Fernandez-Viagas, V.; Ruiz García, R.; Framinan, J. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. European Journal of Operational Research. 257(3):707-721. https://doi.org/10.1016/j.ejor.2016.09.055

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

Victor Fernandez-Viagas[1]*, Rubén Ruiz[2], Jose M. Framinan[1]

[1] Industrial Management, School of Engineering, University of Seville,

Ave. Descubrimientos s/n, E41092 Seville, Spain, {vfernandezviagas,framinan}@us.es

[2] 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, rruiz@eio.upv.es

August 7, 2016

**Abstract**

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 makesan 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.

**Keywords: Scheduling, Flowshop, Heuristics, Metaheuristics, Makespan, Review, Computational evaluation**

*Corresponding author. Tel.: +34-954487220.

# 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 [21, 71, 74]).

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 [36], [20], [87], [51], [22] or [81] for other objectives in the PFSP). According to the notation of [63], this problem is denoted as $Fm|prmu|C_{\max}$. Since [73] 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. [74] 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 [55] was clearly the most efficient among the constructive heuristics for the problem, and the Iterated Local Search [80] presented itself as the most efficient metaheuristic for the problem.

Since the publication of the work by [74], 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 [76]– have improved the best existing algorithms in [74]. 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.).

  - 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 [76]).
  - 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 [88]. 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 [83], [3], [70], [90], [14], [26]).
  - A new indicator has been proposed by [19] 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 [74] 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 $i = 1, ..., m$ and $j = 1, ..., n$, 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. [103], [95]). For an in-depth classification of the $Fm|prmu|C_{\max}$ problem, we refer to [21]. However, in this paper we use the same division as in [74], 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 [74], several constructive heuristics have been proposed in the literature, most of them variants of the NEH heuristic by [55]. 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 [82] (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 [72] 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 [72], NEHR-di [72], NEH [55], NEH-di [72], NEH1 [30] and NEH1-di [72].

  - AD: sum of the mean and deviation of the processing times (proposed by [15]).
  - NM: order proposed by [52] and used in NEMR and NEMR-di heuristics by [72].
  - KK1: Sorting criterion proposed by [31]. This initial order is applied in NEHKK1 [31] and NEHKK1-di [72] heuristics.
  - KK2: Sorting criterion proposed by [32] 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 [30]). 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 [30]; TBKK2, proposed by [31]; TBKK3, proposed by [32]; DCH, proposed by [15]; RCT, proposed by [72]; and the FF, proposed by [18].

- $c$: This field is associated with the reversibility property of the problem (see [72]). 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 [30], [31], [32], [15], and [18] respectively – maintain the original complexity of $O(n^2m)$. Other

variants with a greater complexity have been proposed by [72], 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 [94] 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 [72] 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 [66]. 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, [89] 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, [83]; T2, non-complete set of instances of [83]; R, [70], C, [3]; D, [14]; W, [90]; H, [26]; O, Other set of instances). The seventh column shows the $ARPD$ values of the metaheuristics when tested on Taillard's benchmark [83]. Average Relative Percentage Deviation values of algorithm $j$ are denoted as $ARPD_j$ and are calculated as follows:

$$ARPD_j = \frac{\sum_{\forall i} RPD_{i,j}}{I} \tag{1}$$

where $I$ is the number of instances for which the $RPD$ (Relative Percentage Deviation) values are

Table 1: Summary of heuristics

| Heuristic | NEH Notation | Paper |
|---|---|---|
| **RAER** | $NEH(rand\|RCT\|d)$ | [72] |
| **RAER-di** | $NEH(rand\|RCT\|di)$ | [72] |
| **KKER** | $NEH(KK1\|RCT\|d)$ | [72] |
| **KKER-di** | $NEH(KK1\|RCT\|di)$ | [72] |
| **NEHR** | $NEH(SD\|RCT\|d)$ | [72] |
| **NEHR-di** | $NEH(SD\|RCT\|di)$ | [72] |
| **NEMR** | $NEH(NM\|RCT\|d)$ | [72] |
| **NEMR-di** | $NEH(NM\|RCT\|di)$ | [72] |
| NEH | $NEH(SD\|FS\|d)$ | [55] |
| **NEH-di** | $NEH(SD\|FS\|di)$ | [72] |
| NEH1 | $NEH(SD\|TBKK1\|d)$ | [30] |
| **NEH1-di** | $NEH(SD\|TBKK1\|di)$ | [72] |
| NEHKK1 | $NEH(KK1\|TBKK2\|d)$ | [31] |
| **NEHKK1-di** | $NEH(KK1\|TBKK2\|di)$ | [72] |
| **NEHKK2** | $NEH(KK2\|TBKK3\|d)$ | [32] |
| NEHD | $NEH(AD\|DHC\|d)$ | [15] |
| **NEHD-di** | $NEH(AD\|DHC\|di)$ | [72] |
| **NEHFF** | $NEH(AD\|FF\|d)$ | [18] |
| **$CL_{WTS}$** | $NEH(SD\|FS\|d)$ with a backward shift mechanism in the insertion phase | [94] |
| $CL_{WOTS}$ | $NEH(SD\|RCT\|d)$ with a backward shift mechanism in the insertion phase | [94] |
| NEHI | Best of several runs of $NEH(-\|-\|-)$ | [89] |
| FRB1 | Similar to the $NEH(SD\|FS\|d)$ including a local search method in the insertion phase | [66] |
| **FRB2** | Similar to the $NEH(SD\|FS\|d)$ including a local search method in the insertion phase | [66] |
| **FRB3** | $NEH(SD\|FS\|d)$ including a local search method in the insertion phase | [66] |
| **$FRB4_k$** | $NEH(SD\|FS\|d)$ including a local search method in the insertion phase | [66] |
| **FRB5** | $NEH(SD\|FS\|d)$ including a local search method in the insertion phase | [66] |

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:

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

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 [83], 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 [83].

The eight 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 analyze 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. [76]). 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 [54].

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 [76]). 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 analyzed 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.

Table 2: Summary of metaheuristics I.

| Year | Ref. | Algorithm | Notation | Outperformed by | Testbed | $ARPD$(Taillard) | Coding Lang. | Parameter $t$ |
|---|---|---|---|---|---|---|---|---|
| 2004 | [93] | AC | ACS | [16], [77], [48], [2] | T1 | 1.4** | C++ | 608.11 |
| 2004 | [79] | Neuro-TS | EXTS | [16] | T2 | 0.245 | C++ | 2884.85 |
| 2004 | [67] | AC | PACO | [75]'*, [76]*, [62], [62]", [35], [100], [86], [102], [37], [2]*, [85], [10] | T2 | 0.71** | — | — |
| 2004 | [67] | AC | M-MMAS | [75]'*, [76]*, [62], [62]", [35], [86], [102], [37], [2]*, [85] | T2 | 0.80** | — | — |
| 2004 | [49] | SA | LWK-SA1 | [91], [39], [38] | T2,D | 0.853 | — | 331.8 |
| 2004 | [17] | GA | GA | [42]* | O | — | Pascal | — |
| 2004 | [57] | SA | Hybrid SAA | — | T2 | 0.893 | Pascal | — |
| 2004 | [58] | Hybrid SA | Hybrid SAA | — | T2 | 1.081 | Pascal | 134.06 |
| 2004 | [56] | GA | GA | [98]*, [96]*, [99], [97] | T2 | 1.84** | Pascal | — |
| 2006 | [1] | ALA | NEH-ALA | [53]*, [35] | T1,C,R,H | 1.514 | Visual Basic | 7907.6 |
| 2006 | [75] | GA | GA_RMA | [75]'*, [76]*, [62], [62]", [53]*, [11], [37], [98]* | T1 | 1.12**,1.09**,1.02** | Delphi | 30,60,90 |
| 2006 | [75]' | GA | HGA_RMA | [76]*, [62], [62]", [86], [10] | T1 | 0.55**,0.47**,0.45** | Delphi | 30,60,90 |
| 2006 | [61] | DE | DE | [53]*, [65] | O | — | C++ | — |
| 2006 | [60] | SS | **MSSA** | — | T2 | >0.054 | — | 1139.33 |
| 2006 | [40] | PSO | SPSOA | [9], [99], [5], [27], [4] | T2 | 3.002 | — | — |
| 2006 | [64] | GRASP | GRASP | — | T2,C,R | 19.09 | — | — |
| 2006 | [28] | ILS | LS | — | C,R | — | C | |
| 2007 | [76] | IG | **IG_RS$_{LS}$** | [62], [62]", [18]'* | T1 | 0.44** | Delphi | 60 |
| 2007 | [84] | PSO | PSO$_{spv}$ | [62]', [12], [84]', [29], [42]*, [50] | T2 | 4.01** | C | 21.1 |
| 2007 | [84]' | PSO | PSO$_{vns}$ | [62], [62]", [91], [39], [44], [86], [48], [38], [47], [85], [10], [43] | T2,W | 0.47** | C | 264.64 |
| 2007 | [44] | MA-PSO | PSOMA | [46], [45], [48], [38], [47], [5], [43] | C,R | — | Matlab | — |
| 2007 | [42] | PSO | PSO | [101], [39], [13] | T2,D | 2.409** | C++ | 111.68 |
| 2008 | [16] | TS | **3XTS** | — | T1 | 0.15*** | C++ | 196.93 |
| 2008 | [77] | SS | SS | — | T1 | 1.57*** | C++ | 59.9 |
| 2008 | [53] | GA | CGALS | [48] | T1 | 1.02** | Delphi | 60 |
| 2008 | [23] | GA | BDS | — | T2 | 0.64 | — | 7350 |
| 2008 | [29] | PSO | CPSO | [62], [62]', [62]", [12] | T2 | 3.4** | C++ | 3.42 |
| 2008 | [29]' | PSO | CPSO-PNEH | — | T2 | 0.59** | C++ | 19.12 |
| 2008 | [29]" | PSO | **H-CPSO** | [62], [62]" | T2 | 0.45** | C++ | 229.34 |
| 2008 | [6] | GA | ACGA | [9], [27] | R | — | — | — |
| 2008 | [65] | DE | HDE | [101], [39], [38], [37] | C,R | — | Delphi | — |
| 2008 | [41] | PSO | NPSO | [96]*, [98]*, [97], [47], [34] | T2 | 1.323 | — | — |
| 2008 | [92] | AC | ACS | [98]*, [37] | R | — | VP | — |
| 2008 | [62] | IG | **IG$_{RIS}$** | [18]* | T1 | 0.33** | C++ | 30 |
| 2008 | [62]' | DE | DDE | [62]*, [62]"*, [27] | T1 | 1.05** | C++ | 30 |
| 2008 | [62]" | DE | **DDE$_{RLS}$** | — | T1 | 0.32** | C++ | 30 |
| 2008 | [98] | PSO | IPSO | [37] | T1 | 0.76** | 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; $^*$, Compared under the same conditions; $'$, In case of a paper proposing two methods, it is used to distinguish the second one from the first one; $''$, In case of a paper proposing three methods, it is used to distinguish the third one from the first and second one; $^{**}$, $ARPD$ taken directly from the paper; $^{***}$, $ARPD$ corrected by 0.565.

Table 3: Summary of metaheuristics II

| Year | Ref. | Algorithm | Notation | Outperformed by | Testbed | $ARPD$ (Taillard) | Coding Lang. | Parameter $t$ |
|---|---|---|---|---|---|---|---|---|
| 2009 | [8] | GA | ACEGA | — | R | — | — | — |
| 2009 | [35] | Hybrid | PSA | — | T2 | 1.049 | C | — |
| 2009 | [102] | GA with VNS | $\text{NEGA}_{\text{VNS}}$ | [86], [10] | T1 | 0.468 | C++ | 112.93 |
| 2009 | [97] | PSO | ATPPSO | [39], [38], [47] | T2 | 1.269 | — | — |
| 2009 | [34] | Hybrid PSO | HPSO | [47] | T2 | 0.760 | C | 555.04 |
| 2009 | [85] | Hybrid GA | Hybrid GA | [13] | T2 | 0.63*** | C++ | 199.79 |
| 2009 | [68] | GA | IGA | — | C,R | — | C | — |
| 2010 | [78] | DF | Discrete Firefly | — | D | — | Matlab | — |
| 2010 | [101] | DE | QDEA | [91], [37], [39] | C,R,D | — | — | — |
| 2010 | [99] | PSO | L-CDPSO | [39], [38] | T2 | 0.409 | — | — |
| 2010 | [96] | PSO | I-ATTPSO | — | T2 | 1.331 | — | — |
| 2010 | [24] | NN-GA | ANN-GA-RIPS | — | T2 | 2.519 | — | — |
| 2010 | [7] | GA | ACGA | [5], [27] | R | — | — | — |
| 2011 | [9] | GA | HGIA | [5], [27] | T2,R | 1.16 | C++ | — |
| 2011 | [45] | Hybrid PSO | PSO-EDA_PI | [46], [5] | C,R,W | — | Matlab | — |
| 2011 | [69] | NN | ANN-GA | — | T2 | 2.34** | — | — |
| 2012 | [12] | GA | Self-Guided GA | [62]', [27] | T2 | 1.85** | Java | — |
| 2012 | [86] | EDA with AC | $\mathbf{EDA_{ACS}}$ | — | T1 | 0.572**,0.508**,0.463** | C | 30,60,90,200 |
| 2012 | [37] | BCA | CDABC | — | C,R,T1,D | 0.62** | Matlab | 120 |
| 2012 | [2] | AC | NACA | — | T1 | 0.582** | C++ | 108.92 |
| 2013 | [48] | Hybrid BCA | HDABC | — | T2,R | 0.48** | C++ | 42.52 |
| 2013 | [50] | PSO | PSOENT | [102] | T1 | 1.65** | Fortran | 104.42 |
| 2013 | [46] | MA-PSO | MPSOMA | — | C,R | — | Matlab | — |
| 2013 | [39] | DE-MA | ODDE | [91] | C,R,T2,D | 0.400 | Matlab | — |
| 2013 | [38] | CS | **HCS** | — | C,R,T2,D | 0.401 | Matlab | — |
| 2013 | [5] | EA | BBEA | [27] | T2,R | 1.75** | — | — |
| 2014 | [91] | Hybrid | HTLBO | — | C,R,D | — | C++ | — |
| 2014 | [100] | PSO | **PSO** | — | T2 | 0.39** | C++ | — |
| 2014 | [18] | IG | $\mathbf{IG_{RIS}(TB_{FF})}$ | — | T1 | 0.461,0.385,0.353** | C# | 30,60,90 |
| 2014 | [18]' | IG | $\mathbf{IG\_RS_{LS}(TB_{FF})}$ | — | T1 | 0.461,0.376,0.350** | C# | 30,60,90 |
| 2014 | [11] | SA | SEASA | — | T1 | 0.94** | — | 35 |
| 2014 | [47] | Hybrid DE | L-HDE | — | T2,C,R | 0.750 | Matlab | — |
| 2014 | [4] | EDA | BBEDA | [27] | T2,R | 1.420** | — | — |
| 2015 | [25] | GA-SS | HGSS | — | D | — | C++ | — |
| 2015 | [10] | PA | HLBS | — | T1 | 0.45**,0.38**,0.35**,0.30** | C | 30,60,90,200 |
| 2015 | [13] | CS | DISCS | — | T2,W | 2.84** | Matlab | 10.88 |
| 2015 | [27] | EA | LMBBEA | — | T2,R | 0.89 | — | — |
| 2015 | [43] | EA | HBSA | [65] | C,R | — | — | — |

# 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 GHz and 16 Gbytes 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 GBytes of RAM. Several benchmarks have been used (see e.g. [3,14,26,70,83,90]) in the literature to perform comparisons between algorithms. Among them, the most extended one is the benchmark from [83] 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, [88] 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 [83]. 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 Equation (1)–, and the computational effort by the Average CPU time (denoted as $ACPU$) which can be defined as follows:

$$ACPU_j = \frac{\sum_{\forall i} CPU_{i,j}}{I} \tag{3}$$

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, [19] 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 $Fm|prmu|\sum C_j$ problem contribute more than 88% to the average CPU time indicator). In order to avoid these problems, [19] defined

$ARPT_j^{'}$ as the average relative percentage time consumed by algorithm $j$ as follows:

$$ARPT_j^{'} = \frac{\sum_{\forall i} RPT_{i,j}}{I} \tag{4}$$

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

$$RPT_{i,j} = \frac{CPU_{i,j} - ACT_i}{ACT_i} \tag{5}$$

and $ACT_i$ can be computed as

$$ACT_i = \frac{\sum_{\forall i} CPU_{i,j}}{J} \tag{6}$$

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:

$$ARPT_j = ARPT_j^{'} + 1 \tag{7}$$

$ARPT$ 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 [74]. 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. $\text{FRB4}_6$, $\text{FRB4}_8$) in the same paper. Additionally, the tie-breaking mechanisms of [15], [30], [31] as well as the original one of [55] are statistically outperformed by the tie-breaking mechanism proposed by [18] 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, $\text{CL}_{\text{WTS}}$, FRB2, FRB3, $\text{FRB4}_{\text{k}}$ ($k \in \{2, 4, 6, 8, 10, 12\}$) 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 $\text{FRB4}_{10}$ or $\text{FRB4}_{12}$ for example). Note that there are two possible interpretations of $RCT$, the idle-time- based tie-breaking mechanism proposed by [72]. The authors state that this mechanism can be implemented in $O(n^2m^2)$. However, as explained in [18], 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 [83], 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 $\leq$ 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 $\leq$ 90). 12 metaheuristics fulfil these requirements: EXTS by [79]; HGA_RMA by [75]; MSSA by [60]; IG_RS$_{LS}$ by [76]; IG$_{RIS}$ by [62]; DDE$_{RLS}$ by [62]; 3XTS by [16]; EDA$_{ACS}$ by [86]; PSO by [100]; IG_RS$_{LS}$(TB$_{FF}$) by [18]; IG$_{RIS}$(TB$_{FF}$) by [18]. Among them, EXTS and HGA_RMA, are discarded since they are outperformed in statistically and/or sound comparisons by [16] and [76] respectively. Additionally, the H-CPSO algorithm by [29] has been implemented due to its promising results despite being outperformed by [62] under different stopping criteria and conditions. Metaheuristic HCS by [38] 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 [59] in the comparisons, given its excellent performance and the fact that it was not included in the last computational evaluation by [74]. The reason behind this omission is explained in [76] 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 [100], 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.

Note that even recently published, this computational evaluation follows many of the practices highlighted in [33]. 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 Figures 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$_{\text{WTS}}$ 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$_{10}$,

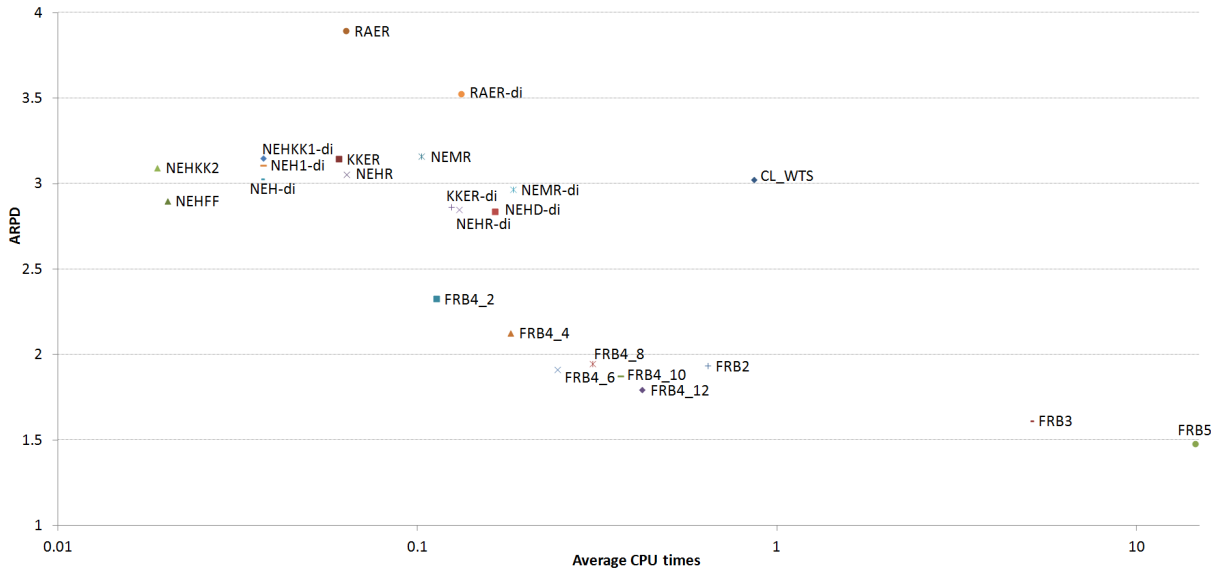


Figure 1: $ARPD$ versus $ACPU$ of heuristics in logarithmic scale on Taillard's instances.

FRB4$_{12}$, FRB3 and FRB5 (shown with a black circle in Figure 2). To be able to compare heuristics with different stopping criteria, they are grouped into clusters with similar $ARPT$ values (see Figure 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). 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

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 |

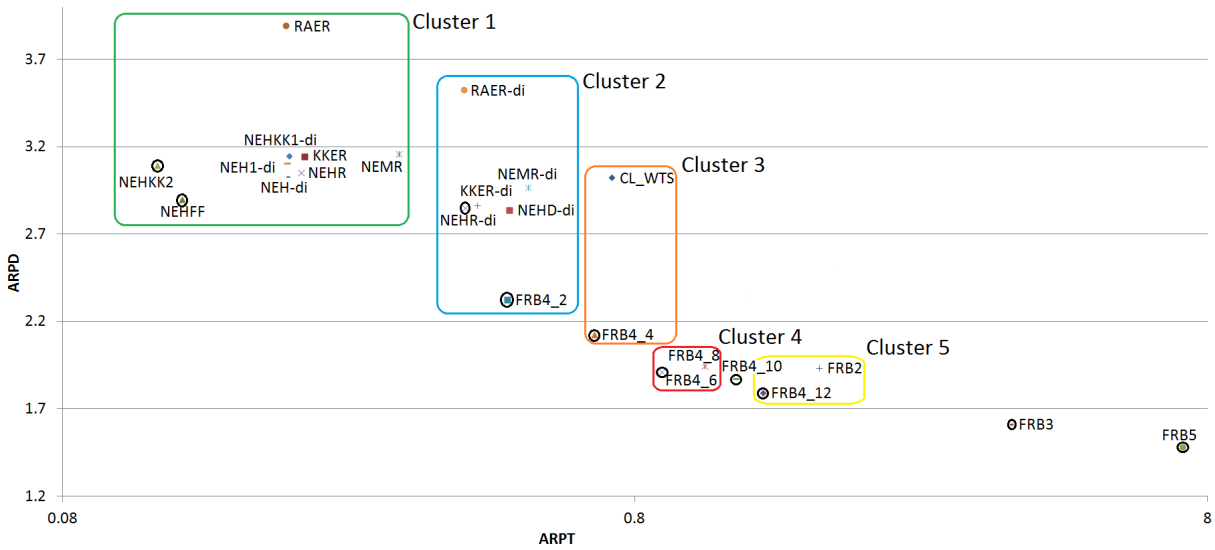


Figure 2: $ARPD$ versus $ARPT$ of heuristics in logarithmic scale on Taillard's instances.

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 | — |

of [88]. 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 Figure 3 comparing $ARPD$ versus $ACPU$, and in Figure 4 comparing $ARPD$ versus $ARPT$. In terms of $ARPD$ and $ARPT$, efficient heuristics are shown with a black circle in Figure 4. Note that regarding the NEH-based heuristics of [72] 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 Figure 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.

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 |

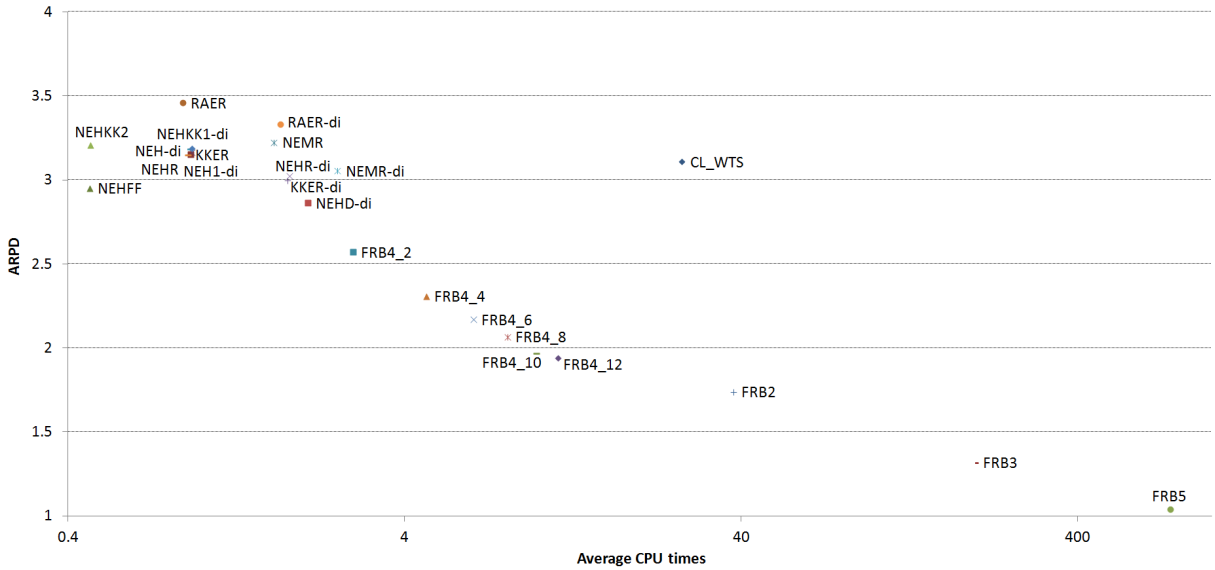


Figure 3: $ARPD$ versus $ACPU$ of heuristics in logarithmic scale on VRF instances.

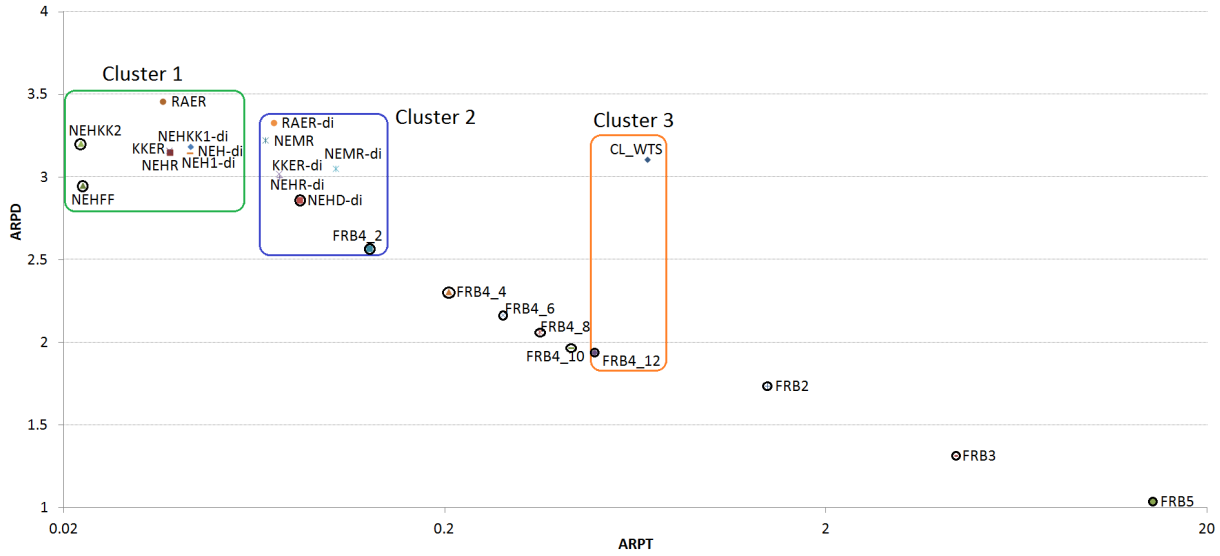


Figure 4: $ARPD$ vs $ARPT$ of heuristics in logarithmic scale on VRF instances.

## 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 [83] and [88]. 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. For both sets of instances, the best metaheuristics are those based on the Iterated Greedy (IG_RS$_{\text{LS}}$) proposed by [76], see the results found by IG_RS$_{\text{LS}}$, IG$_{\text{RIS}}$, IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) and IG$_{\text{RIS}}$(TB$_{\text{FF}}$) for example. These results are also confirmed by the DDE$_{\text{RLS}}$, 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$_{\text{LS}}$(TB$_{\text{FF}}$) proposed by [18], 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$_{\text{LS}}$(TB$_{\text{FF}}$) 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$_{\text{LS}}$(TB$_{\text{FF}}$) 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

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 | [59] | 0.97 | 0.87 | 0.84 | 2.16 | 1.96 | 1.85 |
| MSSA | [60] | 1.00 | 0.91 | 0.84 | 2.17 | 1.96 | 1.84 |
| IG_RS$_{\text{LS}}$ | [76] | 0.47 | 0.40 | 0.37 | 0.96 | 0.77 | 0.67 |
| IG$_{\text{RIS}}$ | [62] | 0.49 | 0.42 | 0.38 | 0.85 | 0.67 | 0.56 |
| DDE$_{\text{RLS}}$ | [62] | 0.52 | 0.47 | 0.43 | 0.92 | 0.77 | 0.69 |
| 3XTS | [16] | 1.64 | 1.34 | 1.24 | 2.89 | 2.65 | 2.47 |
| H-CPSO | [29] | 0.84 | 0.75 | 0.70 | 1.65 | 1.41 | 1.28 |
| EDA$_{\text{ACS}}$ | [86] | 0.60 | 0.51 | 0.47 | 1.43 | 1.25 | 1.16 |
| HCS | [38] | 1.55 | 1.42 | 1.35 | 2.54 | 2.35 | 2.27 |
| PSO | [100] | 1.09 | 0.95 | 0.84 | 2.51 | 2.14 | 1.93 |
| IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) | [18] | 0.37 | 0.32 | 0.28 | 0.60 | 0.46 | 0.37 |
| IG$_{\text{RIS}}$(TB$_{\text{FF}}$) | [18] | 0.42 | 0.34 | 0.31 | 0.61 | 0.47 | 0.38 |

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$_{\text{LS}}$(TB$_{\text{FF}}$) metaheuristic (see the difference of $ARPD$ between 3XTS and IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) 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$_{\text{RIS}}$(TB$_{\text{FF}}$) (see Table 8). In view of the results, although there are many papers proposing metaheuristics, only the Iterated Greedy variants proposed by [18] statistically outperform IG_RS$_{\text{LS}}$ 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 [74] (see Tables 2 and 3) and since the original Iterated Greedy algorithm proposed by [76]. 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$_{\text{RIS}}$(TB$_{\text{FF}}$) and the IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) 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.

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 |

## 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 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 [83] and [88]. A summary of the results is shown in Table 9 as well as in Figures 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 [66] which have $ARPD$ values similar to or even better than those obtained by IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) 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$_{\text{LS}}$(TB$_{\text{FF}}$) as a state-of-the-art method for constructive and improvement heuristics.

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 |

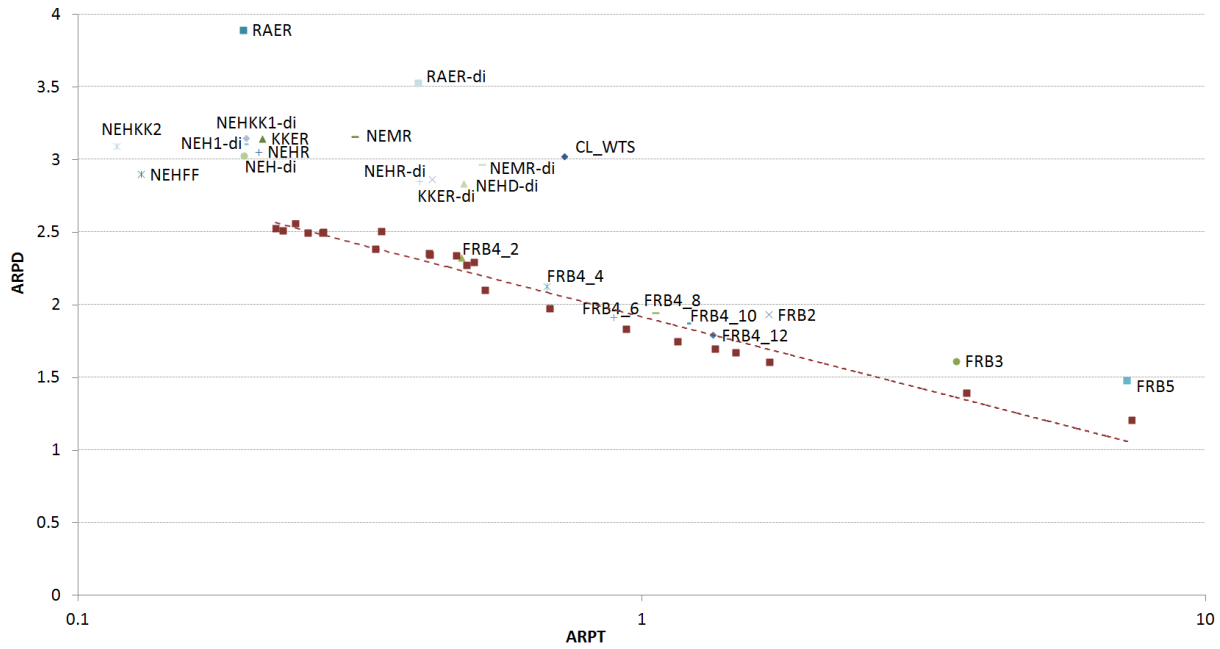


Figure 5: Heuristics versus IG_RS$_{LS}$(TB$_{FF}$) on the set of instances of [83]. X-axis (variable $ARPT$) is shown in logarithmic scale.

# 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 [83] and [88]. 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 analyze tentative best

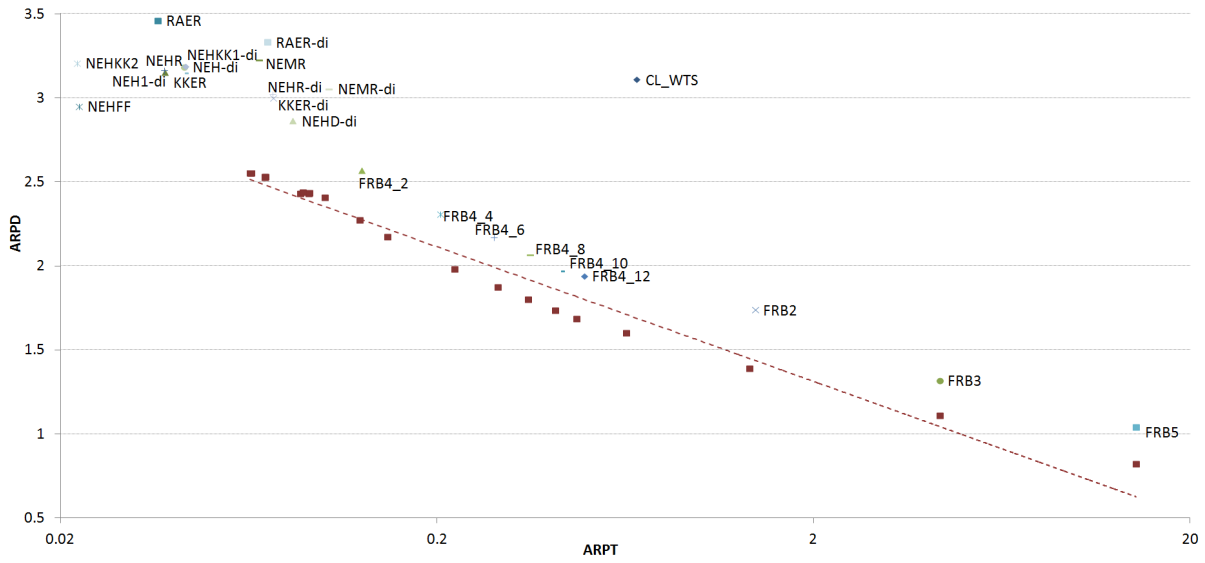


Figure 6: Heuristics versus IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$) on the set of instances of [88]. X-axis (variable $ARPT$) is shown in logarithmic scale.

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 [33].

Among all coded metaheuristics, algorithms based in the IG method of [76] 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 [18] 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 [88] 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 [83] were found to be efficient ones at their time.

Although the excellent performance of non-population based algorithms was shown by [59], [76], [62] and [18], 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, $\text{FRB4}_{\text{k}}$, 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.

# References

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

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

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

[4] P.-C. Chang and M.-H. Chen. A block based estimation of distribution algorithm using bivariate model for scheduling problems. *Soft Computing*, 18(6):1177–1188, 2014.

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

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

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

[8] P.-C. Chang, J.-C. Hsieh, S.-H. Chen, J.-L. Lin, and W.-H. Huang. 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, 2009.

[9] P.-C. Chang, W.-H. Huang, and C.-J. Ting. 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, 2011.

[10] C.-L. Chen, Y.-R. Tzeng, and C.-L. Chen. A new heuristic based on local best solution for permutation flow shop scheduling. *Applied Soft Computing Journal*, 29:75–81, 2015.

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

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

[13] P. Dasgupta and S. Das. A discrete inter-species cuckoo search for flowshop scheduling problems. *Computers and Operations Research*, 60(0):111 – 120, 2015.

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

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

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

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

[18] V. Fernandez-Viagas and J. M. Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research*, 45:60 – 67, 2014.

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

[20] V. Fernandez-Viagas and J.M. Framinan. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers and Operations Research*, 60:27–36, 2015.

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

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

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

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

[25] R. Hariharan and R.J. Golden Renjith Nimal. Solving flow shop scheduling problems using a hybrid genetic scatter search algorithm. *Middle - East Journal of Scientific Research*, 20(3):328–333, 2014.

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

[27] C.-Y. Hsu, P.-C. Chang, and M.-H. Chen. A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 83:159–171, 2015.

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

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

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

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

[32] P. J. Kalczynski and J. Kamburowski. An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, 198(1):93 – 101, 2009.

[33] G. Kendall, R. Bai, J. Błazewicz, P. De Causmaecker, M. Gendreau, R. John, J. Li, B. McCollum, E. Pesch, R. Qu, N. Sabar, G. Vanden Berghe, and A. Yee. Good laboratory practice for optimization research. *Journal of the Operational Research Society*, 67(4):676–689, 2016.

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

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

[36] R. Leisten and C. Rajendran. Variability of completion time differences in permutation flow shop scheduling. *Computers and Operations Research*, 54:155–167, 2014.

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

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

[39] X. Li and M. Yin. An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. *Advances in Engineering Software*, 55:10–31, 2013.

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

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

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

[43] Q. Lin, L. Gao, X. Li, and C. Zhang. A hybrid backtracking search algorithm for permutation flow-shop scheduling problem. *Computers and Industrial Engineering*, 85:437 – 446, 2015.

[44] B. Liu, L. Wang, and Y.-H. Jin. An effective pso-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):18–27, 2007.

[45] H. Liu, L. Gao, and Q.-K. Pan. A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. *Expert Systems with Applications*, 38(4):4348–4360, 2011.

[46] R. Liu, C. Ma, W. Ma, and Y. Li. A multipopulation pso based memetic algorithm for permutation flow shop scheduling. *The Scientific World Journal*, 2013.

[47] Y. Liu, M. Yin, and W. Gu. An effective differential evolution algorithm for permutation flow shop scheduling problem. *Applied Mathematics and Computation*, 248:143–159, 2014.

[48] Y.-F. Liu and S.-Y. Liu. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing Journal*, 13(3):1459–1463, 2013.

[49] C. Low, J.-Y. Yeh, and K.-I. Huang. A robust simulated annealing heuristic for flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 23(9-10):762–767, 2004.

[50] Y. Marinakis and M. Marinaki. Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Computing*, 17(7):1159–1173, 2013.

[51] R. M'Hallah. 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, 2014.

[52] M.S. Nagano and J.V. Moccellin. A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society*, 53(12):1374–1379, 2002.

[53] M.S. Nagano, R. Ruiz, and L.A.N. Lorena. A constructive genetic algorithm for permutation flowshop scheduling. *Computers and Industrial Engineering*, 55(1):195–207, 2008.

[54] S. Nanz and C. A. Furia. A comparative study of programming languages in rosetta code. In *Proceedings of the 37th International Conference on Software Engineering*, volume 1, pages 778–778, 2015.

[55] M. Nawaz, E.E. Enscore Jr., and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95, 1983.

[56] A.C. Nearchou. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88(2):191–203, 2004.

[57] A.C. Nearchou. Flow-shop sequencing using hybrid simulated annealing. *Journal of Intelligent Manufacturing*, 15(3):317–328, 2004.

[58] A.C. Nearchou. A novel metaheuristic approach for the flow shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 17(3):289–300, 2004.

[59] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175, 1996.

[60] E. Nowicki and C. Smutnicki. Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, 169(2):654–666, 2006.

[61] G. Onwubolu and D. Davendra. Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2):674–692, 2006.

[62] Q.-K. Pan, M.F. Tasgetiren, and Y.-C. Liang. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55(4):795–816, 2008.

[63] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, fourth edition, 2012.

[64] G. Prabhaharan, B.S.H. Khan, and L. Rakesh. Implementation of GRASP in flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 30(11-12):1126–1131, 2006.

[65] B. Qian, L. Wang, R. Hu, W.-L. Wang, D.-X. Huang, and X. Wang. A hybrid differential evolution method for permutation flow-shop scheduling. *International Journal of Advanced Manufacturing Technology*, 38(7-8):757–777, 2008.

[66] S. F. Rad, R. Ruiz, and N. Boroojerdian. New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, 37(2):331–345, 2009.

[67] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438, 2004.

[68] R. Rajkumar and P. Shahabudeen. An improved genetic algorithm for the flowshop scheduling problem. *International Journal of Production Research*, 47(1):233–249, 2009.

[69] T.R. Ramanan, R. Sridharan, K.S. Shashikant, and A.N. Haq. An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(2):279–288, 2011.

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

[71] S. Reza Hejazi and S. Saghafian. Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929, 2005.

[72] I. Ribas, R. Companys, and X. Tort-Martorell. Comparing three-step heuristics for the permutation flow shop problem. *Computers and Operations Research*, 37(12):2062–2070, 2010.

[73] A. H. G. Rinnooy Kan. *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague, 1976.

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

[75] R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 34(5):461–476, 2006.

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

[77] M. Saravanan, A. Noorul Haq, A.R. Vivekraj, and T. Prasad. Performance evaluation of the scatter search method for permutation flowshop sequencing problems. *International Journal of Advanced Manufacturing Technology*, 37(11-12):1200–1208, 2008.

[78] M.K. Sayadi, R. Ramezanian, and N. Ghaffari-Nasab. 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, 2010.

[79] M. Solimanpur, P. Vrat, and R. Shankar. A neuro-tabu search heuristic for the flow shop scheduling problem. *Computers and Operations Research*, 31(13):2151–2164, 2004.

[80] T. Stütze. Applying iterated local search to the permutation flow shop problem. *Technical report, AIDA-98-04, FG Intellektik, FB Informatik, TU Darmstadt*, 1998.

[81] Y. Sun, C. Zhang, L. Gao, and X. Wang. Multi-objective optimization algorithms for flow shop scheduling problem: A review and prospects. *International Journal of Advanced Manufacturing Technology*, 55(5-8):723–739, 2011.

[82] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[83] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[84] M.F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz. 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, 2007.

[85] L.-Y. Tseng and Y.-T. Lin. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1):84–92, 2009.

[86] Y.-R. Tzeng and C.-L. Chen. A hybrid eda with acs for solving permutation flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 60(9-12):1139–1147, 2012.

[87] E. Vallada and R. Ruiz. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA, The International Journal of Management Science*, 38(1-2):57–67, 2010.

[88] E. Vallada, R. Ruiz, and J.M. Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240:666–677, 2015.

[89] D. Vasiljevic and M. Danilovic. Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, 35:1–9, 2015.

[90] J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.

[91] Z. Xie, C. Zhang, X. Shao, W. Lin, and H. Zhu. An effective hybrid teaching-learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*, 77:35–47, 2014.

[92] B. Yagmahan and M.M. Yenisey. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers and Industrial Engineering*, 54(3):411–420, 2008.

[93] K.-C. Ying and C.-J. Liao. An ant colony system for permutation flow-shop sequencing. *Computers and Operations Research*, 31(5):791–801, 2004.

[94] K.-C. Ying and S.-W. Lin. A high-performing constructive heuristic for minimizing makespan in permutation flowshops. *Journal of Industrial and Production Engineering*, 30(6):355–362, 2013.

[95] S.H. Zanakis, J.R. Evans, and A.A. Vazacopoulos. Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, 43(1):88–110, 1989.

[96] C. Zhang, J. Ning, and D. Ouyang. A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Computers and Industrial Engineering*, 58(1):1–11, 2010.

[97] C. Zhang and J. Sun. An alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 36(3 PART 1):5162–5167, 2009.

[98] C. Zhang, J. Sun, X. Zhu, and Q. Yang. An improved particle swarm optimization algorithm for flowshop scheduling problem. *Information Processing Letters*, 108(4):204–209, 2008.

[99] J. Zhang, C. Zhang, and S. Liang. The circular discrete particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37(8):5827–5834, 2010.

[100] L. Zhang and J. Wu. A PSO-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*, 2014.

[101] T. Zheng and M. Yamashiro. Solving flow shop scheduling problems by quantum differential evolutionary algorithm. *International Journal of Advanced Manufacturing Technology*, 49(5-8):643–662, 2010.

[102] G.I. Zobolas, C.D. Tarantilis, and G. Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers and Operations Research*, 36(4):1249–1267, 2009.

[103] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts*. Springer, 2010.