Ruiz García, R.; Pan, Q.; Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. Omega. 83:213-222. https://doi.org/10.1016/j.omega.2018.03.004

Iterated Greedy methods for the distributed permutation flowshop scheduling problem

Rubén Ruiz[a,*], Quan-Ke Pan[b], Bahman Naderi[c]

[a] *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.*
[b] *State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, PR China.*
[c] *Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran.*

Abstract

Large manufacturing firms operate more than one production center. As a result, in relation to scheduling problems, which factory manufactures which product is an important consideration. In this paper we study an extension of the well known permutation flowshop scheduling problem in which there is a set of identical factories, each one with a flowshop structure. The objective is to minimize the maximum completion time or makespan among all factories. The resulting problem is known as the distributed permutation flowshop and has attracted considerable interest over the last few years. Contrary to the recent trend in the scheduling literature, where complex nature-inspired or metaphor-based methods are often proposed, we present simple iterated greedy algorithms that have performed well in related problems. Improved initialization, construction and destruction procedures, along with a local search with a strong intensification are proposed. The result is a very effective algorithm with little problem-specific knowledge that is shown to provide demonstrably better solutions in a comprehensive and thorough computational and statistical campaign.

Keywords: distributed flowshop, makespan, metaheuristics, iterated greedy

*Corresponding author. Tel: +34 96 387 70 07. Fax: +34 96 387 74 99
Email addresses: rruiz@eio.upv.es (Rubén Ruiz), 2281393146@qq.com (Quan-Ke Pan), bahman_naderi62@yahoo.com (Bahman Naderi)

1. Introduction

Machine scheduling problems have been studied intensively for more than 60 years since the seminal work of Johnson (1954). Usually, production scheduling appears at an operational level inside operations management departaments within manufacturing companies. Resources, typically machines, are limited and costly and efficient scheduling of production activities contributes to profitability and higher customer satisfaction. The importance of optimized scheduling decisions is highlighted in Framinan et al. (2014); Pinedo (2016) or in McKay et al. (2002), among many others. In scheduling problems, clients' orders, lots or products to manufacture are modeled as jobs. Given the nature of production lines, one of the most studied production scheduling problems is the flowshop. In a flowshop problem we have a known number n of jobs, indexed by $j = \{1, \ldots, n\}$. The machines on the production floor are disposed in series and jobs start at machine 1, continue with machine 2 and go through the shop visiting all machines in the same order. Each job j requires a known amount of processing time p_{ij} at each machine i, $i = \{1, \ldots, m\}$. A job can visit the next machine in the sequence only after it has been completed in the previous machine. Machines cannot process more than one job at the same time and preemption is not allowed, i.e., once started at any machine, jobs cannot be interrupted. The completion time of a job C_j denotes the time at which the job is finished at the last machine m. With this in mind, the most commonly studied objective in the flowshop literature is the minimization of the maximum completion time, commonly referred to as makespan or $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$. Flowshop problems have been thoroughly studied judging from the many reviews available (Reisman et al., 1997; Framinan et al., 2004; Ruiz and Maroto, 2005; Hejazi and Saghafian, 2005; Gupta and Stafford, 2006; Fernandez-Viagas et al., 2017). The flowshop problem or FSP with $C_{\max}$ criterion is $\mathcal{NP}$-Complete in the strong sense (Garey et al., 1976). There are $n!$ possible sequences for each machine and a total of $(n!)^m$ solutions to the problem. A more constrained version is the Permutation Flowshop Scheduling Problem or PFSP by which the same permutation of jobs is maintained for all machines and hence $n!$ solutions are explored. This much more common variant is denoted as $F/prmu/C_{\max}$ (Graham et al., 1979; Pinedo, 2016) and belongs to the same complexity class as the general flowshop.

In order to bridge the gap between practical and academic scheduling (McKay et al., 1988; MacCarthy and Liu, 1993), authors have been studying extensions

of the flowshop problem so as to tackle more and more realistic problems. One such extension is the Distributed Permutation Flowshop Problem or DPFSP proposed for the first time by Naderi and Ruiz (2010). The DPFSP explores a key issue in modern manufacturing which is the fact that large companies have several production centers and one has to decide the production allocated to each factory along with the scheduling. Distributed manufacturing is a hot topic today and several studies approach the many implications of having more than one factory. Interested readers are referred to Chan and Chung (2013) for a survey. It is therefore of interest to study distributed variants of flowshop problems. More specifically, in the DPFSP (denoted as $DF/prmu/C_{\max}$ by Naderi and Ruiz, 2010), there are F identical factories each one with the same m machine flowshop. The processing times for the jobs do not change from factory to factory. The additional dimension is to decide which jobs should go to each factory with the objective of minimizing the global makespan, i.e., minimizing the maximum makespan among the F factories. Since typically $n >> F$ the problem is far from trivial. It follows that the DPFSP is also an $\mathcal{NP}$-Hard problem as the specific case where $F = 1$ is the well known PFSP.

Since the initial work of Naderi and Ruiz (2010), there has been a lot of interest in this problem. While Naderi and Ruiz (2010) only proposed some mathematical models and some local search approaches, other authors presented more advanced algorithms and metaheuristics, including some very highly performing methods. Of particular interest are the simple yet very effective Iterated Greedy (IG) algorithms from Ruiz and Stützle (2007). Despite its simplicity, IG has demonstrated state-of-the-art performance for many different flowshop problems and variants (Fernandez-Viagas et al., 2017). Unlike complex metaphor-based methods –currently subject to severe criticism, (Sörensen, 2015)–, IG is simply an iterated search method with no memory and few structures. It is very easy to code and to understand. Results are easy to replicate and to extend to other problems, in which lies its appeal. Some authors have already presented IG methods for the DPFSP, most notably Lin et al. (2013) and Fernandez-Viagas and Framinan (2015), with good results. However, recently some more advanced IG methods have been proposed. Pan et al. (2017) postulated that hybrid scheduling problems require the combination of different search strategies within an IG. These ideas have been explored before for other problems and settings, like in Urlings et al. (2010). Given the excellent results obtained by these papers, the idea of proposing IG methods with different search strategies in the DPFSP seems promising

and hence is the main objective of this paper. As we will show, the two stage IG method proposed in this paper is able to significantly outperform all other presented approaches by a wide margin, while maintaining most of its simplicity.

The rest of the paper is organized as follows. The next section reviews the existing literature for the DFPSP. Section 3 presents the IG methods which are calibrated and thoroughly computationally tested in Section 4. Section 5 concludes the paper and proposes lines of further research.

2. Literature review

As mentioned, from the initial paper of Naderi and Ruiz (2010) on the DPFSP there has been a large number of publications focusing on this problem. Naderi and Ruiz (2010) introduced some simple constructive heuristics, basically by adding factory assignment rules to the well known NEH heuristic of Nawaz et al. (1983). Also, two Variable Neighborhood Descent (VND, Mladenović and Hansen, 1997) procedures were given, VND(a) being the best of the two while at the same time using small CPU times. A rather convoluted Electromagnetism Method (EM) procedure combining several local search neighborhoods was presented in Liu and Gao (2010). A Hybrid Genetic Algorithm (HGA) was proposed by Gao and Chen (2011a). An improved version of the NEH2 of Naderi and Ruiz (2010) was put forward by Gao and Chen (2011b). An improved GA was introduced by Gao et al. (2012b) and another improved VND by Gao et al. (2012a). A Tabu Search method was proposed by Gao et al. (2013) and a rather complex Estimation of Distribution Algorithm (EDA) was published by Wang et al. (2013). The same year Lin et al. (2013) presented an IG method.

Almost all of the aforementioned algorithms were independently recoded and tested against a Scatter Search (SS) procedure in Naderi and Ruiz (2014). A total of 11 methods were compared using the 720 original instances of Naderi and Ruiz (2010) which are, in turn, based on the well known 120 instances of Taillard (1993). In a comprehensive computational and statistical test (requiring almost 165 days of CPU time), the SS procedure was shown to statistically outperform all other compared methods. As a matter of fact, SS improved 719 of the best known original results of the 720 instances of Naderi and Ruiz (2010) with an average CPU time of just 0.61 seconds. For larger CPU times of 1.22 seconds on average, all 720 best known results were improved. Xu et al. (2014) introduced a hybrid immune algorithm (HIA). The

authors claimed it improved 585 out of the 720 instances of Naderi and Ruiz (2010). However, the only comparison carried out is against VND(a) for which better results are obtained, at the expense of significantly larger CPU times. Fernandez-Viagas and Framinan (2015) presented an advanced IG procedure, referred to as BSIG. This method incorporates several different local search procedures. To reduce the CPU time used in these searches, some clever properties of the DPFSP are exploited in the form of restricted (bounded) local search procedures which save almost a third of CPU time. While BSIG was not compared against SS, it was compared against the EDA of Wang et al. (2013), the IG of Lin et al. (2013) and the best methods presented in Naderi and Ruiz (2010). The results provided show the clear superiority of BSIG over the other three tested methods. The authors also improved 263 of the original 720 best known solutions. More recently, other authors have presented new methodologies. Bargaoui et al. (2016) have proposed a rather bizarre "Chemical Reaction Optimization" or CRO. The authors test their proposed approach against VND(a) of Naderi and Ruiz (2010) and the HGA of Gao and Chen (2011a). CRO is shown to work better than the other two methods in up to 50 jobs but it is bested by HGA on the large instances. The authors did not test their approach against the SS of Naderi and Ruiz (2014) or the BSIG of Fernandez-Viagas and Framinan (2015). Given the previous results, where the SS of Naderi and Ruiz (2014) was shown to clearly outperform the HGA of Gao and Chen (2011a) (average relative percentage deviation found at Table 2 of Naderi and Ruiz, 2014 for HGA being 3.5% vs. the deviation of SS being 1.31%) it is clear that CRO is outperformed by SS and BSIG. Another recent method is a Hybrid Discrete Cuckoo Search (HDCS) method by Wang et al. (2016a). The algorithm includes a number of operators such as crossover and several local search procedures among others. HDCS is compared against the EDA of Wang et al. (2013) and all of the methods from Naderi and Ruiz (2010). HDCS is shown to be slightly better than EDA. Once again, Naderi and Ruiz (2014) showed SS to be superior to EDA in their evaluations so it is safe to assume that HDCS is also outperformed.

From this review, an important question is how BSIG and SS compare. Data indicates that BSIG should outperform SS but as with all previous results, only a fair comparison using the same codes, same computer and same stopping criterion can tell. HIA seems to be comparable in performance to the EDA of the same authors but it has not been compared either. Additionally, BSIG includes three different local searches and considers some problem-specific knowledge of the DPFSP that allows for faster local search schemes. An

important research question is if better performance can be obtained with simple versions of the IG procedure.

Note that there are other recent papers proposed for variations of the DPFSP like the assembly variant (Hatami et al., 2013, 2015), multi-objective (Rifai et al., 2016; Deng and Wang, 2017), machine breakdowns (Wang et al., 2016b), and blocking (Ribas et al., 2017), etc. However, for the sake of brevity, these are not comprehensively reviewed here. The interested reader can see updated reviews with many more references (not limited to flowshops) published by Wang et al. (2016c) and Behnamian and Fatemi Ghomi (2016).

3. Iterated Greedy Procedures

As mentioned, the Iterated Greedy (IG) algorithm of Ruiz and Stützle (2007) is among the best methods for many different flowshop problems. Furthermore, it is very simple. Figure 1 shows the basic outline of the IG.

procedure Iterated_Greedy
 π_0 := GenerateInitialSolution
 π := LocalSearch(π_0)
 while (termination criterion not satisfied) **do**
  π_D := Destruction(π)
  π' := Reconstruction(π_D, π_R)
  π'' := LocalSearch(π')
  π := AcceptanceCriterion(π'', π)
 endwhile
end

Figure 1: Iterated Greedy (IG) algorithm of Ruiz and Stützle (2007).

In a nutshell, IG uses a high performing heuristic to initialize the search. Most of the IG literature employs variants of the NEH procedure of Nawaz et al. (1983). Some form of local search is applied to this initial solution. The main loop of the IG is the iterative application of four operators. 1) Destruction, where the incumbent solution is partially destroyed, i.e., some elements of the solution are removed, resulting in two partial permutations, one containing the jobs that have been removed (π_R) and another one with the leftover jobs from the original permutation (π_D). 2) Reconstruction, where a heuristic (usually a greedy one) is applied in order to reintroduce the removed elements back into the solution resulting in a new complete solution. 3) Local search is used again to improve this newly reconstructed solution. 4)

Finally, a decision about accepting the new solution has to be made. If the new solution is better than the best solution obtained so far it is obviously accepted. However, it might be the case that the new solution is not better than the incumbent and yet still be accepted probabilistically, like in simulated annealing, so as to be able to explore other regions of the solution space.
Most proposed IG methods for scheduling problems introduce variations in the aforementioned operators. In the case of the DPFSP, the IG of Lin et al. (2013) proposed two main changes. First, the elements to remove in the destruction operator are not fixed but controlled during the algorithm. Second, the acceptance criterion contains a more elaborate procedure rather than the fixed temperature simulated annealing-like method of Ruiz and Stützle (2007). The BSIG of Fernandez-Viagas and Framinan (2015) introduces changes in the local search procedure by applying three different local search operators consecutively. Most of the other operators are very similar or identical to those of the original IG of Ruiz and Stützle (2007), in particular, IG and BSIG employ the NEH2 initialization procedure, reconstruction and many of the operators of Naderi and Ruiz (2010)
In what follows we describe different proposed IG alternatives that expand on the original initialization of Naderi and Ruiz (2010) and different destruction, reconstruction and local search operators. Finally we present a two stage IG exploiting the hybrid nature of the DPFSP.

3.1. Representation and initialization procedure

As for the solution representation, most of the DPFSP literature employs the original representation introduced by Naderi and Ruiz (2010). This is simply a set of F lists, one per factory. Each list contains the jobs assigned to each factory, in the order in which they have to be processed. This representation is very efficient and we also employ it in this paper.

As for the initialization procedure we test the NEH2 procedure of Naderi and Ruiz (2010) but with a simple extension. After inserting a job into the best position among all the F factories, either the previous or the following job (at random) is extracted and tested in all positions in the same factory. This is somehow similar to the FRB4$_k$ of Rad et al. (2009) or more precisely, it would be a DPFSP adaptation of an FRB4$_{\frac{1}{2}}$ inspired by Pan and Ruiz (2014). We also employ the accelerations presented in Fernandez-Viagas and Framinan (2015) (BSIG). We refer to this NEH2 improvement as NEH2_en and is depicted in Figure 2. Note that the worst case computational complexity is

the same as NEH2 ($\mathcal{O}(n^2mF)$).

We tried more extensive solution initialization procedures, for example a

procedure NEH2_en
  Calculate $P_j = \sum_{i=1}^{m} p_{ij}, \forall n \in N$
  π^{LPT} := Sort the n jobs according to P_j in decreasing order
  for $f := 1$ **to** F **do** $\pi_f := \varnothing$ % (empty initial solution)
  $\pi := \{\pi_1, \pi_2, \ldots, \pi_F\}$
  for $step := 1$ **to** n **do**
    $j := \pi^{LPT}[step]$
    for $f := 1$ **to** F **do**
      Test job j in all possible positions of π_f % (Taillard-BSIG accelerations)
      $C^f_{\max}$ is the lowest $C_{\max}$ obtained
      p^f is the position where the lowest $C_{\max}$ is obtained
    endfor
    $f_{\min} = \arg\left(\min_{f=1}^{F}(C^f_{\max})\right)$
    Insert job j in $\pi_{f_{\min}}$ at position $p^{f_{\min}}$ resulting in the lowest $C_{\max}$
    Extract at random job h from position $p^{f_{\min}} - 1$ or $p^{f_{\min}} + 1$ from $\pi_{f_{\min}}$
    Test job h in all possible positions of $\pi_{f_{\min}}$ % (Taillard-BSIG accelerations)
    Insert job h in $\pi_{f_{\min}}$ at the position resulting in the lowest $C_{\max}$
  endfor
end

Figure 2: Procedure NEH2_en to generate an initial solution.

full FRB4_2 but observed little effect. After all, in IG algorithms, the effect of the initial solution is quickly neutralized by the local search procedures. This is not to say that a random or low quality solution is enough but rather that once a high quality starting solution is obtained, there is little to be gained in spending more CPU time on improving this initial solution. Still, for large or difficult instances, the effect of the solution initialization is clearly measurable. This was recently observed by Fernandez-Viagas et al. (2017) where elaborate constructive heuristics are outperformed by the combination of a simple constructive method and a few iterations of IG methods. In any case, in later sections we will statistically test the effect of employing the regular NEH2 or the proposed NEH2_en initialization procedures.

It is important to note that similar to most previous DPFSP literature, all insertion procedures employ the well known Taillard accelerations of the insertion neighborhood in order to significantly increase the speed of the procedures. Interested readers are referred to Taillard (1990) for more details.

3.2. Destruction

We test three different destruction operators. The first one is the original from Ruiz and Stützle (2007) in which a given number d of jobs are randomly extracted (each one from its factory) and put into a list π_R of jobs that have been removed. The number d has to be calibrated. This is identical to the destruction operator used in the BSIG of Fernandez-Viagas and Framinan (2015).

The second operator is the one employed by the IG of Lin et al. (2013). As mentioned, the authors employed a destruction operator where d varies randomly. More specifically, the destruction operator works as follows: d is randomly set following a uniform distribution between 2 and 7 (as per their original experiments) at each iteration. Then, the F factories are sorted from highest to lowest $C_{\max}$. A random job is extracted from the factory with the highest $C_{\max}$, another job is randomly extracted from the factory with the second highest $C_{\max}$ and so on. The procedure stops if $d \leq F$. If $d > F$ the remaining $d - F$ jobs are extracted at random from the F factories. Basically, the destruction is biased towards the factories with largest $C_{\max}$ values.

The third operator is a new one. We prefer simple operators that do not result in a significantly more complex IG algorithm. It makes sense to bias the destruction towards factories with larger $C_{\max}$ values. Therefore, the third operator simply removes, at random, $d/2$ jobs from the factory with the largest $C_{\max}$ value (there is no attempt at tie breaking if there is more than one factory with the largest $C_{\max}$) and the remaining $d/2$ jobs are randomly removed from the remaining $F - 1$ factories, i.e., excluding the factory with the largest $C_{\max}$ value.

3.3. Reconstruction

We tested a simple reconstruction based on the original IG of Ruiz and Stützle (2007), which is, in turn, similar to the reconstruction employed by the IG of Lin et al. (2013) and the BSIG of Fernandez-Viagas and Framinan (2015). The procedure is simple, every job in π_R that was removed during the destruction phase is tested in every position in all the F factories, using Taillard's accelerations. Note that BSIG uses the bounded search procedure to save on some insertions. However, we again found that the previously detailed procedure NEH2_en yielded better results (including the BSIG accelerations). Basically, after the next job in π_D is placed in the best position in the best factory (with the least $C_{\max}$ increase), we randomly extract either the previous or the following job and this job is tested in all positions in the factory. This

small change was able to improve solutions, especially in the hardest cases as will be shown later.

3.4. Local search methods

Akin to what Fernandez-Viagas and Framinan (2015) carried out in the BSIG procedure, we significantly improve upon the basic insertion local search of the original IG of Ruiz and Stützle (2007) as this is clearly key in improving results in the DPFSP. However, in order to test these improvements, we try three local search procedures. Note that we use BSIG accelerations whenever possible in all tested methods.

The first one is the VND(a) presented originally in Naderi and Ruiz (2010). This VND is composed of two neighborhoods: LS_1 and LS_2. LS_1 reinserts all jobs within each factory using the insertion neighborhood, i.e., each job in each factory is extracted and tested in all possible positions of its corresponding factory list. In LS_2 all jobs from the factory generating the $C_{\max}$ are extracted and inserted into all positions in all the other factories. These movements have a special acceptance criterion. Basically, a movement is accepted if the makespan of the complete problem is improved. This local search will be referred to as VND(a) to respect the original name given in Naderi and Ruiz (2010)

The second local search tested is a seemingly minor but nevertheless important variation of VND(a). In both LS_1 and LS_2 jobs are extracted in order, starting from job 1, then job 2 and so on until the last job in each factory. Both local searches are applied until there are no improvements in the $C_{\max}$ value, i.e., if an improvement move is carried out, the search starts again from the first job and the search finishes when all jobs have been reinserted with no gains. Also, if an improvement is found in LS_2, the search goes back to LS_1 and hence the VND structure. For the second local search procedure tested, jobs are not extracted in order but at random and without repetition. This is a fundamental change that transforms the local search into a first-improvement pivoting rule. Therefore, if there are n_f jobs in a given factory f, we carry out, in a single pass of LS_1 or LS_2, n_f iterations. At each iteration, a random job (without repetition) is extracted and tested in all positions in the same factory (in the case of LS_1) or in all other factories (LS_2). This second local search is slightly more complex to code but has the same computational complexity and, as we will see, produces better results. This second local search will be referred to as VND(a)$_R$ to denote the random extraction of jobs.

The third local search tested incorporates parts of the previous LS_1 and LS_2. We refer to this method as LS3. It considers the dimensions of factory assignment and job sequencing simultaneously. The factory generating the $C_{\max}$ is selected. A job is randomly extracted from this factory and is inserted into all possible positions in all factories (including the one generating the makespan). If the best $C_{\max}$ in all these insertions is better than the starting $C_{\max}$, the job is relocated and the search starts again from the beginning otherwise the job is reinserted back into its original position and the search continues. The procedure iterates until all jobs from the factory generating the $C_{\max}$ have been tested (at random and without repetition). Note that LS3 does not deal with two neighborhoods in an VND loop and it is, therefore, faster as regards CPU time requirements. A pseudocode of LS3 is given in Figure 3

procedure LS3($\pi = \{\pi_1, \pi_2, \ldots, \pi_F\}$)
  $C^*_{\max} = \max^F_{f=1}\{C_{\max}(\pi_1), C_{\max}(\pi_2), \ldots, C_{\max}(\pi_F)\}$
  $f_{\max} = \arg(C^*_{\max})$ % (factory with the largest $C_{\max}$)
  $Cnt := 0$
  while $Cnt < |\pi_{f_{\max}}|$ **do** % (all jobs in factory $f_{\max}$)
    Randomly extract, without repetition, a job j from position k of $\pi_{f_{\max}}$
    for $f := 1$ **to** F **do**
      Test job j in all possible positions of π_f % (Taillard-BSIG accelerations)
      $C^f_{\max}$ is the lowest $C_{\max}$ obtained
      p^f is the position where the lowest $C_{\max}$ is obtained
    endfor
    $f_{\min} = \arg\left(\min^F_{f=1}(C^f_{\max})\right)$
    if $C^f_{\max} < C^*_{\max}$ **then**
      Place job j at position p^f of factory $f_{\min}$
      $C^*_{\max} = \max^F_{f=1}\{C_{\max}(\pi_1), C_{\max}(\pi_2), \ldots, C_{\max}(\pi_F)\}$
      $f_{\max} = \arg(C^*_{\max})$ % (factory with the largest $C_{\max}$)
      $Cnt := 0$
    elseif
      Return job j to position k of $f_{\max}$
      $Cnt := Cnt + 1$
    endif
  endwhile
end

Figure 3: Third local search procedure LS3.

3.5. Acceptance criterion

Ruiz and Stützle (2007) proposed a very simple constant temperature

acceptance criterion based on a parameter T as follows: Temperature = $T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$, where T is to be calibrated but has been shown to be rather robust (mostly any value different from zero and not overly high works well). Other authors, most notably Hatami et al. (2013) and Hatami et al. (2015) have presented acceptance criteria without parameters. However, our initial testing did not yield significant improvements using these alternate acceptance criteria and we decided to stick with the classic acceptance criterion proposed in Ruiz and Stützle (2007).

3.6. Two stage iterated greedy

As we will later show in the computational tests, the previous improved initialization, destruction and reconstruction operators, along with LS3 improved the best known solutions for the DPFSP. However, we detected that all IG variants that we tested reached a point at which no improvements could be found. Upon closer inspection we reached the conclusion that the iteration of the destruction-reconstruction-local search operators is not enough to reduce the $C_{\max}$ in the factory that has the largest makespan value among all factories. We started experimenting with nested and two-stage Iterated Greedy methods. A nested Iterated Greedy is basically an IG within an IG. Of course, applying a whole IG is much more CPU time consuming than a few iterations of a local search procedure. Therefore, such approaches have to be applied with caution as otherwise valuable CPU time is lost. Let us first describe the nested/two stage IG that we propose and then we detail how it was incorporated into the regular IG.

The nested/two stage IG is applied to the factory generating the $C_{\max}$ and only to this factory. All the operators are applied to this factory and are a simplification of all previously detailed methods. Destruction is the original Ruiz and Stützle (2007) procedure by which d jobs are removed, at random, from the factory generating the $C_{\max}$ and introduced into a partial permutation of removed jobs π_R. The remaining jobs stay in the partial solution π_D. The reconstruction is a random and improved variant of the original reconstruction of Ruiz and Stützle (2007). A random job is selected from π_R and tested in all positions in the partial solution π_D. The job is placed into the position resulting in the best partial $C_{\max}$. Similarly to the aforementioned reconstruction operator and to the NEH2_en of Figure 2, the job in either the previous or next position (at random) is extracted and tested

in all possible positions of π_D, and is finally placed in the position resulting in the lowest partial $C_{\max}$. The procedure continues until π_R is empty. After this destruction-reconstruction we apply a simple local search, which is the random variant of LS_1 detailed in Section 3.4. The acceptance criterion is very simple, as the new solution for the $C_{\max}$ generating factory is only accepted if the $C_{\max}$ value is improved. Again, BSIG accelerations are used throughout the methods.

We tried several possibilities/combinations of this nested/second stage IG. The first and obvious one is to apply the proposed IG and the second stage IG sequentially at each iteration, i.e., first carry out an iteration of the proposed IG and then the second stage that focuses on the $C_{\max}$ generating factory. The second straightforward nested IG is the case in which the nested IG is applied only when a new best solution is found. Another option is to set up a probability and apply the nested IG at each iteration according to this probability. We tried all these as well as several other possibilities. Our results were mixed and not as promising as expected. The main reason is that the nested IG consumes a lot of time (we tested it applying it for a few iterations and for a limited time) when compared to the local search steps and therefore the total number of regular IG iterations that one can do is severely reduced. Additionally, it makes relatively little sense to carry out a deep intensification with the nested IG over a solution that is to be improved later after a few iterations. In a sense, all this CPU time is wasted. In the end we found that it was better to apply the proposed IG for a portion ρ of the given CPU time and for the remaining $1-\rho$ fraction of the CPU time apply the second stage simplified IG (working over the $C_{\max}$ generating factory). With this very basic two stage IG we obtained better solutions in most situations. Basically, we let the original IG do most of the work and in the last moments a few iterations of the second stage IG allow for a few key improvements to get a very high quality final solution.

4. Computational and statistical experimentation

Metaheuristics in general, and Iterated Greedy methods in particular, need comprehensive computational and statistical testing to validate the results. Following the excellent paper of Kendall et al. (2016) we apply several of their good laboratory practices to ensure the maximum reproducibility of the

presented approaches and generalization of results. In the following sections we give comprehensive details of all the experiments carried out.

4.1. Experimental settings and tested methods

Naderi and Ruiz (2010) presented 420 small instances of up to 16 jobs, 5 machines and 4 factories. They also proposed 720 larger instances after adding the number of factories F to the 120 well known instances of Taillard (1993). There are 6 sets with values of $F = \{2, 3, 4, 5, 6, 7\}$. For each set we have the 120 instances of Taillard ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. In Naderi and Ruiz (2014) the authors added a set of 50 different instances intended for calibration with n, m and F values randomly sampled from the set of 720 instances. Small instances are easily solved by most existing metaheuristics for the DPFSP and therefore are not used in this work. We will use the large test and calibration instances which are, along with the new best solutions that have been obtained over the course of this paper, available for download at `http://soa.iti.es`.

We test two variants of the proposed IG method. The first one is the single stage IG, referred to as IG1S. The second is the two stage IG or IG2S. Both methods need calibration, which will be detailed in the next section. We will also study a vanilla variant of IG1S which uses the regular NEH2 initialization procedure and the VND(a)$_R$ local search. This version, referred to as IG1S$^-$ is used to assess the contribution of the improved initialization and LS3 local search present in IG1S. In order to fully clarify these variants, we detail the operators used for these algorithms in Table 1.

Algorithm Variant	Two Stage	Initialization	Local Search
IG1S$^-$	No	NEH2	VND(a)$_R$
IG1S	No	NEH2_en	LS3
IG2S	Yes	NEH2_en	LS3

Table 1: Variants of the proposed algorithm along with their operators (all other not mentioned operators are the same).

We test these proposed methods against the HIA of Xu et al. (2014), SS of Naderi and Ruiz (2014) and BSIG of Fernandez-Viagas and Framinan (2015). Note that the proposd methods, as well as the BSIG, use the accelerations proposed in Fernandez-Viagas and Framinan (2015)so as to keep all methods as close to their original versions as possible. As explained in Section 2, all

other existing methods from the literature have been shown to be outperformed by either SS or BSIG and therefore it is not necessary to test them.

All compared algorithms HIA, BSIG, SS, IG1S, IG1S$^-$ and IG2S have been coded in C++ language and have been compiled with Visual Studio 2015 with the x64 compiler with all optimization flags enabled. All methods use the same important functions in the codes, like Taillard's accelerations. HIA was independently recoded by the authors. For BSIG, the original authors were extremely supportive and provided us with source codes in C# which allowed us to recode their method with great accuracy in C++. We ran their version against ours and we found our C++ code to be more effective, most probably due to C++ compiler generating faster code than C#.

As a response variable in the experiments we measure the Relative Percentage Deviation (RPD) as follows: $RPD = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$. Here $Some_{sol}$ is the $C_{\max}$ obtained by an algorithm for a given instance and $Best_{sol}$ is the best known $C_{\max}$ value for the same instance. All tested metaheuristic methods need a stopping criterion. This is set as a maximum elapsed CPU time following the expression $n \cdot m \cdot C$ milliseconds where C is a parameter that will be tested at several levels. For the calibration experiments, we set $C = 10$. For the comparisons among the algorithms we test 5 levels for C: 20, 40, 60, 80 and 100. Considering that the smallest instances have 20 jobs and 5 machines and the largest instances 500 jobs and 20 machines, the range of C values translates into CPU times as short as 2 seconds for $C = 20$ and the smallest instances all the way up to 1000 seconds for the largest instances and $C = 100$. All runs are independent, i.e., each run is started from scratch (5 separated runs for each C value). All calibration instances are run for 5 different replicates, maintaining the random seed value for each replicate as a variance reduction technique. Similarly, for the comparisons we run 10 independent replicates.

The experiments in this paper are performed on virtual machines with 2 virtual processing cores and 8 GBytes of RAM running Windows 10 Enterprise 64 bits operating system. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each one with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz. and 256 GB of RAM, for a total of 576 cores and 3 TBytes of RAM. There is no parallel computing, just a random distribution of all computations among the

virtual machines so as to speed up the completion of all the experimentation. Considering that all algorithms share most of the code, are coded in the same language, compiled in the same environment and run on the same hardware for the same length of CPU time we can conclude that the comparisons are fair.

The proposed experimental settings imply a massive undertaking. If we consider only the final testing, each metaheuristic algorithm is run for 10 replicates on each one of the 720 instances and for 5 values of C. Therefore, there are 36,000 results for each algorithm and 216,000 results in total. The total CPU time employed for testing all algorithms is almost 6,600 hours. The complete results, summary files, logs and excel files are available as accompanying online materials.

4.2. Calibration of the proposed IG methods

IG1S and IG2S have several parameters that might affect their performance. In particular, IG1S has 5 controlled factors. These are 1) The type of initialization (InitType), tested at 2 variants, NEH2 and NEH2_en, 2) Type of destruction (DType) tested at 3 variants, original, Lin (from Lin et al., 2013) and new, 3) Type of local search (LSType), tested at 3 variants, VND(a), VND(a)$_R$ and LS3, 4) number of jobs to remove in the destruction (d), tested at 3 levels, 4, 5, and 6 and finally 5) Temperature (T) tested at 3 levels, 0.1, 0.2 and 0.4. Considering all possible combinations there are $2 \times 3 \times 3 \times 3 \times 3 = 162$ possible IG1S configurations. Each configuration is run for 45 independent replicates on the 50 calibration instances. Therefore, the total number of results is $162 \cdot 45 \cdot 50 = 364,500$. The instance factors n, m and F are considered as noise factors and are not controlled in the experiment in order to avoid an instance-specific calibration. Recall that all algorithm configurations are run for the same CPU time with $C = 10$ and that the response variable is the RPD.

IG2S shares the factors of IG1S but with some additional levels and two more additional factors. More specifically, factors InitType, DType and LSType are the same as in IG1S. However, factor d is tested at 4 levels, 4, 5, 6, and 7 and factor T also at 4 levels, 0.1, 0.2, 0.3 and 0.4. There are two new factors, the proportion of CPU time that we give to the first stage (ρ) tested at three levels, 0.9, 0.95 and 1.0 (note that this last level means that there is no second stage, so it will help us in assessing the contribution of the second stage) and the factor of the number of jobs to remove in the second stage ($d2$) tested at three levels, 2, 4, and 6. In total there are $2 \times 3 \times 3 \times 4 \times 4 \times 3 \times 3 = 2,592$

IG2S configurations. This time, only 5 replicates are needed for a total of $2,592 \cdot 5 \cdot 50 = 648,000$ results.

The set of results is scrutinized with a powerful statistical tool which is the Design of Experiments (DOE) coupled with the Analysis of Variance (ANOVA) technique (Montgomery, 2012). ANOVA is based on a statistical model and is therefore parametric. Three main hypotheses have to be met: normality, homoskedasticity and independence of the residuals. In computer experimentation these hypotheses are easy to accept. ANOVA is frequently used in the scheduling literature to calibrate metaheuristics. Even though we are analyzing a large number of results with a powerful technique, the process is far from being a fine tuned calibration as we are basically using a simple full factorial design of experiments. The interested reader can refer to Bartz-Beielstein et al. (2010) for a detailed description of far more advanced methodologies.

Detailed ANOVA tables are omitted due to space considerations. Instead, we provide as online materials the complete datasets resulting from the calibration. Figure 4 shows the relevant means plots with 95% Tukey's Honest Significant Difference (HSD) confidence intervals for IG1S. It has to be stressed that if the confidence intervals among two means overlap, the observed difference among the overlapped means is not statistically significant and is therefore, meaningless. The means plots of Figure 4 are presented

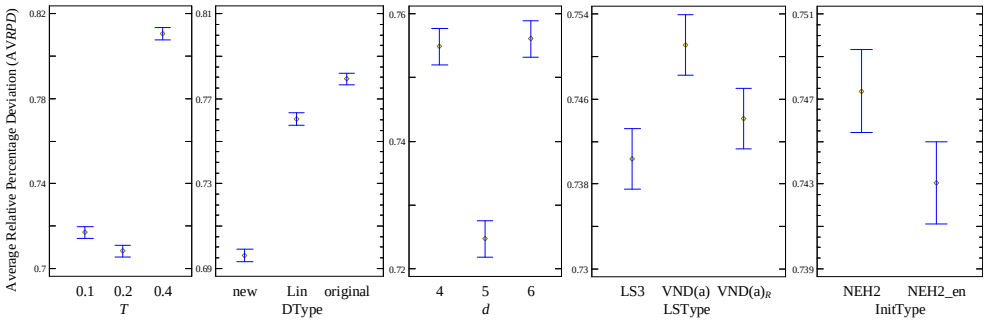

Figure 4: Means plots for all the factors in the ANOVA IG1S calibration. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

from left to right in order of significance. Therefore, factor T is the most statistically significant with 0.2 being the best value. The next factor is the type of destruction operator DType. We can see how the destruction operator

of Lin et al. (2013) is indeed better than the original operator of Ruiz and Stützle (2007) but the new proposed operator is significantly better. Considering that the new destruction operator is much simpler, this is a very interesting outcome. The number of jobs to remove d is set at 5 as it is the best value. For the local search operator LSType we observe that LS3 and VND(a)$_R$ are statistically better than VND(a). Even though in the overall plot VND(a)$_R$ and LS3 partially overlap, for the hardest instances and some settings, LS3 is statistically better. Therefore, we select LS3 as the local search in IG1S. Finally, we see that we obtain a slightly better performance with a statistically significant difference if the improved NEH2_en initialization is used. All these results show that the proposed improvements over the regular IG methodology pay off and better solutions are obtained. We will check later how IG1S compares with SS or BSIG.

Carrying out a similar analysis for IG2S we obtain the means plots of Figure 5. For IG2S, the most significant factor is the type of destruction which

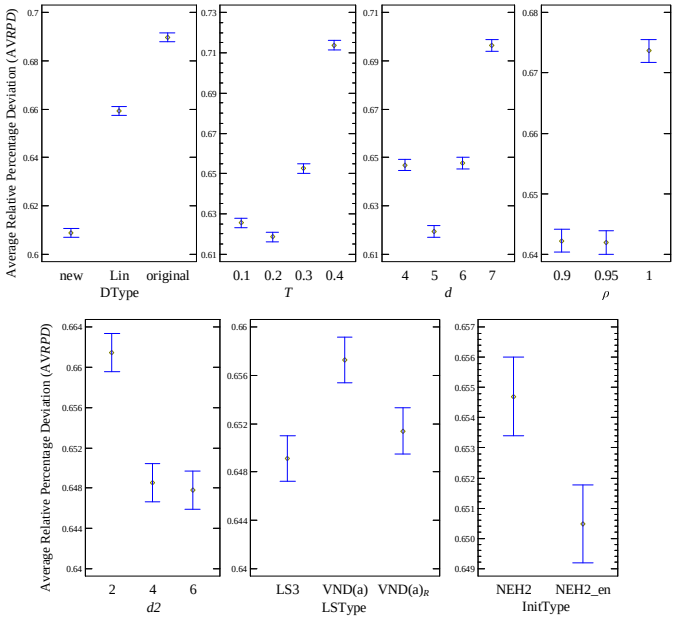

Figure 5: Means plots for all the factors in the ANOVA IG2S calibration. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

again shows that the proposed destruction is statistically superior. T is also set at 0.2 and d to 5 as in the IG1S calibration. Of interest is the means plot

of the ρ factor. The plot shows that removing the second stage ($\rho = 1$) results in an algorithm which is clearly worse. The best value is $\rho = 0.95$, i.e., only 5% of the allotted CPU time is dedicated to the second stage IG. The number of jobs to remove in the second stage $d2$ is set to the best value 6. Also, as in the previous IG1S calibration, the best local search and initialization operators are LS3 and NEH2_en respectively.

4.3. Comparison of methods

We now test the calibrated versions of IG1S and IG2S against the HIA of Xu et al. (2014), the SS of Naderi and Ruiz (2014) and the BSIG of Fernandez-Viagas and Framinan (2015). Recall that there are 5 values of C, 720 instances and that each algorithm is run 10 independent times on each instance. Table 2 shows the Average Relative Percentage Deviation for each method, grouped per C and F. Each cell within the table is the average RPD over 1200 results. The complete spreadsheets with all detailed results are available as on-line materials. Naderi and Ruiz (2014) used a CPU time termination criterion following the expression $n \cdot m \cdot F \cdot C$ and therefore they used from 2 to 7 times more CPU time than in our results. Still, SS has not improved results significantly, with a global $AVRPD$ of 1.59% in this work compared to the 1.31% of Naderi and Ruiz (2014). The difference is due to the improved best solutions in this work which increase the relative deviations. One striking result is the poor performance of HIA with a global $AVRPD$ of almost a 10%. Our hypothesis is that HIA starts from a random solution and it is therefore hard for HIA to converge to good solutions in a limited CPU time. Note however that the average CPU time when $C = 60$ is about 110 seconds, very similar to the 101 seconds reported by Xu et al. (2014) in their paper. Still, for $C = 60$ the $AVRPD$ of HIA is 9.78% which is not competitive.

Our results confirm that BSIG is significantly better than SS. Actually, the global $AVRPD$ of BSIG is about half. If we take the best result obtained for all 10 replicates of each algorithm, BSIG obtains, over the 720 instances, 580 better results than SS. Both methods tie in 135 cases and only in 5 cases does SS give a better result. Therefore, BSIG is better than SS by a wide margin. Furthermore, BSIG is much simpler and easier to code. While the accelerations present in BSIG could be implemented into the SS, this would not be enough to make SS competitive. As shown in Fernandez-Viagas and Framinan (2015), accelerations basically reduce CPU times by a third and we observe that BSIG has an $AVRPD$ of 0.97% for $C = 20$ while SS gives a

C	F	HIA	SS	BSIG	IG1S$^-$	IG1S	IG2S
20	2	8.28	1.05	0.60	0.54	0.52	**0.50**
	3	10.26	1.57	0.83	0.65	0.62	**0.59**
	4	11.04	1.82	0.98	0.68	0.62	**0.61**
	5	11.39	2.03	1.14	0.70	0.65	**0.63**
	6	11.31	2.14	1.10	0.70	0.64	**0.62**
	7	10.94	2.21	1.19	0.72	0.65	**0.63**
Average		10.54	1.80	0.97	0.66	0.62	**0.60**
40	2	7.98	0.93	0.46	0.43	0.41	**0.39**
	3	9.81	1.39	0.70	0.51	0.47	**0.45**
	4	10.46	1.63	0.83	0.51	0.47	**0.46**
	5	10.77	1.85	1.01	0.53	0.49	**0.47**
	6	10.83	1.96	0.95	0.53	0.48	**0.46**
	7	10.53	2.06	1.05	0.55	0.48	**0.46**
Average		10.06	1.64	0.83	0.51	0.47	**0.45**
60	2	7.77	0.87	0.40	0.36	0.34	**0.33**
	3	9.63	1.30	0.63	0.44	0.40	**0.38**
	4	10.19	1.54	0.77	0.43	0.39	**0.38**
	5	10.53	1.76	0.94	0.44	0.40	**0.39**
	6	10.38	1.85	0.87	0.44	0.39	**0.37**
	7	10.19	1.96	0.98	0.45	0.39	**0.38**
Average		9.78	1.55	0.77	0.43	0.39	**0.37**
80	2	7.59	0.81	0.36	0.32	0.30	**0.28**
	3	9.45	1.24	0.58	0.38	0.34	**0.33**
	4	9.98	1.48	0.74	0.38	0.34	**0.32**
	5	10.35	1.68	0.90	0.39	0.35	**0.33**
	6	10.17	1.81	0.83	0.38	0.33	**0.32**
	7	9.95	1.92	0.93	0.39	0.33	**0.32**
Average		9.58	1.49	0.72	0.37	0.33	**0.32**
100	2	7.35	0.77	0.34	0.29	0.26	**0.25**
	3	9.21	1.20	0.53	0.34	0.31	**0.29**
	4	9.78	1.45	0.71	0.34	0.29	**0.28**
	5	10.12	1.64	0.88	0.35	0.30	**0.29**
	6	10.04	1.77	0.80	0.33	0.29	**0.27**
	7	9.74	1.87	0.89	0.35	0.29	**0.28**
Average		9.37	1.45	0.69	0.33	0.29	**0.28**
Tot. average		9.87	1.59	0.80	0.46	0.42	**0.40**

Table 2: Average Relative Percentage Deviation ($AVRPD$) for the tested methods grouped by CPU time limit C and number of factories F (best values in bold).

1.45% for $C = 100$, i.e., significantly worse results even when CPU times are quintupled. Therefore, while the accelerations might reduce the differences, they are surely not enough to bridge such a large performance gap.

When looking at the results of the two proposed methods, the single stage IG1S and the two stage IG2S, we observe that they obtain the best results. From the results of Table 2, IG1S has, for all combinations of C and F, a lower $AVRPD$ than BSIG. More specifically, for the shortest CPU times of $C = 20$, the $AVRPD$ of BSIG is 0.97% while that of IG1S is 0.62%. While it might seem that the difference between 0.97% and 0.62% is just 0.35%, in relative terms, the $AVRPD$ of IG1S when $C = 20$ is 36.71% lower than that of BSIG. Furthermore, the differences increase with the value of C. For example, for $C = 100$, the $AVRPD$ of BSIG is 0.69% and IG1S gives 0.29%, i.e., a reduction of a 57.98%. When comparing the best results obtained for the 10 replicates for the 720 instances of BSIG and IG1S, we observe that IG1S yields better $C_{\max}$ values in 531 cases, both methods tie in 182 cases and only in 7 instances out of the 720 does BSIG yield better results.

Another way of looking at the superiority of IG1S is that IG1S is able to obtain a slightly better $AVRPD$ for $C = 20$ compared to BSIG for $C = 100$. In other words, IG1S is able to produce results that are more than 50% better than BSIG, halving the $AVRPD$ values or to obtain comparable results in one fifth of the CPU time.

An interesting comparison is the version without the LS3 local search and the NEH2_en initialization of IG1S$^-$. Recall that the only difference between IG1S and IG1S$^-$ is that the later uses VND(a)$_R$ local search and NEH2 initialization. These two small changes yield tangible benefits as IG1S has $AVRPD$ values that are between 7% and 12% lower than IG1S$^-$, depending on the value of C. The two stage IG2S also improves results. Just dedicating 5% of the available CPU time for the second stage ($\rho = 0.95$ as per the calibration of IG2S) generates reductions in the $AVRPD$ when compared to IG1S of between 3% and almost 5%, and between 39% and 60% when compared to BSIG, depending on the C value.

While the differences are large enough to be statistically significant, it is still advisable to do the test. We carry out a multifactor ANOVA with RPD as the response variable, controlling the factors Algorithm (main factor to study), $n \times m$, F and C. HIA is removed from the experiment as its large RPD values were creating normality problems in the ANOVA. SS is also removed as it is clearly worse than the other methods. The means plot of

the interaction between the algorithm and C factor is shown in Figure 6. The statistical analysis confirms that intervals do not overlap and that all

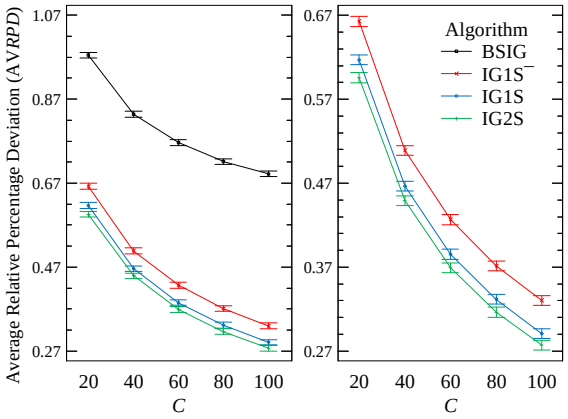

Figure 6: Means plots for the interaction between algorithms BSIG, IG1S$^-$, IG1S, IG2S and CPU time C. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals. On the right plot BSIG is removed.

observed differences in the $AVRPD$ values of Table 2 are indeed statistically significant. It has to be noted that even though the differences in $AVRPD$ values between IG1S and IG2S are small (between 3% and almost 5%), they are large enough to be considered statistically significant, as can be seen from the right plot of Figure 6. Similarly, the differences between IG1S$^-$ and IG1S are clearly significant.

5. Conclusions and future research

In this paper we have studied a generalization of the permutation flowshop scheduling problem in which a manufacturing company operates several identical factories and the additional decision of where to produce each job arises. The distributed permutation flowshop scheduling problem was proposed initially by Naderi and Ruiz (2010) and many authors have presented new methods over the course of the last 7 years. Of particular interest are the simple and effective Iterated Greedy methods, which require little instantiation and yet produce excellent results. We have proposed two IG algorithms that incorporate an enhanced initialization, a biased destruction operator that simply extracts 50% of the jobs to be removed from the factory generating the $C_{\max}$, an improved reconstruction and a local search with a higher degree of intensification. Also, we have explored the concept of a second stage IG

where, once the first stage is finished and for a limited amount of time, a second stage IG focusing on the factory generating the $C_{\max}$ is able to squeeze out additional performance.

We have observed how since the initial work of Naderi and Ruiz (2010), the state-of-the-art has improved considerably. Naderi and Ruiz (2014) managed to improve all 720 best upper bounds of the original results of Naderi and Ruiz (2010) and now with the results of the proposed IG2S, 497 new upper bounds have been found. Average Relative Percentage Deviations have been reduced by 60% when compared to the best competitor, the BSIG of Fernandez-Viagas and Framinan (2015) and by 81% when compared to the SS of Naderi and Ruiz (2014). These improvements have not required complex algorithms or deep problem specific constructions but rather an increased diversification and intensification in the main IG operators.

Additional research lines open from the consideration of other optimization objectives, for which very little work exists for the DPFSP. Also, the joint consideration of sequence dependent setup times and non-identical factories in this problem is of interest as it would bring the problem closer to real-life settings.

Acknowledgments

Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds. Quan-Ke Pan is supported by the National Natural Science Foundation of China (Grant No. 51575212).

References

Bargaoui, H., Belkahla Driss, O., and Ghedira, K. (2016). Minimizing makespan in multi-factory flow shop problem using a chemical reaction metaheuristic. In *2016 IEEE Congress on Evolutionary Computation, CEC 2016; Vancouver; Canada*, pages 2919–2926. Institute of Electrical and Electronics Engineers.

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, New York.

Behnamian, J. and Fatemi Ghomi, S. (2016). A survey of multi-factory scheduling. *Journal of Intelligent Manufacturing*, 27(1):231–249.

Chan, H. and Chung, S. (2013). Optimisation approaches for distributed scheduling problems. *International Journal of Production Research*, 51(9):2571–2577.

Deng, J. and Wang, L. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and Evolutionary Computation*, 32:121–131.

Fernandez-Viagas, V. and Framinan, J. M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 53(4):1111–1123.

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

Framinan, J. M., Gupta, J. N. D., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(1):1243–1255.

Framinan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*. Springer, New York.

Gao, J. and Chen, R. (2011a). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, 4(4):497–508.

Gao, J. and Chen, R. (2011b). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. *Scientific Research and Essays*, 6(14):3094–3100.

Gao, J., Chen, R., and Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651.

Gao, J., Chen, R., Deng, W., and Liu, Y. (2012a). Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm. *Journal of Computational Information Systems*, 8(5):2025–2032.

Gao, J., Chen, R., and Liu, Y. (2012b). A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple factories. *International Journal of Advancements in Computing Technology*, 4(7):121–129.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326.

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

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, 51(17):5292–5308.

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 169:76–88.

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

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

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

Lin, S.-W., Ying, K.-C., and Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, 51(16):5029–5038.

Liu, H. and Gao, L. (2010). A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In *Proceedings - 6th International Conference on Manufacturing Automation, ICMA 2010; Hong Kong; China*, pages 156–163. IEEE Computer Society.

MacCarthy, B. L. and Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1):59–79.

McKay, K. N., Pinedo, M., and Webster, S. (2002). Practice-focused research issues for scheduling systems. *Production and Operations Management*, 11(2):249–258.

McKay, K. N., Safayeni, F. R., and Buzacott, J. A. (1988). Job-shop scheduling theory: What is relevant? *Interfaces*, 18(4):84–90.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Montgomery, D. C. (2012). *Design and Analysis of Experiments*. Wiley, eight edition.

Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768.

Naderi, B. and Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, 239(2):323–334.

Nawaz, M., Enscore, Jr, E. E., and 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.

Pan, Q.-K. and Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 44(1):41–50.

Pan, Q.-K., Ruiz, R., and Alfaro-Fernández, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80:50–60.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms and Systems*. Springer, New York, fifth edition.

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

Reisman, A., Kumar, A., and Motwani, J. (1997). Flowshop scheduling/sequencing research: A statistical review of the literature, 1952-1994. *IEEE Transactions on Engineering Management*, 44(3):316–329.

Ribas, I., Companys, R., and Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, 74:41–54.

Rifai, A., Nguyen, H.-T., and Dawal, S. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing Journal*, 40:42–57.

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

Ruiz, R. and 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.

Sörensen, K. (2015). Metaheuristics-the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):67–74.

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

Urlings, T., Ruiz, R., and Stützle, T. (2010). Shifting representation search for hybrid flexible flowline problems. *European Journal of Operational Research*, 207(2):1086–1095.

Wang, J., Wang, L., and Shen, J. (2016a). A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem. In *2016 IEEE Congress on Evolutionary Computation, CEC 2016; Vancouver; Canada*, pages 2240–2246. Institute of Electrical and Electronics Engineers.

Wang, K., Huang, Y., and Qin, H. (2016b). A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. *Journal of the Operational Research Society*, 67(1):62–82.

Wang, L., Deng, J., and Wang, S.-y. (2016c). Survey on optimization algorithms for distributed

shop scheduling. *Control and Decision*, 31(1):1–11.

Wang, S.-Y., Wang, L., Liu, M., and Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, 145(1):387–396.

Xu, Y., Wang, L., Wang, S.-y., and Liu, M. (2014). An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Engineering Optimization*, 46(9):1269–1283.