

Fig. 4. *ARPD* vs. *ARPT* of heuristics in logarithmic scale on VRF instances.

For both sets of instances, the best metaheuristics are those based on the Iterated Greedy (IG_RS_{LS}) proposed by Ruiz and Stützle (2007), see the results found by IG_RS_{LS}, IG_{RIS}, $IG\_RS_{LS}(TB_{FF})$ and $IG_{RIS}(TB_{FF})$ for example. These results are also confirmed by the DDE_{RLS}, a discrete differential evolution algorithm which uses similar phases.

Regarding Taillard's instances, the *ARPDs* of Iterated Greedy metaheuristics for $t = 90$ is between 0.28 and 0.38 which clearly outperforms non IG-based metaheuristics (the *ARPDs* of 3XTS, H-CPSO, HCS and PSO are, respectively, 1.24, 0.70, 1.35 and 0.84 for $t = 90$). The best *ARPD* value is obtained by $IG\_RS_{LS}(TB_{FF})$ proposed by Fernandez-Viagas and Framinan (2014), with 0.37, 0.32 and 0.37 for $t = 30$, $t = 60$ and $t = 90$ on Taillard's instances, respectively. Let us highlight the fast convergence behaviour of $IG\_RS_{LS}(TB_{FF})$ where the *ARPD* obtained for $t = 30$ is lower than or equal to every other metaheuristic for $t = 90$. Metaheuristics are compared with $IG\_RS_{LS}(TB_{FF})$ using the non-parametric Wilcoxon signed-rank test (see Table 8). Note that each *p*-value on the Taillard's instances is less than or equal to 0.003 regardless the value of *t*.

Regarding the VRF instances, the superiority of the IG-based algorithms is more clear, as VRF instances include a wider range of values of *n* and *m*. Thereby, the differences between the *ARPD* values of the metaheuristics greatly increase with respect to the $IG\_RS_{LS}(TB_{FF})$ metaheuristic (see the difference of *ARPD* between 3XTS and $IG\_RS_{LS}(TB_{FF})$ is 0.96 on Taillard's instances and 2.10 on VRF instances for $t = 90$ for example). Statistical significance has been found for all metaheuristics (maximum *p*- value equal to 0.000) with the exception of $IG_{RIS}(TB_{FF})$ (see Table 8). In view of

Table 6
Hypotheses, analysis of dependence and Friedman two-way analysis on VRF instances.

	Comparison	Analysis of Dependence		Friedman	Wilcoxon
		correlation	Sig.	Sig.	Sig.
Cluster 1 (green)	NEHFF vs NEHKK2	0.950	0.000	0.000	0.000
	NEHFF vs NEH-di	0.954	0.000		0.000
	NEHFF vs NEHKK1-di	0.952	0.000		0.000
	NEHFF vs NEHR	0.946	0.000		0.000
	NEHFF vs NEH1-di	0.939	0.000		0.000
	NEHFF vs KKER	0.952	0.000		0.000
	NEHFF vs RAER	0.945	0.000		0.000
Cluster 2 (blue)	$FRB4_2$ vs NEMR	0.943	0.000	0.000	0.000
	$FRB4_2$ vs RAER-di	0.946	0.000		0.000
	$FRB4_2$ vs NEHR-di	0.958	0.000		0.000
	$FRB4_2$ vs KKER-di	0.953	0.000		0.000
	$FRB4_2$ vs NEHD-di	0.948	0.000		0.000
	$FRB4_2$ vs NEMR-di	0.952	0.000		0.000
Cluster 3 (orange)	$FRB4_{12}$ vs CL_{WTS}	0.942	0.000	0.000	0.000

Table 7
Summary of *ARPDs* of the metaheuristics.

Metaheuristic	Ref.	Taillard			VRF		
		$t = 30$	$t = 60$	$t = 90$	$t = 30$	$t = 60$	$t = 90$
TSAB	Nowicki and Smutnicki (1996)	0.97	0.87	0.84	2.16	1.96	1.85
MSSA	Nowicki and Smutnicki (2006)	1.00	0.91	0.84	2.17	1.96	1.84
IG_RS_{LS}	Ruiz and Stützle (2007)	0.47	0.40	0.37	0.96	0.77	0.67
IG_{RIS}	Pan et al. (2008)	0.49	0.42	0.38	0.85	0.67	0.56
DDE_{RLS}	Pan et al. (2008)	0.52	0.47	0.43	0.92	0.77	0.69
3XTS	Eksioglu et al. (2008)	1.64	1.34	1.24	2.89	2.65	2.47
H-CPSO	Jarboui et al. (2008)	0.84	0.75	0.70	1.65	1.41	1.28
EDA_{ACS}	Tzeng and Chen (2012)	0.60	0.51	0.47	1.43	1.25	1.16
HCS	Li and Yin (2013a)	1.55	1.42	1.35	2.54	2.35	2.27
PSO	Zhang and Wu (2014)	1.09	0.95	0.84	2.51	2.14	1.93
$IG\_RS_{LS}(TB_{FF})$	Fernandez-Viagas and Framinan (2014)	0.37	0.32	0.28	0.60	0.46	0.37
$IG_{RIS}(TB_{FF})$	Fernandez-Viagas and Framinan (2014)	0.42	0.34	0.31	0.61	0.47	0.38

Table 8
Comparison of metaheuristics using Wilcoxon signed-rank tests.

Comparison	Taillard (Sig.)			VRF (Sig.)		
	$t = 30$	$t = 60$	$t = 90$	$t = 30$	$t = 60$	$t = 90$
TSAB vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
MSSA vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
IG_{RIS} vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
IG_RS_{LS} vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
DDE_{RLS} vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
3XTS vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
H-CPSO vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
EDA_{ACS} vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
HCS vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
PSO vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.000	0.000	0.000	0.000	0.000
$IG_{RIS}(TB_{FF})$ vs $IG\_RS_{LS}(TB_{FF})$	0.000	0.003	0.000	0.155	0.220	0.137

the results, although there are many papers proposing metaheuristics, only the Iterated Greedy variants proposed by Fernandez-Viagas and Framinan (2014) statistically outperform IG_RS_{LS} on both Taillard's and VRF instances.

We have already commented that many metaheuristics have been published since the last computational evaluation and review of meheuristics proposed by Ruiz and Maroto (2005) (see Tables 2 and 3) and since the original Iterated Greedy algorithm proposed by Ruiz and Stützle (2007). On one hand, in view of Tables 2 and 3, only 12 metaheuristics have promising results in terms of quality of solutions and computational effort. On the other hand, in view of the results in this section, only the $IG_{RIS}(TB_{FF})$ and the $IG\_RS_{LS}(TB_{FF})$ algorithms are state-of-the-art methods. It follows that many metaheuristics were not state-of-the-art even at the time on their publication, a fact that strongly highlights the need for a review and framework for computational evaluation such as the one proposed in this paper.

4.3. Comparison of heuristics with metaheuristics

Traditionally, researchers have focused either on finding efficient heuristics, or on obtaining the best metaheuristic for the problem. The former are implemented to find a good fast solution and/or a good initial seed sequence for the problem, while the latter are typically implemented to find better solutions using longer CPU times. As a consequence, typically both heuristics and metaheuristics have been separately evaluated and compared. In this section, we analyse both heuristics and metaheuristics together, as there are several heuristics requiring long CPU times and vice versa. Therefore, each heuristic is compared with one of the best metaheuristics, i.e., the iterated greedy $IG\_RS_{LS}(TB_{FF})$. In

Table 9
Comparison between heuristics and the best metaheuristic.

Algorithm	Taillard							VRF						
	Original heuristics			IG_RS$_{LS}$(TB$_{FF}$)			Wilcoxon	Original heuristics			IG_RS$_{LS}$(TB$_{FF}$)			Wilcoxon
	ARPD	*ACPU*	*ARPT*	*ARPD*	*ACPU*	*ARPT*	Sig.	*ARPD*	*ACPU*	*ARPT*	*ARPD*	*ACPU*	*ARPT*	Sig.
NEHKK2	3.09	0.02	0.12	–	–	–	–	3.21	0.47	0.02	–	–	–	–
NEHFF	2.90	0.02	0.13	–	–	–	–	2.95	0.46	0.02	–	–	–	–
NEH-di	3.03	0.04	0.20	2.53	0.06	0.22	0.000	3.18	0.91	0.04	2.55	1.42	0.06	0.000
NEH1-di	3.11	0.04	0.20	2.50	0.06	0.26	0.000	3.15	0.91	0.04	2.55	1.42	0.06	0.000
NEHKK1-di	3.15	0.04	0.20	2.52	0.06	0.23	0.000	3.19	0.93	0.04	2.55	1.47	0.06	0.000
RAER	3.89	0.06	0.20	2.50	0.09	0.27	0.000	3.46	0.88	0.04	2.53	1.61	0.07	0.000
NEHR	3.05	0.06	0.21	2.51	0.08	0.34	0.000	3.16	0.93	0.04	2.53	1.62	0.07	0.000
KKER	3.15	0.06	0.21	2.50	0.08	0.27	0.000	3.15	0.93	0.04	2.53	1.63	0.07	0.000
NEMR	3.16	0.10	0.31	2.39	0.12	0.34	0.000	3.22	1.64	0.07	2.43	2.11	0.09	0.000
RAER-di	3.53	0.13	0.40	2.36	0.15	0.42	0.000	3.33	1.71	0.07	2.44	2.04	0.09	0.000
NEHR-di	2.85	0.13	0.40	2.35	0.15	0.42	0.000	3.02	1.82	0.07	2.44	2.16	0.09	0.000
KKER-di	2.86	0.12	0.42	2.34	0.14	0.47	0.000	3.00	1.79	0.07	2.43	2.12	0.09	0.000
NEHD-di	2.84	0.16	0.48	2.30	0.17	0.50	0.000	2.86	2.06	0.08	2.41	2.42	0.10	0.000
NEMR-di	2.97	0.18	0.52	2.28	0.20	0.49	0.000	3.05	2.53	0.10	2.27	2.90	0.12	0.000
CL$_{WTS}$	3.02	0.86	0.73	2.05	0.84	0.73	0.000	3.11	26.63	0.68	1.60	24.84	0.64	0.000
FRB4$_2$	2.33	0.11	0.48	2.11	0.13	0.53	0.001	2.57	2.81	0.13	2.18	3.36	0.15	0.000
FRB4$_4$	2.13	0.18	0.68	1.98	0.19	0.68	0.008	2.31	4.65	0.20	1.98	5.16	0.22	0.000
FRB4$_6$	1.91	0.25	0.89	1.83	0.24	0.94	0.019	2.17	6.42	0.28	1.88	6.86	0.29	0.000
FRB4$_8$	1.95	0.31	1.06	1.75	0.30	1.15	0.005	2.07	8.09	0.35	1.80	8.47	0.35	0.000
FRB4$_{10}$	1.87	0.37	1.20	1.70	0.34	1.34	0.002	1.97	9.87	0.43	1.74	10.10	0.41	0.000
FRB4$_{12}$	1.79	0.42	1.34	1.67	0.37	1.46	0.026	1.94	11.42	0.49	1.69	11.57	0.47	0.000
FRB2	1.93	0.64	1.68	1.61	0.61	1.68	0.000	1.74	37.97	1.40	1.39	35.29	1.35	0.000
FRB3	1.61	5.08	3.61	1.40	5.06	3.76	0.000	1.32	198.31	4.34	1.11	197.65	4.32	0.000
FRB5	1.48	14.59	7.23	1.21	14.58	7.38	0.000	1.04	753.56	14.36	0.82	753.03	14.34	0.000

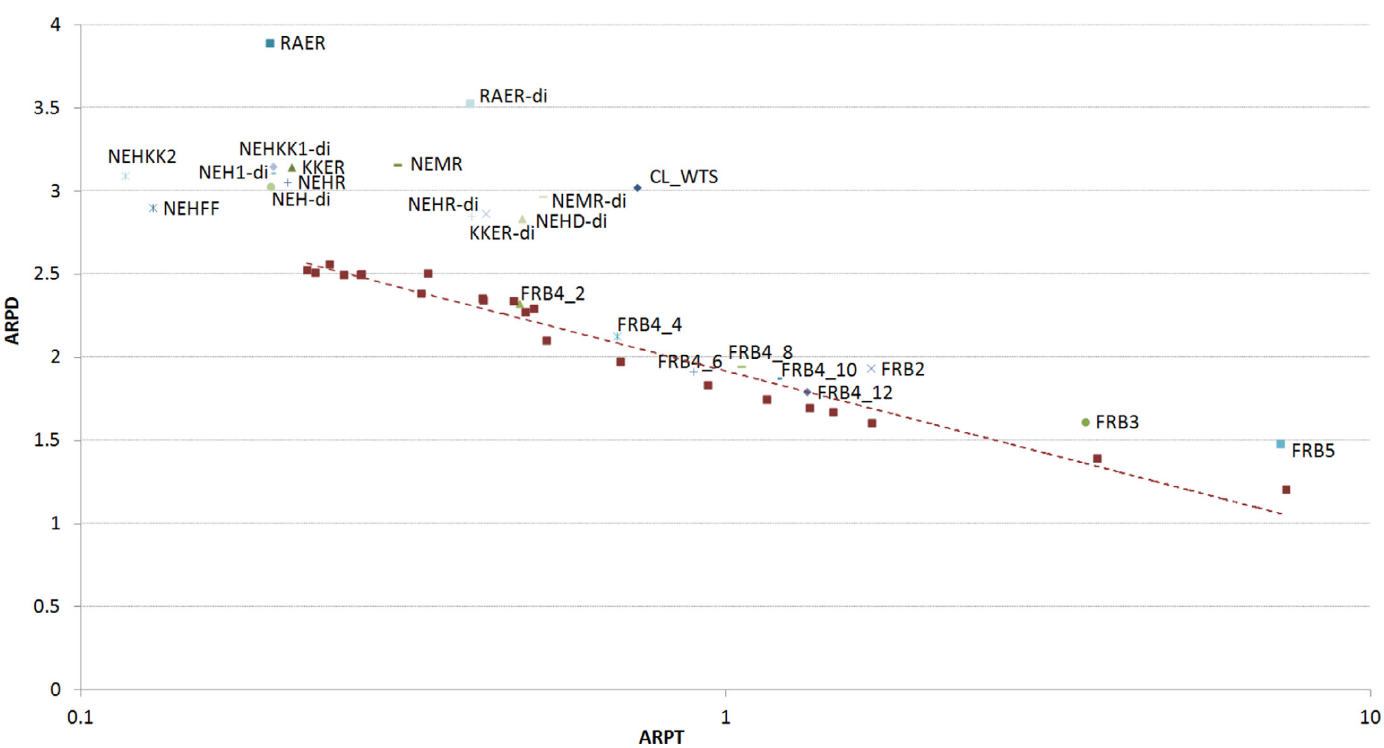

Fig. 5. Heuristics vs. IG_RS$_{LS}$(TB$_{FF}$) on the set of instances of Taillard (1993). *X*-axis (variable *ARPT*) is shown in logarithmic scale.

order to have a fair comparison, the metaheuristic is stopped at the CPU time used by each heuristic. These comparisons are performed using the sets of instances of (Taillard, 1993) and (Vallada et al., 2015). A summary of the results is shown in Table 9 as well as in Figs. 5 and 6 for these benchmarks, respectively, where the dotted lines represent logarithmic trend lines for the heuristics and the red squares represent all values obtained by IG_RS$_{LS}$(TB$_{FF}$). Note that IG_RS$_{LS}$(TB$_{FF}$) starts with the sequence obtained by NEHFF and therefore, NEHKK2 and NEHFF are not included in the comparison as they need shorter CPU times. For all other heuristics, the metaheuristic outperforms them in terms of *ARPD*. All compared heuristics are outperformed by IG_RS$_{LS}$(TB$_{FF}$), especially when compared on the VRF instances. The statistical significance of these comparisons is established by means of the non-parametric Wilcoxon signed-rank test since the normality and homoscedasticity assumptions are not fulfilled. Note that statistical significances are found for each comparison on the Taillard instances, even against the heuristics proposed by Rad et al. (2009) which

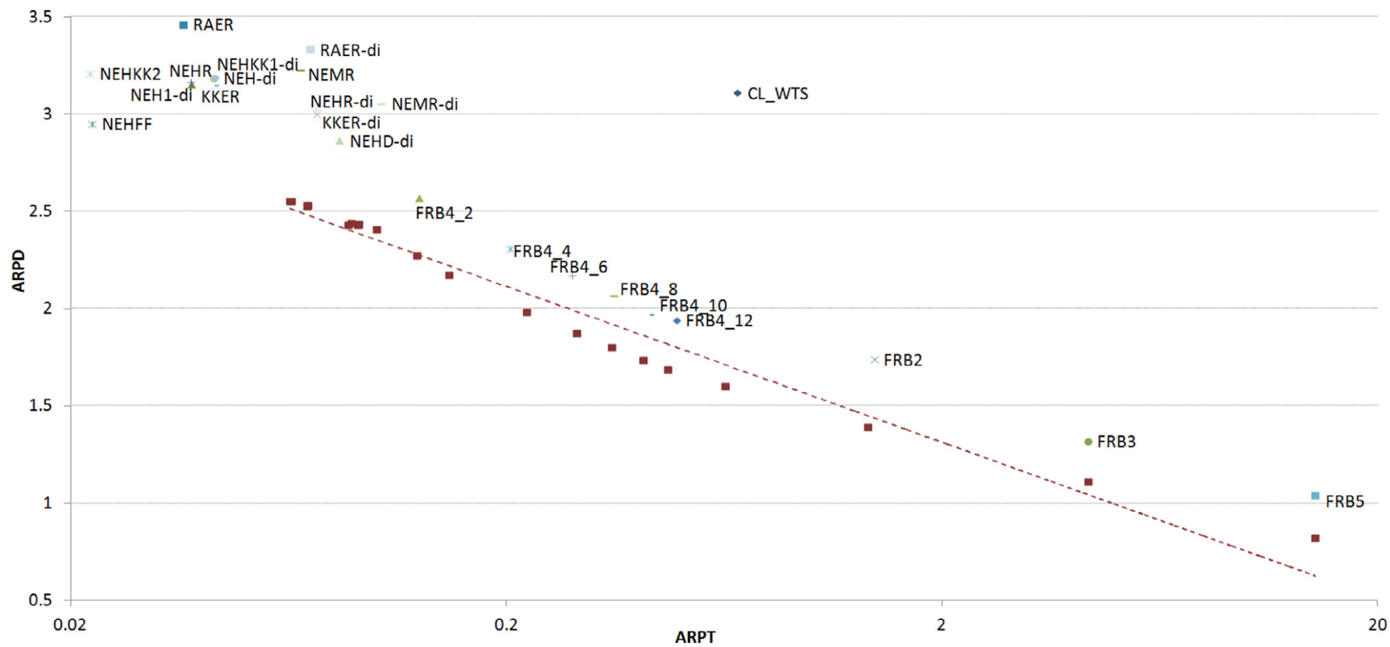

Fig. 6. Heuristics vs. $IG\_RS_{LS}(TB_{FF})$ on the set of instances of Vallada et al. (2015). *X*-axis (variable *ARPT*) is shown in logarithmic scale.

have *ARPD* values similar to or even better than those obtained by $IG\_RS_{LS}(TB_{FF})$ for several problem sizes. Similarly, each corresponding null hypothesis is rejected on VRF instances, 0.001 being the highest *p* value. This section highlights the exceptional performance of IG-based algorithms for short periods of time and also serves to classify $IG\_RS_{LS}(TB_{FF})$ as a state-of-the-art method for constructive and improvement heuristics.

5. Conclusions

Since the last reviews in 2005, a large number of heuristics and metaheuristics have been proposed for the permutation flowshop scheduling problem to minimize makespan. Most of them are compared with other non-efficient algorithms and/or under uncomparable conditions. Thus, it was not clear which algorithms were state-of-the-art. In this paper, an exhaustive review and evaluation of algorithms for the permutation flowshop is proposed, with special attention being paid to conducting a fair comparison of algorithms. The most promising ones, i.e., a total of 31 algorithms (19 constructive heuristics and 12 metaheuristics), have been implemented and compared under the same conditions. The comparisons have been done using the benchmarks of (Taillard, 1993) and (Vallada et al., 2015). On one hand, the metaheuristics are compared under three different stopping criteria to analyse the evolution of the each algorithm with the computational effort. On the other hand, the comparison of (constructive and improvement) heuristics has been performed using two relative indicators to measure the quality of the solution and the computational effort in order to identify the efficient ones. Statistical analyses of the quality of the solutions have been carried out to study the efficiency of the heuristics as well as to compare the metaheuristics. Additionally, each heuristic has been compared with the best metaheuristic under the stopping criterion of the heuristic to analyse tentative best seed sequences for the metaheuristics. Therefore, we believe that this paper may represent a starting point for future researchers who attempt to propose new algorithms for the permutation flowshop scheduling problem with makespan objective.

Notice that all analysed algorithms have been completely recoded. The authors later contacted the corresponding authors of many papers in order to avoid different interpretations in their description of the algorithms. It is worth highlighting that sometimes the great differences in the quality of the solutions are due to the different interpretations of the algorithms. Small variations in some algorithms have even resulted in greater differences than, for example, completely changing the algorithm. To ensure the repeatability and the reproducibility of the algorithms, we consider that at the least a clear pseudo code should be included in the papers, if not the publication of the full source codes on-line, as recommended by the Good Laboratory Practice for Optimization Research (GLP4OPT) practices, recently published by Kendall et al. (2016).

Among all coded metaheuristics, algorithms based in the IG method of (Ruiz & Stützle, 2007) have been clearly identified as the most efficient metaheuristics for the problem. This fact is further confirmed since other well-performing metaheuristics also incorporate some part of the IG algorithm (see metaheuristics EDA_ACS or DDE_RLS for example). In particular, the implementation proposed by Fernandez-Viagas and Framinan (2014) is the most efficient one. Additionally, the difference in solution quality between IG-based algorithms and other methods is even greater in the new set of instances of (Vallada et al., 2015) which also consider a higher number of jobs and machines, a fact which explains why some metaheuristics tested on just a subset of the instances of (Taillard, 1993) were found to be efficient ones at their time.

Although the excellent performance of non-population based algorithms was shown by Nowicki and Smutnicki (1996); Pan et al. (2008); Ruiz and Stützle (2007) and Fernandez-Viagas and Framinan (2014), the literature using this type of metaheuristic is scarce and researchers have mainly been focused on the implementation of algorithms using several populations in parallel. In fact, most common metaheuristics chosen by the researches were Particle Swarm Optimization Algorithm (17 times), Genetic Algorithm (15 times), Ant Colony Algorithm (6 times) and Differential Algorithm (6 times). The remaining types have been implemented less than 4 times in the papers analysed.

Regarding heuristics, most have been identified and classified as variations of the NEH algorithm. Among the 19 coded algorithms, only 5 heuristics (NEHFF, $FRB4_k$, FRB2, FRB3 and FRB5) could be classified as efficient. Similar results have been found for both Taillard and VRF instances. Nevertheless, when they are compared with the best metaheuristic under the stopping criteria of the heuristic, all efficient heuristics have been outperformed by the metaheuristic, with the exception of NEHFF since that heuristic is the initial solution of the metaheuristic. Hence, this fact clearly indicates a way of proceeding when future new heuristics are proposed in the literature. From now, constructive and improvement heuristics should be directly compared either with the best metaheuristic under the same stopping criterion or with NEHFF with at least the same computational effort, as it might turn out that a few iterations of a good metaheuristic already give better results.

Note that the best metaheuristic and the best heuristics include Taillard's acceleration as well as tie-breaking mechanisms, which are two special characteristics of the $Fm|prmu|C_{\max}$ problem. Obviously, the former probably represent the main reason for the excellent behaviour of insertion phases in the algorithms and could explain its extensive use in the heuristics and metaheuristics of the last decade, as well as the excellent performance of the NEH and IG-based algorithms. The latter represents an advance in the intensification of the algorithms applying special knowledge of the problem. In our opinion, these facts highlight that future advances in this field will come from a better understanding of the problem and its properties.

Acknowledgements

The authors are sincerely grateful to the anonymous referees, who provide very valuable comments on the earlier version of the paper. This research has been funded by the Spanish Ministry of Science and Innovation, under projects "ADDRESS" (DPI2013-44461-P/DPI) and "SCHEYARD" (DPI2015-65895-R) co-financed by FEDER funds.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2016.09.055.

References

Agarwal, A., Colak, S., & Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research, 169*(3), 801–815.

Ahmadizar, F. (2012). A new ant colony algorithm for makespan minimization in permutation flow shops. *Computers and Industrial Engineering, 63*(2), 355–361.

Carlier, J. (1978). Ordonnancements a contraintes disjonctives. *RAIRO Recherche Operationnelle, 12*(4), 333–350.

Chang, P.-C., & Chen, M.-H. (2014). A block based estimation of distribution algorithm using bivariate model for scheduling problems. *Soft Computing, 18*(6), 1177–1188. doi:10.1007/s00500-013-1136-1.

Chang, P.-C., Chen, M.-H., Tiwari, M., & Iquebal, A. (2013). A block-based evolutionary algorithm for flow-shop scheduling problem. *Applied Soft Computing Journal, 13*(12), 4536–4547.

Chang, P.-C., Chen, S.-H., Fan, C.-Y., & Chan, C.-L. (2008). Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems. *Applied Mathematics and Computation, 205*(2), 550–561.

Chang, P.-C., Chen, S.-H., Fan, C.-Y., & Mani, V. (2010). Generating artificial chromosomes with probability control in genetic algorithm for machine scheduling problems. *Annals of Operations Research, 180*(1), 197–211.

Chang, P.-C., Hsieh, J.-C., Chen, S.-H., Lin, J.-L., & Huang, W.-H. (2009). Artificial chromosomes embedded in genetic algorithm for a chip resistor scheduling problem in minimizing the makespan. *Expert Systems with Applications, 36*(3 PART 2), 7135–7141.

Chang, P.-C., Huang, W.-H., & Ting, C.-J. (2011). A hybrid genetic-immune algorithm with improved lifespan and elite antigen for flow-shop scheduling problems. *International Journal of Production Research, 49*(17), 5207–5230.

Chen, C.-L., Tzeng, Y.-R., & Chen, C.-L. (2015). A new heuristic based on local best solution for permutation flow shop scheduling. *Applied Soft Computing Journal, 29*, 75–81. doi:10.1016/j.asoc.2014.12.011.

Chen, R.-M., & Hsieh, F.-R. (2014). An exchange local search heuristic based scheme for permutation flow shop problems. *Applied Mathematics and Information Sciences, 8*(1 L), 209–215.

Chen, S.-H., Chang, P.-C., Cheng, T., & Zhang, Q. (2012). A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers and Operations Research, 39*(7), 1450–1457.

Dasgupta, P., & Das, S. (2015). A discrete inter-species cuckoo search for flowshop scheduling problems. *Computers and Operations Research, 60*(0), 111–120. http://dx.doi.org/10.1016/j.cor.2015.01.005.

Demirkol, E., Mehta, S., & Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research, 109*(1), 137–141.

Dong, X., Huang, H., & Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers and Operations Research, 35*(12), 3962–3968.

Eksioglu, B., Eksioglu, S., & Jain, P. (2008). A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. *Computers and Industrial Engineering, 54*(1), 1–11.

Etiler, O., Toklu, B., Atak, M., & Wilson, J. (2004). A genetic algorithm for flow shop scheduling problems. *Journal of the Operational Research Society, 55*(8), 830–835.

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

Fernandez-Viagas, V., & Framinan, J. (2015b). NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers and Operations Research, 60*, 27–36. doi:10.1016/j.cor.2015.02.002.

Fernandez-Viagas, V., & Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research, 45*, 60–67. http://dx.doi.org/10.1016/j.cor.2013.12.012.

Framinan, J., Gupta, J., & 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.

Framinan, J., & Leisten, R. (2006). A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics, 99*(1–2), 28–40.

Framinan, J., & Pastor, R. (2008). A proposal for a hybrid meta-strategy for combinatorial optimization problems. *Journal of Heuristics, 14*(4), 375–390.

Haq, A., Ramanan, T., Shashikant, K., & Sridharan, R. (2010). A hybrid neural network-genetic algorithm approach for permutation flow shop scheduling. *International Journal of Production Research, 48*(14), 4217–4231.

Hariharan, R., & Golden Renjith Nimal, R. (2014). Solving flow shop scheduling problems using a hybrid genetic scatter search algorithm. *Middle - East Journal of Scientific Research, 20*(3), 328–333. doi:10.5829/idosi.mejsr.2014.20.03.100.

Heller, J. (1960). Some numerical experiments for an m x j flow shop and its decision-theoretical aspects. *Operations Research, 8*(2), 178–184.

Hsu, C.-Y., Chang, P.-C., & Chen, M.-H. (2015). A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers and Industrial Engineering, 83*, 159–171. doi:10.1016/j.cie.2015.02.009.

Huang, W., & Wang, L. (2006). A local search method for permutation flow shop scheduling. *Journal of the Operational Research Society, 57*(10), 1248–1251.

Jarboui, B., Ibrahim, S., Siarry, P., & Rebai, A. (2008). A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers and Industrial Engineering, 54*(3), 526–538.

Kalczynski, P. J., & Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flow shops. *OMEGA, The International Journal of Management Science, 35*(1), 53–60.

Kalczynski, P. J., & Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers and Operations Research, 35*(9), 3001–3008.

Kalczynski, P. J., & Kamburowski, J. (2009). An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research, 198*(1), 93–101. http://dx.doi.org/10.1016/j.ejor.2008.08.021.

Kendall, G., Bai, R., Błazewicz, J., Causmaecker, P. D., Gendreau, M., John, R., et al. (2016). Good laboratory practice for optimization research. *Journal of the Operational Research Society, 67*(4), 676–689.

Kuo, I.-H., Horng, S.-J., Kao, T.-W., Lin, T.-L., Lee, C.-L., Terano, T., et al. (2009). An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications, 36*(3 PART 2), 7027–7032.

Laha, D., & Chakraborty, U. (2009). An efficient hybrid heuristic for makespan minimization in permutation flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 44*(5–6), 559–569.

Leisten, R., & Rajendran, C. (2014). Variability of completion time differences in permutation flow shop scheduling. *Computers and Operations Research, 54*, 155–167. doi:10.1016/j.cor.2014.08.015.

Li, X., & Yin, M. (2012). A discrete artificial bee colony algorithm with composite mutation strategies for permutation flow shop scheduling problem. *Scientia Iranica, 19*(6), 1921–1935.

Li, X., & Yin, M. (2013a). A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. *International Journal of Production Research, 51*(16), 4732–4754.

Li, X., & Yin, M. (2013b). An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. *Advances in Engineering Software, 55*, 10–31.

Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied Mathematics and Computation, 175*(1), 773–785.

Lian, Z., Gu, X., & Jiao, B. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals, 35*(5), 851–861.

Liao, C.-J., Tseng, C.-T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers and Operations Research, 34*(10), 3099–3111.

Lin, Q., Gao, L., Li, X., & Zhang, C. (2015). A hybrid backtracking search algorithm for permutation flow-shop scheduling problem. *Computers and Industrial Engineering, 85*, 437–446. http://dx.doi.org/10.1016/j.cie.2015.04.009.
Liu, B., Wang, L., & Jin, Y.-H. (2007). An effective pso-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 37*(1), 18–27.
Liu, H., Gao, L., & Pan, Q.-K. (2011). A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. *Expert Systems with Applications, 38*(4), 4348–4360.
Liu, R., Ma, C., Ma, W., & Li, Y. (2013). A multipopulation pso based memetic algorithm for permutation flow shop scheduling. *The Scientific World Journal*.
Liu, Y., Yin, M., & Gu, W. (2014). An effective differential evolution algorithm for permutation flow shop scheduling problem. *Applied Mathematics and Computation, 248*, 143–159. doi:10.1016/j.amc.2014.09.010.
Liu, Y.-F., & Liu, S.-Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing Journal, 13*(3), 1459–1463.
Low, C., Yeh, J.-Y., & Huang, K.-I. (2004). A robust simulated annealing heuristic for flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology, 23*(9–10), 762–767.
Marinakis, Y., & Marinaki, M. (2013). Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Computing, 17*(7), 1159–1173.
M'Hallah, R. (2014). An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *International Journal of Production Research, 52*(13), 3802–3819.
Nagano, M., & Moccellin, J. (2002). A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society, 53*(12), 1374–1379.
Nagano, M., Ruiz, R., & Lorena, L. (2008). A constructive genetic algorithm for permutation flowshop scheduling. *Computers and Industrial Engineering, 55*(1), 195–207.
Nanz, S., & Furia, C. A. (2015). A comparative study of programming languages in rosetta code. In *Proceedings of the 37th international conference on software engineering: vol. 1* (p. 778).
Nawaz, M., Enscore, E., Jr., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science, 11*(1), 91–95.
Nearchou, A. (2004a). The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics, 88*(2), 191–203.
Nearchou, A. (2004b). Flow-shop sequencing using hybrid simulated annealing. *Journal of Intelligent Manufacturing, 15*(3), 317–328.
Nearchou, A. (2004c). A novel metaheuristic approach for the flow shop scheduling problem. *Engineering Applications of Artificial Intelligence, 17*(3), 289–300.
Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research, 91*(1), 160–175. doi:10.1016/0377-2217(95)00037-2.
Nowicki, E., & Smutnicki, C. (2006). Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research, 169*(2), 654–666.
Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research, 171*(2), 674–692.
Pan, Q.-K., Tasgetiren, M., & Liang, Y.-C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering, 55*(4), 795–816.
Pinedo, M. (2012). *Scheduling: Theory, algorithms and systems* (4th ed.). Prentice Hall.
Prabhaharan, G., Khan, B., & Rakesh, L. (2006). Implementation of GRASP in flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 30*(11–12), 1126–1131.
Qian, B., Wang, L., Hu, R., Wang, W.-L., Huang, D.-X., & Wang, X. (2008). A hybrid differential evolution method for permutation flow-shop scheduling. *International Journal of Advanced Manufacturing Technology, 38*(7–8), 757–777.
Rad, S. F., Ruiz, R., & Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science, 37*(2), 331–345.
Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research, 155*(2), 426–438.
Rajkumar, R., & Shahabudeen, P. (2009). An improved genetic algorithm for the flowshop scheduling problem. *International Journal of Production Research, 47*(1), 233–249.
Ramanan, T., Sridharan, R., Shashikant, K., & Haq, A. (2011). An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing, 22*(2), 279–288.
Reeves, C. (1995). A genetic algorithm for flowshop sequencing. *Computers and Operations Research, 22*(1), 5–13.
Reza Hejazi, S., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research, 43*(14), 2895–2929.
Ribas, I., Companys, R., & Tort-Martorell, X. (2010). Comparing three-step heuristics for the permutation flow shop problem. *Computers and Operations Research, 37*(12), 2062–2070.
Rinnooy Kan, A. H. G. (1976). *Machine scheduling problems: Classification, complexity and computations*. The Hague: Martinus Nijhoff.
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, The International Journal of Management Science, 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.
Saravanan, M., Noorul Haq, A., Vivekraj, A., & Prasad, T. (2008). Performance evaluation of the scatter search method for permutation flowshop sequencing problems. *International Journal of Advanced Manufacturing Technology, 37*(11–12), 1200–1208.
Sayadi, M., Ramezanian, R., & Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations, 1*(1), 1–10.
Solimanpur, M., Vrat, P., & Shankar, R. (2004). A neuro-tabu search heuristic for the flow shop scheduling problem. *Computers and Operations Research, 31*(13), 2151–2164.
Stütze, T. (1998). Applying iterated local search to the permutation flow shop problem. *Technical report, AIDA-98-04, FG Intellektik, FB Informatik, TU Darmstadt*.
Sun, Y., Zhang, C., Gao, L., & Wang, X. (2011). Multi-objective optimization algorithms for flow shop scheduling problem: A review and prospects. *International Journal of Advanced Manufacturing Technology, 55*(5–8), 723–739. doi:10.1007/s00170-010-3094-4.
Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research, 47*(1), 65–74.
Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285.
Tasgetiren, M., Liang, Y.-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research, 177*(3), 1930–1947.
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.
Tzeng, Y.-R., & Chen, C.-L. (2012). A hybrid eda with acs for solving permutation flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 60*(9–12), 1139–1147.
Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA, The International Journal of Management Science, 38*(1–2), 57–67.
Vallada, E., Ruiz, R., & Framinan, J. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research, 240*, 666–677.
Vasiljevic, D., & Danilovic, M. (2015). Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems, 35*, 1–9. doi:10.1016/j.jmsy.2014.11.011.
Watson, J., Barbulescu, L., Whitley, L., & Howe, A. (2002). Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing, 14*(2), 98–123.
Xie, Z., Zhang, C., Shao, X., Lin, W., & Zhu, H. (2014). An effective hybrid teaching-learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software, 77*, 35–47.
Yagmahan, B., & Yenisey, M. (2008). Ant colony optimization for multi-objective flow shop scheduling problem. *Computers and Industrial Engineering, 54*(3), 411–420.
Ying, K.-C., & Liao, C.-J. (2004). An ant colony system for permutation flow-shop sequencing. *Computers and Operations Research, 31*(5), 791–801.
Ying, K.-C., & Lin, S.-W. (2013). A high-performing constructive heuristic for minimizing makespan in permutation flowshops. *Journal of Industrial and Production Engineering, 30*(6), 355–362.
Zanakis, S., Evans, J., & Vazacopoulos, A. (1989). Heuristic methods and applications: A categorized survey. *European Journal of Operational Research, 43*(1), 88–110.
Zhang, C., Ning, J., & Ouyang, D. (2010a). A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Computers and Industrial Engineering, 58*(1), 1–11.
Zhang, C., & Sun, J. (2009). An alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications, 36*(3 PART 1), 5162–5167.
Zhang, C., Sun, J., Zhu, X., & Yang, Q. (2008). An improved particle swarm optimization algorithm for flowshop scheduling problem. *Information Processing Letters, 108*(4), 204–209.
Zhang, J., Zhang, C., & Liang, S. (2010b). The circular discrete particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications, 37*(8), 5827–5834.
Zhang, L., & Wu, J. (2014). A PSO-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*.
Zheng, T., & Yamashiro, M. (2010). Solving flow shop scheduling problems by quantum differential evolutionary algorithm. *International Journal of Advanced Manufacturing Technology, 49*(5–8), 643–662.
Zobolas, G., Tarantilis, C., & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers and Operations Research, 36*(4), 1249–1267.
Zäpfel, G., Braune, R., & Bögl, M. (2010). *Metaheuristic search concepts*. Springer.