Hatami, S.; Ruiz García, R.; 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. doi:10.1016/j.ijpe.2015.07.027.

Heuristics and Metaheuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times

Sara Hatami[a], Rubén Ruiz[b,*], Carlos Andrés-Romano[a]

[a] *Departamento de Organización de Empresas, Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain*
[b] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*

Abstract

We consider a Distributed Assembly Permutation Flowshop Scheduling Problem with sequence dependent setup times and the objective of makespan minimization. The problem consists of two stages, production and assembly. The first stage comprises f identical factories, where each factory is a flowshop that produces jobs which are later assembled into final products through an identical assembly program in a second assembly stage made by a single machine. Both stages have sequence dependent setup times. This is a realistic and complex problem and therefore, we propose two simple heuristics and two metaheuristics to solve it. A complete calibration and analysis through a Design Of Experiments (DOE) approach is carried out. In the process, important knowledge of the studied problem is obtained as well as some simplifications for the powerful Iterated Greedy methodology which results in a simpler approach with less parameters. Finally, the performance of the proposed methods is compared through extensive computational and statistical experiments.

Keywords: Distributed assembly permutation flowshop, Variable neighborhood descent, Iterated greedy, Sequence dependent setup times

*Corresponding author
Email addresses: saha4@upvnet.upv.es (Sara Hatami), rruiz@eio.upv.es (Rubén Ruiz), candres@omp.upv.es (Carlos Andrés-Romano)

1. Introduction

An assembly production floor typically contains two differentiated stages; a production and an assembly section. In this paper we study a distributed assembly flowshop with many potential applications. Assembly flowshops have been widely studied recently and constitute a hot topic for research. The scheduling setting considered in this paper is composed of a production section that is a distributed flowshop problem in itself where jobs are manufactured in a set of machines that are disposed in series. After individual jobs are produced, they are assembled in a single assembly machine to form final products. These production systems are referred to as Assembly Flowshop Scheduling Problems (AFSP) according to [1]. The AFSP applications range from fire engine assembly (2) to personal computer manufacturing (3). As pointed out in [1], AFSP settings are capable of producing large product varieties by using modular structures at a controlled cost.

We also consider several extensions to the studied problem so as to bring it as close as possible to the reality of production shops. For example, single factories are not common in practice and many companies operate several factories working as distributed production environments (4). Distributed production is key in modern manufacturing (5). Additionally, distributed manufacturing leads to high quality production and other benefits such as reduced production costs, decreased management risks and more (4, 6, 7, 8, among others). As a first extension we consider several distributed assembly flowshops to reap these benefits.

The second extension considered is the addition of setup times. Unlike processing times, setups are non-productive periods of time in between the production of successive jobs in machines where cleaning, configurations, adjustments and other procedures are carried out. Setups are broadly classified into Sequence Independent Setup Times (SIST) and Sequence Dependent Setup Times (SDST). This last category is more realistic and general and appears when the amount of setup time depends on the job that has been finished by the machine and the job that is to be produced next. Scheduling with setup times is a very important area of research and a large number of review papers have been published, such as [9], [10, 11] or [12].

More precisely, the flowshop problem consists of scheduling a set N of n jobs in a set M of m machines. Jobs have to visit a predetermined machine sequence which is, without loss of generality, $\{1, 2, \ldots, m\}$. The machines are disposed in series and a job is broken down into m tasks, one per machine.

The processing time of a given job at a machine is a known, deterministic and non-negative quantity referred to as p_{ij}, $i \in M, j \in N$, which is furthermore usually an integer. The objective is to obtain a sequence of the products in the machines so that a criterion is optimized. There are n tasks per machine and any ordering is possible. Therefore, there are $(n!)^m$ possible solutions in this problem. In order to reduce the search space, the most studied variant of this problem is the so called Permutation Flowshop Scheduling Problem or PFSP. In this case, job passing is not allowed and once a production sequence of the jobs is determined for the first machine, it is maintained for all other machines, reducing the search space to $n!$ solutions or sequences. The PFSP comes with some assumptions: A task from a given job can only start at a machine i when the processing of the task of the same job at the previous machine $i-1$ has finished and also only when machine i is free after processing the previous task in the sequence. No breakdowns are experienced by the machines and they are always available. Each machine can only process one job at a time and each job can only be processed by one machine at the same time. The first task of each job on machine 1 is ready for processing at time 0. There is no preemption, i.e., once a task begins processing in a machine it cannot be stopped until completion. Finally, jobs can wait indefinitely in between machines and an infinite storage of in-process products exists (13). If we define by C_j the time at which job $j \in N$ is completed at the last machine m, the most commonly studied criterion is the minimization of the maximum completion time, commonly referred to as makespan or $C_{\max}$. The PFSP with this criterion has been studied extensively in the scheduling literature. Some reviews are [14], [15], [16] and [17].

The extension of the PFSP to distributed manufacturing, referred to as the Distributed Permutation Flowshop Problem (DPFSP) was studied for the first time in [18]. In this extension, we have a set F of f identical factories. Each factory is a PFSP. Each job has to be first assigned to one of the factories and the problem then consists of solving f PFSPs while minimizing the maximal $C_{\max}$ among the f factories. It is assumed that once a job $j \in N$ is assigned to a factory $f \in F$, it is completed there and no reassignments are possible. The authors of [19] recently studied the Distributed Assembly Permutation Flowshop Scheduling Problem or DAPFSP for the first time. In this problem, the first stage is a distributed flowshop and the second stage is a single assembly machine. The authors presented a Mixed Integer Linear Programming Model (MILP), several constructive heuristics and simple local search based Variable Neigborhood Descent (VND) methods. In this paper

we further generalize the DAPFSP with the addition of sequence dependent setup times both in the distributed flowshop production stage as well as in the single machine assembly stage. We improve on the previous VND and also present an effective Iterated Greedy (IG) method. IG has shown excellent performance in the regular PFSP (20) and also in the PFSP with SDST (21) and hence is chosen as a promising approach.

This DAPFSP with sequence dependent setup times (DAPFSP-SDST) is now explained in detail. There is a set T of t (unrelated) products that are manufactured through an assembly of n jobs, each fabricated in the PFSP factories of the production stage. There is a defined assembly program for each product $h \in T$ carried out on a single assembly machine, referred to as M_A. Each product $h \in T$ is assembled from a subset $N_h, N_h \subseteq N$ of jobs that need to be assembled into product h. Therefore, product h consists of $|N_h|$ jobs. Each job belongs to a single assembly program of a given product and therefore we have $\sum_{h=1}^{t} |N_h| = n$. A product h can be assembled at the single machine assembly stage only after all jobs in N_h have been completed in the f distributed factories. The assembly processing time in the single machine assembly stage is referred to as p_h. Furthermore, S_{ijk} denotes the sequence dependent setup time that is needed at machine i of any of the f factories after having processed job j and before processing job k. This setup time is separable from the processing time. There is also an initial setup time. As a result, a $(n + 1 \times n)$ setup time matrix is considered for each production machine. Setup time matrices do not change from factory to factory as factories are assumed to be identical. We also consider sequence dependent setup times in the single machine assembly stage. We denote by SA_{ls} the setup between the assembly of products l and s, $l \neq s, l, s \in T$. Note that an initial setup is also needed to prepare the assembly machine for the assembly of the first product $h \in T$, referred to as SA_{0h}. Again, a $(t + 1 \times t)$ assembly setup time matrix is required. All setups are non-negative integers that are known in advance and deterministic.

The paper is arranged as follows: Section 2 presents a brief literature review on previous and related research. Section 3 introduces two simple constructive heuristic methods for the considered problem. Sections 4 and 5 describe the proposed VND and IG methods, respectively. In section 6, the proposed methods are calibrated. Section 7 presents a complete computational evaluation of the proposed algorithms. Finally, Section 8 concludes the paper and presents some future research questions.

2. Literature review

The DAPFSP is a combination of the assembly (AFSP) and distributed (DPFSP) permutation flowshop problems. Together with the regular flowshop, the literature is extensive. The reader is again referred to the many existing reviews (14, 15, 16, 17).

As regards the AFSP, there is also a significant amount of existing results. In [2] a three-machine assembly-type flowshop scheduling problem with makespan minimization is presented. Each product consists of two jobs, each to be produced in the first and second machine respectively, where the third machine assembles the two jobs into a product. The authors present a branch-and-bound exact method and an approximate solution procedure. In [3] m parallel production machines in the first stage are considered. A compact vector summation technique to find approximated solutions with worse-case absolute performance guarantees is applied. In [22] a branch-and-bound algorithm for the same model is developed. A two-stage assembly scheduling problem is considered in [23]. A lower bound and a dominance criterion are developed and incorporated into a branch-and-bound procedure, this time with total weighted flow time minimization as an objective. A heuristic procedure to find an initial upper bound is also proposed. In [24] the same model is studied and metaheuristics such as simulated annealing (SA), tabu search (TS), and hybrid tabu search heuristics to solve the problem are proposed. In [25] a two-stage AFSP is considered and TS, particle swarm optimization (PSO), and self-adaptive differential evolution (SDE) are applied to minimize the weighted sum of makespan and maximum lateness. In [26] powerful heuristics for minimizing the makespan in a fixed three machine assembly-type flowshop problem are presented.

The literature about the distributed permutation flowshop problem is comparatively small, especially when compared with that of the AFSP and PFSP. The DPFSP is introduced in [18] for the first time. They developed six different Mixed Integer Linear Programming (MILP) models and proposed two simple factory assignment rules and 14 heuristics based on dispatching rules, effective constructive heuristics and VND methods. More recently, in [27] a TS algorithm with a better performance when compared to previous algorithms presented by the same authors is presented. The authors of [28] have proposed an effective Iterated Greedy method and in [29] an Estimation of Distribution algorithm is proposed. The authors of [19] introduced for the first time the DAPFSP and proposed a MILP, three constructive algorithms

and a VND. To the best of our knowledge, the DAPFSP with a single assembly machine has not been studied by any other authors in the literature.

Setup times are also considered in the non-distributed assembly flowshop literature (and much more in the regular flowshop). The authors of [30] presented a two-stage production system, where there is a single production machine with setup times that produces parts and a single assembly machine. A near-optimal schedule is obtained by using a pseudo-dynamic programming method and a tight lower bound is proposed to evaluate its accuracy. The objective function considered is the minimization of the mean completion time. The same author built upon the previous model in [31] by extending the single machine manufacturing stage to a flowshop with setup times. A pseudo-dynamic programming method and a branch-and-bound procedure are presented. The authors of [32] addressed the two-stage AFSP with sequence independent setup times. They derived a dominance relation and applied SDE, PSO, TS and Earliest Due Date heuristics to minimize the maximum lateness. The same model is considered in [33], where the authors presented a dominance relation and proposed three heuristics to minimize the makespan. The authors of [34] presented a three stage AFSP by considering a transfer stage as a middle stage and SDST in the first stage. They presented a mathematical model, a lower bound and two heuristics (TS and SA) to solve the problem. In [35] the authors also addressed the two-stage AFSP by considering multiple non-identical assembly machines and SDST in the first production stage. They developed a MILP and a hybrid VNS heuristic to minimize the weighted sum of makespan and mean completion time. Comprehensive reviews of the state-of the art of scheduling with setup times are carried out in [9], [12], [15] and [10, 11]. As can be seen, the DAPFSP with SDST considered in this paper has not been, to the best of our knowledge, studied before in the scheduling literature.

3. Simple constructive heuristic methods

The DPFSP is an $\mathcal{NP}$-Hard problem if $(n > f)$ (18). Therefore, the DAPFSP with sequence dependent setups is also $\mathcal{NP}$-Hard as the DPFSP is a special case. As a result, the design of heuristic methods for obtaining good solutions in reasonable CPU times is necessary. In the following we present two simple constructive heuristics.

We first present a simple example problem that will be used to illustrate the proposed heuristics. The example consists of eight jobs ($n = 8$), three

	Job							
Machine	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	46	48	94	2	4	47	50	33
M_2	47	2	83	13	69	42	26	95
	Product 1		Product 2			Product 3		
M_A	30		60			89		

Table 1: Job and product assembly times for the example.

Product	Product		
	T_1	T_2	T_3
T_0	6	3	1
T_1	0	4	5
T_2	3	0	6
T_3	7	2	0

Table 2: Assembly stage setup time matrix for the example.

products ($t = 3$), two factories ($f = 2$) with a two machine flowshop each ($m = 2$). The assembly programs of the three products are: $N_1 = \{1, 6, 7\}$, $N_2 = \{2, 5\}$ and $N_3 = \{3, 4, 8\}$. Tables 1 to 3 present job processing times at factories, product assembly times at the single machine assembly stage and production and assembly machine setup matrices, respectively.

We introduce some necessary notation. π represents a *product sequence*, that is, a possible sequence for the assembly of the products, e.g., $\pi = \{1, 3, 2\}$. Each product h is composed of a number of jobs and a possible sequence for these jobs is referred to as π_h, denoting the *job sequence for product* h, e.g., $\pi_1 = \{7, 6, 1\}$, $\pi_2 = \{2, 5\}$ and $\pi_3 = \{8, 3, 4\}$ are possible job sequences for the three products in the example. A *Complete job sequence*, π_T, represents a possible sequence of the all jobs, and is the result of concatenating all job sequences for the products after the master product sequence π, e.g., $\pi_T = \{7, 6, 1, 8, 3, 4, 2, 5\}$ following the example. To start processing the first job at each factory and for assembling the first product at the assembly stage, an initial setup is necessary. We use J_0 and T_0 to represent the first dummy job and product, respectively.

Job	Machine 1								Machine 2							
	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
J_0	2	7	3	8	4	1	9	3	8	9	7	1	9	8	7	4
J_1	0	5	9	4	9	1	3	2	0	1	8	8	9	2	6	4
J_2	3	0	7	3	2	1	6	7	7	0	2	5	2	1	3	6
J_3	4	4	0	5	5	8	3	4	5	9	0	4	9	1	6	9
J_4	8	2	4	0	2	1	3	3	4	2	5	0	8	8	1	1
J_5	5	3	3	4	0	3	7	5	1	4	3	2	0	6	1	5
J_6	8	1	8	4	3	0	1	3	1	4	8	2	7	0	6	6
J_7	5	5	8	3	7	4	0	3	7	7	5	7	4	2	0	5
J_8	9	3	8	2	7	8	7	0	7	7	1	8	1	5	6	0

Table 3: Production stage setup time matrix for the example.

To assign jobs to factories, the two job to factory assignment rules presented in [18] are considered in this paper. The first one, referred to as (NR_1), assigns

job j to the factory with the lowest current $C_{\max}$, not considering job j. The second rule (NR_2) assigns job j to the factory with the lowest $C_{\max}$ after scheduling job j.

3.1. Heuristic 1

The first heuristic obtains a complete job sequence π_T and consists of three simple steps. The first obtains a product sequence (π) on the single assembly machine. The product with the minimum sum of initial setup and assembly time is scheduled first in π. The remaining $h-1$ products are scheduled one by one, each time selecting the product with the smallest completion time after being scheduled, considering the sequence dependent setup time. Once all products are scheduled the second step in the heuristic determines the job sequence (π_h) of each individual product h. The jobs of each product h are considered one by one. Initially all factories are empty. Therefore, the first f jobs with the minimum completion times (initial setup plus processing time) are the first f jobs on π_h and occupy the first positions in the f factories. Of course, if $|N_h| \leq f$, π_h is equal to the the assembly program of product h, N_h. Otherwise, after f initial jobs are scheduled, the other $|N_h| - f$ jobs from product h are considered. Among the remaining jobs of product h, the job that is scheduled next is the one resulting in the smallest completion time after applying either the NR_1 or NR_2 job to factory assignment rules. The process continues until all jobs of product h have been considered. This second step is applied to each product separately to determine the job sequence for each individual product. After all products have been considered, the third step constructs the complete job sequence π_T by putting together all obtained π_h, following the product order established in π. At this point there are two possibilities: to assign all jobs in π_T to factories using the NR_1 or NR_2 rules. Depending on the case we have the proposed heuristic CH_{11} or CH_{12}, respectively. The sequence of products in the assembly stage is simply determined by ordering the products by increasing completion time of all the jobs in the production stage. To better illustrate the heuristic, all steps are explained through the previous example.

Product 1 is considered as the first product to be included into π. It is scheduled first in the single assembly machine which results in a completion time of 6+30=36 (considering the initial setup and the assembly times). The same procedure is carried out for the remaining products 2 and 3 which result in completion times of 3+60=63 and 1+89=90, respectively. Since product 1 results in the shortest completion time, it is scheduled first in π. Now we

have to reconsider products 2 and 3 in the single assembly machine. They are scheduled now after product 1 which has been already scheduled. The completion times are 36+4+60=100 (completion time of product 1 plus the setup time in the assembly stage between products 1 and 2 and processing time of product 2) for product 2 and 36+5+89=130 for product 3. Therefore, product 2 is scheduled after product 1. Finally, no additional calculations are needed for scheduling the last product 3 in the third position. As a result, the product sequence π is $\{1, 2, 3\}$. Note that this first step of the heuristic is carried out $\frac{t(t+1)}{2} - 1$ times and therefore has a computational complexity of $\mathcal{O}(t^2)$. The next step is to find a good job sequence for each product h, π_h. Recall that there are $|N_h|$ jobs that belong to product h. We consider product 1 as an example that consists of jobs $\{1, 6, 7\}$ in the example.

We begin by calculating the completion times of jobs 1, 6 and 7, separately. Since the two available factories are empty, we consider the initial setups and the completion times on the two machines at the flowshop of each factory. For example, the completion time of job 1 is 2 (initial setup at machine 1)+46 (processing time at machine 1)+47 (processing time at machine 2)=95. Note that the initial setup of 8 units in machine 2 can be performed before job 1 arrives to that machine. Applying the same procedure we calculate the completion times for jobs 6 and 7 which are 90 and 85, respectively. Since we have $f = 2$ factories, we select the f jobs with smallest completion times and schedule them. In this case jobs 7 and 6 are scheduled in factories 1 and 2, respectively and occupy the first two positions of the product sequence for product 1 (π_1). To schedule the remaining jobs in π_h, each one should be tested at each factory using either the NR_1 rule for CH_{11} or NR_2 for CH_{12}. The job resulting in the minimum completion time is scheduled next in π_h. This process continues until all jobs in N_h have been scheduled and is repeated for all the product sequences. In this example only job 1 remains and therefore occupies the last position in π_1. Therefore π_1 is $\{7, 6, 1\}$. Applying the same procedure results in the job sequences for products 2 and 3 to be $\pi_2 = \{2, 5\}$ and $\pi_3 = \{4, 8, 3\}$, respectively. Note that for each product h this second step requires $\frac{|N_h|(|N_h|+1)}{2} - f - 1$ steps. It is difficult to calculate the computational complexity for this step as usually $|N_h|$ is not expected to be orders of magnitude larger than f and therefore the term $-f$ in the previous expression is important. However, if we assume that $|N_h| \gg f$ and that there is a single product where $|N_h| = n$ then the computational complexity of this second step is $\mathcal{O}(n^2)$. Note however that this is a pathological worst case

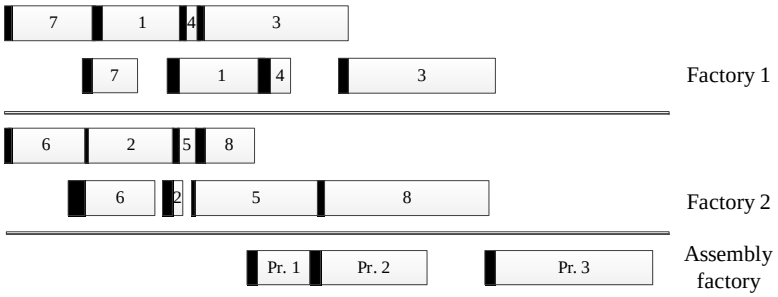

Figure 1: Gantt chart with the result of CH_{11} for the example problem.

and the empirical complexity is expected to be much lower. Finally, in the third step the complete job sequence π_T is completed by concatenating all job sequences following the product sequence π. This sequence is therefore $\pi_T = \{7, 6, 1, 2, 5, 4, 8, 3\}$. In order to calculate the maximal makespan among the factories, the individual jobs in π_T are assigned to factories in the order they appear in π_T, using the rules NR_1 or NR_2 for heuristics CH_{11} and CH_{12}, respectively. This last step has a computational complexity of $\mathcal{O}(nf)$. Therefore, considering that $n \gg f$, the overall worst case computational complexity of this first heuristic is $\mathcal{O}(n^2)$. In the considered example, we obtain the makespan value of 386 for CH_{11} and 387 for CH_{12}. The solution given by CH_{11} is represented as a Gantt chart in Figure 1.

3.2. Heuristic 2

This heuristic is based on the second constructive method presented in [19]. The idea is to consider the production stage and to sequence all jobs of each product and construct the different π_h sequences so that priority is given to products whose jobs have small completion times. In this way, the single assembly machine is occupied as soon as possible. In order to obtain good job sequences π_h for all products, the second step of the previous heuristic 1 is used. After all jobs for a given product h are scheduled, we calculate the earliest assembly start time for product h, denoted by E_h which is equal to $\max_{j=1}^{|N_h|}\{C_j\}$. After all individual product job sequences are determined, the product sequence π, is formed by sorting all t products according to ascending values of E_h. Finally, the complete sequence π_T is obtained after concatenating all job sequences π_h following the product sequence established in π. Similarly to heuristic 1, jobs in π_T are assigned to factories using NR_1 or NR_2 which results in heuristics CH_{21} and CH_{22}, respectively. The sequence of products for the assembly stage is obtained as in heuristic 1.

The computational complexity of this second heuristic is dominated by the second step, which corresponds to the second step of the previous heuristic 1. Therefore, the computational complexity is the same in the worst case: $\mathcal{O}(n^2)$.

Following the job sequences obtained for the three products in the example of the previous heuristic, the earliest assembly start times of the products are $E_1 = 157$, $E_2 = 78$ and $E_3 = 191$. Therefore the product sequence π is $\{2, 1, 3\}$ by sorting all E_h in ascending order. The complete sequence π_T is therefore $\{2, 5, 7, 6, 1, 4, 8, 3\}$. After assigning each job to factories we obtain makespan values of 387 for heuristic CH_{21} (NR_1 rule) and 391 for heuristic CH_{22} (NR_2 rule).

The four proposed heuristics will be tested later on as seed solutions of the other proposed approaches for solving the DAPFSP-SDST problem.

4. A simple Variable Neighborhood Search

Variable Neighborhood Descent (VND) is the simplest variant of the more general Variable Neighborhood Search (VNS) of [36]. Starting from an initial solution, VND explores different neighborhood structures, $N_1, \ldots, N_q$. These are usually explored in increasing cardinality starting with the smallest neighborhood N_1. The search continues with N_2 only after a local optimum has been obtained in N_1. If the local optimum obtained after exploring N_2 is different from the one obtained after analyzing N_1, the search goes back to exploring N_1. The process ends when all neighborhoods, including N_q, have been searched and the final solution is a local optimum with respect to all neighborhood structures. VND is very simple yet it performs well for the distributed flowshop and DAPFSP problems as shown in [18] and in [19]. In the following we summarize the proposed VND which employs two different solution representations and two neighborhood structures.

4.1. Solution representation

In this work, and differently from [19], we consider two different solution representations. The base encoding is a permutation of all jobs, i.e., we work with the complete job sequence π_T. Using this encoding we define the full permutation solution representation or (Pr1) as the ordering of the n jobs regardless of the products to which they belong. Hence, $n!$ different job permutations are possible with this representation.

Pr1 is a relaxation of the more restricted representation given in [19]. This

second representation, referred to as multi-permutation or Pr2 is also a complete job sequence but the jobs belonging to the same product are never separated and intermingled with jobs belonging to other products. Following the previous example, if we have a product sequence $\pi = \{2, 3, 1\}$, two possible representations could be $\{2, 5, 8, 3, 4, 7, 6, 1\}$ or $\{5, 2, 4, 8, 3, 7, 1, 6\}$. However, $\{2, 8, 5, 3, 4, 7, 6, 1\}$ is not valid as job 8, which belongs to product 3 is scheduled before job 5 which belongs to product 2 and the product sequence π forces all jobs of product 2 to be scheduled before all jobs of product 3. Note that Pr2 is smaller than Pr1 as in total Pr2 contains $t! \times \prod_{h=1}^{t} |N_h|!$ possible solutions.

4.2. Pr1 neighborhoods

Two neighborhoods are considered after the work of [18], the first one, referred to as LS_1, works at each factory by extracting each job and reinserting it in all possible positions of the PFSP at that factory. The process continues until all jobs have been examined with no improvements in the $C_{\max}$ for all factories. The second neighborhood, LS_2, takes all jobs assigned to each factory and inserts them at all possible positions in all other factories looking for a makespan improvement at the involved factories. For more details, the reader is referred to [18].

4.3. Pr2 neighborhoods

Again two neighborhoods are employed. These are based on the VND proposed in [19]. The first neighborhood is referred to as LS_P and works over the product sequence π. It extracts and reinserts each product into all possible $t - 1$ positions of π. Note that this is equivalent to extracting and inserting the block of consecutive jobs that correspond to each product h in π_T. The second neighborhood is referred to as LS_J. It is also an insertion neighborhood but in this case all jobs that make a product are extracted and inserted into all possible positions of the job sequence for product h, i.e., all t products are considered and all of their $|N_h|$ jobs are extracted and inserted into all job sequences. After each insertion and in both neighborhoods we obtain a complete job sequence π_T, therefore, all jobs need to be assigned to factories using the NR_1 or NR_2 assignment rules. More details are given in [19].

5. Iterated Greedy algorithm

Iterated Greedy (IG) was first applied to the regular permutation flowshop problem by [20] with the objective of minimizing makespan. The good results obtained have encouraged the application of the IG methodology to other scheduling problems. Regular flowshops with blocking constraints were approached by [37]. No-wait flowshop was successfully solved with IG algorithms by [38]. IG showed excellent performance in no idle and mixed no-idle flowshops recently in [39]. The SDST PFSP was tackled with IG methods in [21]. Also, other objectives apart from makespan have been considered, like tardiness (40) and total flowtime (41). Multiobjective flowshops have also been adequately solved with IG techniques in [42] or even with the addition of setup times in [43]. Finally, and as commented in Section 2, the DPFSP has been also solved with IG methods by [28]. Given all these previous successes, applying IG to the DAPFSP-SDST seems promising.

The most relevant characteristic of the IG methodology is its simplicity which does not preclude obtaining competitive results for most tested scheduling settings. IG has very few parameters and does not employ specific problem knowledge. As with most metaheuristics, IG starts from a high-quality initial solution. This starting solution is initially equal to both the incumbent and the best solution. Then, usually four phases are iteratively applied to the incumbent solution until a user set termination criterion is reached. The first phase is a partial destruction of the incumbent solution where some elements of it are (usually randomly) removed. The second phase consists of the reconstruction of the incumbent solution. The removed elements are reinserted in the solution following a greedy heuristic. The result is a new complete solution. The third phase is a local search where the complete solution is improved. The fourth and last operator is the application of an acceptance criterion to decide if the new solution replaces the incumbent one.

In the proposed IG we will test which one of the four proposed heuristics (CH_{11}, CH_{12}, CH_{21} or CH_{22}) will serve as a method to construct the initial solution. In the following sections we explain the four phases of the proposed IG. Note that there are differences depending on the solution representation Pr1 or Pr2.

5.1. Destruction, reconstruction and local search for Pr1

After the initial solution has been obtained we have a starting complete job sequence π_T along with a list of all jobs assigned to each factory. Let

us denote by π_f the sequence of jobs assigned to a factory $f \in F$. In the destruction phase, a percentage of the n jobs ($d\%$) jobs are randomly selected, without repetition, removed from the factories and inserted into a list in the order in which they were selected. Note that according to [18], no factory must be left empty when minimizing makespan. Therefore, a selected job will not be removed from a factory if it is its last job. The destruction procedure, explained in Pseudocode 1, returns the list of removed jobs D and all sequences of jobs assigned to factories, after the removal of the jobs.

Pseudocode 1 Destruction_Pr1(d)

```
i ← 0;
while i < (d · n/100) do
    a ← Job randomly selected among the remaining n − i jobs;
    f ← Factory where job a is assigned;
    if |π_f| > 1 then
        D ← Insert job a;
        π_f ← Remove job a from π_f;
        i ← i + 1;
    end if
end while
return D and all π_f, f ∈ F;
```

In the construction phase, jobs in D are selected, one by one, and reinserted into all possible positions in all factories. Among all positions, the one resulting in the sequence with the smallest $C_{\max}$ is chosen for the job. This process is repeated $d \cdot n/100$ times until D is empty. The local search operator used in the IG is the LS_1 procedure explained in section 4.2. In this local search, for each factory f, jobs are removed from π_f and reinserted into all $|\pi_f| - 1$ possible positions in factory f.

5.2. Destruction, reconstruction and local search for Pr2

The destruction operator is different from Pr1 in a small but important respect. Each one of the $d \cdot n/100$ removed jobs belong to a product h and we do not allow job sequences for any product h (π_h) to be empty, so at least one job must remain in the job sequence of products. In the reconstruction procedure, each job is inserted into all positions of its corresponding job sequence. To decide the best placement for each job in D, all job sequences π_h are coalesced into a complete job sequence π_T and jobs are assigned to factories following the NR_1 or NR_2 job to factory assignment rules. The

process is finished when D is empty and all product job sequences contain all the jobs. For the local search we use a product inter-exchange variant of the aforementioned LS_P local search of Pr2. We denote this local search by LS_{PI} and all $t \times (t-1)$ pairs of products are interchanged in the product sequence π. Duplicate moves are ignored and the inter-exchange resulting in the best improving $C_{\max}$ is carried out. The process is repeated until all movements result in non-improving makespan values.

5.3. Acceptance criteria

Similar to most existing IG literature, including the previously cited papers, once the first three phases (destruction, reconstruction and local search) are carried out over the incumbent solution, we obtain a possibly different schedule and must determine if it replaces the incumbent one. It is known that a simple descent acceptance criterion, i.e., accepting new solutions only if they improve the best found $C_{\max}$ value, results in IG methods that are prone to stagnation and premature convergence. In the initial work of [20] it was proposed that a simple simulated annealing-like type of acceptance criterion with a constant temperature, based on the earlier work of [44] is enough to avoid premature convergence. This acceptance criterion is as follows. Let us denote by π'_T to the incumbent complete solution after the first three phases have been applied and by π_T to the previous solution. Obviously if $C_{\max}(\pi'_T) < C_{\max}(\pi_T)$ then the new solution π'_T is directly accepted. If this is not the case, then solution π'_T is probabilistically accepted following the expression $random \leq e^{-\frac{C_{\max}(\pi'_T)-C_{\max}(\pi_T)}{Temp}}$ where $random$ is a random number uniformly distributed between 0 and 1. Note that $Temp$ is another expression that was proposed originally by [44] as $Temp = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$ where T is a factor that needs to be calibrated. This constant temperature simulated annealing-like type of acceptance criterion has been extensively used in the IG literature. For example [21] used the same acceptance criterion albeit their problem considered sequence dependent setup times. There are at least three potential improvements to this acceptance criterion when applying it to our DAPFSP-SDST problem. First, $Temp$ is not correctly calculated as it does not consider the distributed factories, assembly stage, number of products or setup times. It is not clear how to extend this calculation to obtain a sensible parameter. Second, as shown in [20], [21] and other authors, the T factor inside the calculation of $Temp$ proved not to be statistically significant in a wide range of values in extensive calibration tests. Third, in

the temperature calculation of [44], the final probability of accepting a worse solution basically depends only on the difference $C_{\max}(\pi'_T) - C_{\max}(\pi_T)$. Let us examine this in detail. The expression $Temp = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$ can be reduced to just $Temp = T \cdot 5$, this is because processing times p_{ij}, as we will detail later, are commonly obtained from a uniform distribution in the range 1, 99 in most of the scheduling literature. The average of such a uniform distribution is $(1+99)/2 = 50$, therefore, we have that the numerator of *Temp* approximates to $n \cdot m \cdot 50$. Considering the denominator, $Temp = T \cdot \frac{n \cdot m \cdot 50}{n \cdot m \cdot 10}$ reduces to the stated $T \cdot 5$. There is a potential problem in this approach. The final probability of accepting a final solution depends on the size of the instance and on the magnitude of the $C_{\max}$ value. Take two instances A and B with corresponding $C_{\max}$ values of the incumbent and new solutions as $C_{\max}(\pi_{T_A}) = 100$, $C_{\max}(\pi_{T'_A}) = 110$, $C_{\max}(\pi_{T_B}) = 1000$ and $C_{\max}(\pi_{T'_B}) = 1010$. Both new solutions for A and B are worse than the incumbent by 10 units. However, for instance A these 10 units translate into a 10% solution quality deterioration whereas for instance B, the same 10 units are only a 1% deterioration. The problem with the calculation given in [44] is that both cases have the same probability of acceptance.

To remedy these three potential shortcomings, and as an additional contribution of this paper, we propose two additional acceptance criteria. The first one, and similarly to the one of [44] is very simple. We basically substitute the difference $C_{\max}(\pi'_T) - C_{\max}(\pi_T)$ for the Relative Percentage Difference (RPD) between the makespan value of these two solutions which is calculated as $\text{RPD} = \frac{C_{\max}(\pi'_T) - C_{\max}(\pi_T)}{C_{\max}(\pi_T)} \times 100$. This results in an acceptance criterion calculation as $random \leq e^{-\frac{RPD}{Temp}}$.
The second proposed acceptance criterion, and in order to avoid the statistically insignificant T factor is further simplified as follows: $random \leq e^{-RPD}$.

In total we will test three different acceptance criteria. The original in [44] as described, denoted as AC_1 and the two newly proposed ones, referred to as AC_2 and AC_3, respectively. We will later use sound statistical techniques to test if the two new proposed ones result in better solutions for the DAPFSP-SDST problem.

6. Calibration of the proposed VND and IG methods

We proceed with the calibration of the proposed methods. We are not interested in a high quality and fine tuned process. Instead, we will use some statistical tools to achieve a coarse calibration. The technique of choice is the Design Of Experiments (DOE) approach (45) where we will basically be using screening factorial designs which are sound statistical techniques but still result in an exploratory calibration. The literature on calibration methodologies for metaheuristic methods is slowly gaining traction. Much more advanced methods are given in [46]. We decide to use simpler approaches in order to have a clearer picture of the performance of the proposed methods. Should an advanced tuning methodology be used, it would be difficult to conclude if the proposed methods behave well because they are good for the problem studied or just because a fine tuning calibration has been carried out. The results of the experimental designs are examined by means of the Analysis of Variance technique (ANOVA). ANOVA is a robust parametric tool and at least three main hypotheses must be checked. Some are less important but others are crucial. From more to less important the hypotheses are; independence of the residuals, homoscesdasticity of the factor's levels (homogeneity of variance) and normality in the residuals. All these hypotheses are satisfied in all the following tests but it must be noted in any case that ANOVA has been proven to be extremely robust as stated in [47]. Other authors, like [48] study ANOVA in detail and test it against other non-parametric approaches with data that significantly departs from the three main hypotheses and conclude that ANOVA is preferable to non-parametric approaches most of the time. Furthermore, the most important hypothesis, the independence of the residual, is easy to satisfy in a controlled computational experimentation environment according to [49]. Therefore, the calibration methodology employed should give us a fair, not over-tuned and at the same time sound result.

A set of instances is generated to calibrate the proposed VND and IG. Calibrating methods with the same test instances that will be used in the computational evaluations is ill-advised. When a given method is calibrated with the same test instances later used for comparisons there is a big risk of having a bias in the results (over-fitting). There is no guarantee that with a different benchmark results will hold. Therefore, we calibrate the proposed methods with a different calibration benchmark. 60 instances are

generated randomly with the following combinations of number of jobs (n), machines (m), factories (f), products (t) and distributions for the setup times of production and assembly machines. More specifically, n is tested at two levels (100, 200), m at three (5, 10, 20), f and t are also tested at three levels each, (4, 6, 8) and (30, 40, 50), respectively. Job processing times at the distributed flowshops in the production stage are generated according to a uniform distribution in the range $[1, 99]$ as is common in the scheduling literature. Finally, the product assembly times in the single machine assembly stage depend on the number of jobs assigned to each product h and follow a uniform distribution in the range $[1 \times |N_h|, 99 \times |N_h|]$. Finally, for the setup times we test two uniformly distributed intervals, $[1, 50]$ and $[1, 125]$ for production and assembly setups. All the calibration instances are available at `http://soa.iti.es`.

The response variable studied in the experiments is the Relative Percentage Deviation (RPD), where $RPD = \frac{SOL_{ALG} - BEST_{TOTAL}}{BEST_{TOTAL}} \times 100$. $BEST_{TOTAL}$ is the best known solution obtained over the course of this research for each calibration instance and SOL_{ALG} is the makespan value obtained by any algorithm tested over the same instance. Experimentation is performed in a scientific computation cluster with 30 blades. Each one with 16 GBytes of RAM memory and two Intel XEON E5420 2.5 GHz processors. Each processor has 4 physical computing cores (8 per blade) but no parallel computing is employed in this paper as the 30 servers are only used to split the experimentation work and reduce the total time to obtain results. At each blade we use Windows XP virtual machines with one virtual processor with two cores and 2 GB of RAM memory.

6.1. VND calibration

The proposed VND mainly has three factors or algorithm features that should be tested. The first is the type of solution representation. This factor will be referred to as Pr and is tested at two variants, which correspond to the two different proposed solution representations of Section 4.1 (Pr1 and Pr2). The second factor is the two different job to factory assignment rules (NR) which is tested at two variants NR_1 and NR_2. The third and last factor is the simple constructive heuristic used for initialization (INI), tested at four variants (CH_{11}, CH_{12}, CH_{21} and CH_{22}). The response variable is the RPD and we carry out a multifactor ANOVA to analyze experiments. The number of treatments is the result of all the combinations of all previous factors ($2 \times 2 \times 4 = 16$) and each treatment is tested with all 60 calibration

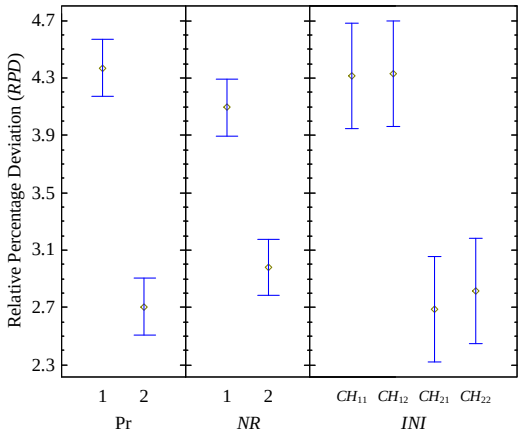

Figure 2: Means plot and 99% confidence level Tukey's HSD intervals for the type of solution presentation Pr, job assignment rules NR, and initial solutions INI for the proposed VND methods.

instances so the total number of experiences is $16 \times 60 = 960$. There is no need for replicates as the proposed VND methods are deterministic.

The analysis and ANOVA table shows that, all studied factors (Pr, NR and INI) are statistically significant. The most significant is the representation (Pr), then job to factory assignment rule (NR) and lastly the initial solution (INI). The means plot and 99% confidence level Tukey's Honest Significance Differences (HSD) intervals for all three factors are given in Figure 2.

The second solution representation, as well as the second job to factory assignment rules result in statistically better performance. As regards the solution representation, the larger cardinality of the solution space in the first representation deteriorates performance, possibly indicating that more neighborhoods or larger neighborhoods are needed. Our experiments confirm that the second job to factory assignment rule works better, which is in line with previous findings (18, 19). However, this assignment rule does not have an effect on the constructive heuristics which only depend on the representation. In the end we select Pr= 2, $NR = 2$ and $INI = CH_{21}$.

6.2. Experimental parameter tuning of the IG

IG has three factors in common with VND to calibrate (Pr, NR and INI). These are tested at the same variants as before. Furthermore, there are three additional factors: percentage of jobs to destruct in the destruction phase (d), type of acceptance criterion (AC) and the value of T used in the calculation of $Temp$ (T). As explained in Section 5.3, we propose three different acceptance

criteria. The first two (AC_1 and AC_2) do depend on the aforementioned parameter T, whereas the third (AC_3) does not have a T factor. As a result, we have to carry out two different experiments. In the first one we test two variants for Pr (Pr1 and Pr2), two variants for NR (NR_1 and NR_2), four variants for initial solution (CH_{11}, CH_{12}, CH_{21} and CH_{22}), three levels for d $(5, 10, 15)\% \times n$, two levels for acceptance criterion (AC_1 and AC_2) and three levels for $T : (0.5, 1, 2.5)$. This results in $2 \times 2 \times 4 \times 3 \times 2 \times 3 = 288$ algorithm configurations. Each one of the 60 calibration instances is run for five different replicates in each configuration resulting in $288 \times 5 = 1,440$ treatments as IG is an stochastic algorithm. Since each treatment is tested with all 60 calibration instances the total number of experiences is $1,440 \times 60 = 86,400$. Additionally, as IG is a metaheuristic with a stopping criterion, we set the elapsed CPU time as a termination criterion, which is fixed at $n \cdot m \cdot f \cdot 45$ milliseconds. This way of setting the termination criterion as a function of the size of the instance helps in decoupling the effect of the instance size in the results. Additionally, all algorithm configurations have the same CPU budget. Not doing so would result in a calibration biased for more time consuming configurations. We employ the same computers for this test as before. With this first experiment, the idea is to set the value of the parameter T for the first two levels of the acceptance criterion only (AC_1 and AC_2). Once T is fixed, we will be able to analyze the three different acceptance criterion together in a second experiment. The results of the first experiment (not shown here due to reasons of space) indicate that the only non-significant factor is T with a p-value very close to 1. However, the interaction between T and the type of acceptance criterion (AC) is significant with a p-value of 0.0004. Both means plots, for the single factor as well as for the interaction are given in Figure 3.

As we can see, the single factor T is not significant as the three levels in the means plot completely overlap. The interaction is significant as the behavior of the T factor greatly depends on the type of acceptance criterion. For AC_1, which recall is the original [20] type of acceptance criterion, increasing the value of T results in better solutions. Originally, [20] tested values of T of 0, 0.1, 0.2, 0.3, 0.4 and 0.5. Here we have tested larger values but the three intervals overlap, meaning that even though solutions improve, the improvement is not consistent enough so as to be statistically significant. The situation is just the opposite for the second acceptance criterion AC_2 as increasing the value of T deteriorates solutions. This together with the fact that overall T is not significant and the previous studies into the IG methodology where T has

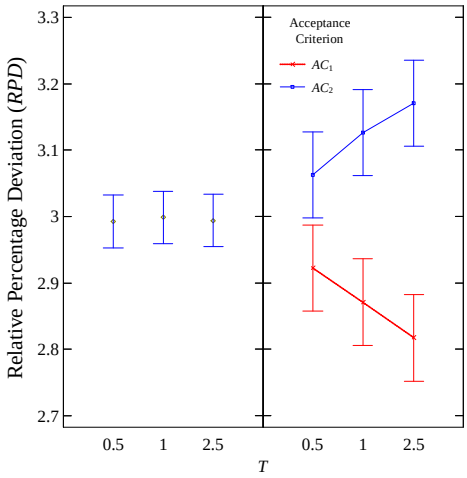

Figure 3: Means plot and 99% confidence level Tukey's HSD intervals for the temperature T parameter and the interaction between the temperature T parameter and acceptance criterion (AC) for the first calibration experiment for the proposed Iterated Greedy methods.

been shown to be statistically insignificant reinforces our idea that T should be removed from the acceptance criterion. For the next experiment we set T at 2.5 for AC_1 and to 0.5 for AC_2.

The second experiment involves all previous factors and all three acceptance criteria but having fixed T as mentioned for the first two acceptance criteria. Therefore, the total number of experiences is now 43,200. The ANOVA results indicate that the interaction between the solution representation (Pr) and the job to factory assignment rule (NR) factors is the most significant effect. This interaction is shown in Figure 4.

Similar to VND, for the proposed Iterated Greedy method the second solution representation and the second job to factory assignment rule result in the best performance by a significant margin. Actually, with the exception of the percentage of jobs to destruct in the destruction phase (d), all other factors are not significant. The initial solution INI is not statistically significant with a p-value close to 0.25. However, this is across all instances. Some statistically significant differences are found in some instance groups when using CH_{21} or CH_{22}. Therefore, and again similar to VND, INI is set to CH_{21}. Of particular interest is the statistical insignificance of the type of acceptance criterion factor (AC) with a very large p-value of more than 0.85. This means that there are very little (if any) differences between the three proposed acceptance criteria. The third proposed criterion does not employ a temperature factor.

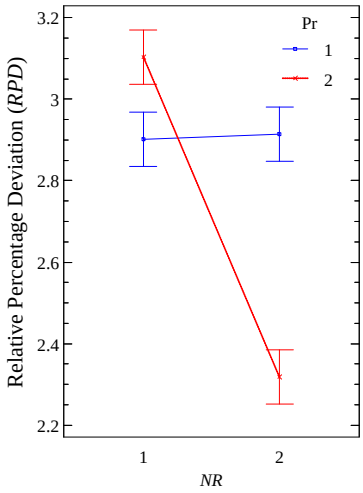

Figure 4: Means plot and 99% confidence level Tukey's HSD intervals for the interaction between the solution representation (Pr) and the job to factory assignment rule (NR) factors in the second calibration experiment for the proposed Iterated Greedy methods.

As a result, it is preferable to employ AC_3 as it is equivalent performance wise and at the same time simpler with one less parameter. In any case, for the final experiments we will also test the original [20] acceptance criterion (AC_1) to conclude in a sound way if our new acceptance criterion is actually equivalent or not. Finally, d is marginally significant, offering different results when related with the instance factors (n, m, f and t). Since we want to avoid an instance-specific factor level, we finally settle for $d = 5\%$ regardless of instance size.

7. Computational evaluation

We are now ready to computationally test the proposed approaches. We are going to compare first the four proposed simple constructive heuristics $CH_{11}, CH_{12}, CH_{21}$ and CH_{22}. These are very fast methods and take very little CPU time. We employ the same computing platform used for the calibration in the tests.

As mentioned, the benchmark of test instances is different from the previous calibration instances. Recall that in the calibration instances we have 60 random combinations of number of jobs (n), machines (m), factories (f), products (t) and distributions for the setup times of production and assembly machines. In the test instances we consider all possible combinations ($2 \times 3^3 \times 2 = 108$). For each combination we generate five different instances

resulting in a total of 540 test instances. For all tested methods we calculate the Relative Percentage Deviation from the best solution known. This solution is the best obtained throughout the course of this paper. All instances as well as the best solutions are available at `http://soa.iti.es`.

Table 4 shows the results of the four tested heuristics. There are 540 instances and four tested heuristics. Therefore, the total number of results is 2,160. We have grouped these by instance characteristics. CPU times are not reported as they are extremely small. As a matter of fact, among the 2,160 observed CPU times in the results, the maximum reported is just 0.079 seconds. The average observed CPU time in all results is only 0.008 seconds. It can be concluded that the reported heuristics are almost instantaneous even for the largest tested instances of 200 jobs, 20 machines, 8 factories and 50 products.

		RPD			
		CH_{11}	CH_{12}	CH_{21}	CH_{22}
n	100	21.17	20.02	22.36	22.12
	200	13.11	12.24	13.22	13.31
m	5	16.01	14.82	18.04	17.94
	10	16.95	16.24	18.06	17.94
	20	18.46	17.34	17.27	17.25
f	4	18.69	17.66	18.65	18.41
	6	16.89	15.47	17.55	17.53
	8	15.83	15.27	17.16	17.19
t	30	15.58	14.58	14.53	14.47
	40	17.92	16.77	18.37	18.37
	50	17.92	17.04	20.47	20.30
Setup interval	$U[1, 50]$	12.70	12.11	10.43	10.31
	$U[1, 125]$	21.58	20.15	25.15	25.11
Average		17.14	16.13	17.79	17.71

Table 4: Average Relative Percentage Deviation (RPD) over the best known solution, grouped by instance characteristics of the proposed constructive heuristics.

As can be seen, all four heuristics provide similar results. The average deviations are between a little more than 16% and below 18%. Although not detailed here, there is a large variability in the results as well. The minimum observed RPD is just 3.59% and the maximum 51.51%. In order to closely analyze these results, we carry out an ANOVA statistical test on the obtained results. We consider all instance factors (n, m, f and t) as non-controllable factors as well as a single factor which is the heuristic, at four variants. The results of the ANOVA, which are not shown here due to reasons of space,

indicate that three non-controllable factors n, t and f are very significant, in this order. This is expected as with more jobs and products the instances are harder to solve. Note, however, that a larger number of factories results in easier instances as there are less jobs per factory. As for the algorithms, the result is that CH_{12} is statistically better than the rest, followed by CH_{11} which is in turn better than CH_{21} and CH_{22}. There are no statistically significant differences between these last two methods. Note that this is not a contradictory result. While in the heuristic testing, CH_{12}, is the best heuristic, the calibration experiments for VND and IG resulted in CH_{21} being the best initialization method. We should not assume that the best heuristic should be used as an initialization for a metaheuristic as the initialization interacts with all other algorithm parameters.

In a separate experiment we test the more time consuming methods. The algorithms to compare are the VND with the parameters obtained in the calibration (Pr= 2, $NR = 2$ and $INI = CH_{21}$) and two similarly configured IG methods also from the calibration result. These differ only in the acceptance criterion. The common parameters are Pr= 2, $NR = 2$, $INI = CH_{21}$ and $d = 5\%$. In the first tested IG, referred to as IG_1, we employ the original [20] acceptance criterion, AC_1 (which is, in turn, based on the criterion of 44). Since we need a value for T in this acceptance criterion, we use $T = 2.5$ as per the result of the calibration. The second tested IG, referred to as IG_3, uses the third proposed acceptance criterion AC_3 which does not have a T parameter.

The two Iterated Greedy methods need a termination criteria which is tested at two levels: $n \cdot m \cdot f \cdot 30$ and $n \cdot m \cdot f \cdot 60$ milliseconds elapsed CPU time ($\rho = 30, 60$). Additionally, since IG is stochastic, we run it five times for each instance and CPU time termination. Conversely, VND is deterministic and does not have a termination criterion and is therefore run only once with each instance. In total we have 540 results for the VND and 2,700 for each IG method and termination criterion (10,800 results). We first present the average Relative Percentage Deviation over the best solutions known for each instance. Table 5 shows these results, grouped by instance characteristics, among other information regarding CPU times.

As can be seen, VND results in relatively good solutions which average a RPD of 5.33% in all tests. The average CPU time needed is a little more than 37 seconds. Note how the CPU times clearly depend on the size of the instance (number of jobs n, number of machines m and number of products t). The proposed Iterated Greedy methods are tested at two termination

		RPD					CPU times (sec.)		
		IG_1		IG_3			$IG_{1/3}$	$IG_{1/3}$	
		$\rho = 30$	$\rho = 60$	$\rho = 30$	$\rho = 60$	VND	$\rho = 30$	$\rho = 60$	VND
n	100	2.39	1.33	2.23	**1.26**	8.02	210	420	18.04
	200	0.73	0.43	0.67	**0.37**	2.64	420	840	56.87
m	5	1.77	0.97	1.66	**0.93**	5.35	135	270	23.12
	10	1.54	0.88	1.44	**0.82**	5.45	270	540	32.91
	20	1.36	0.79	1.26	**0.69**	5.20	540	1080	56.33
f	4	1.43	0.77	1.33	**0.72**	4.37	210	420	37.44
	6	1.60	0.88	1.50	**0.83**	5.49	315	630	39.26
	8	1.64	0.99	1.52	**0.89**	6.14	420	840	35.65
t	30	0.65	0.49	0.58	**0.41**	3.18	315	630	15.42
	40	1.41	0.85	1.32	**0.79**	5.59	315	630	33.11
	50	2.61	1.30	2.45	**1.24**	7.23	315	630	63.82
Setup interval	$U[1, 50]$	0.93	0.53	0.89	**0.49**	3.12	315	630	37.55
	$U[1, 125]$	2.18	1.23	2.01	**1.14**	7.55	315	630	37.35
Average		1.56	0.88	1.45	**0.81**	5.33	315	630	37.45

Table 5: Average Relative Percentage Deviation (RPD) over the best known solution, grouped by instance characteristics and average CPU times of the proposed algorithms. Bold values indicate the best obtained average relative percentage deviations.

criteria and it is clear that with double the CPU time, the results improve. An interesting conclusion is that the third acceptance criterion (AC_3), albeit simpler and with one less parameter, gives better results when compared with the regular acceptance criterion. It is safe to conclude that IG_3, a simpler version with only one main parameter compared to the original version of [20], works better for the studied problem.

We also carry out a multi-factor ANOVA to check if the observed average differences from Table 5 are indeed statistically significant. Once again we consider all instance characteristics as non-controllable factors. Preliminary tests indicate that VND is clearly not statistically better than the IG methods. Therefore, to avoid lack of normality in the residuals and to have a clearer picture of the performance of the IG methods, VND is removed from the final statistical test. We control two factors, the type of IG at two variants (IG_1 and IG_3) and the termination time ρ at two levels (30 and 60). The results of the ANOVA indicate that IG_3 is statistically better than IG_1 in most instances except for the easy ones, this is further illustrated in Figure 5.

From the results we have shown that the proposed heuristics provide reasonable results almost instantaneously whereas the presented VND method gives much better results which deviate, on average, about a 5% from the best

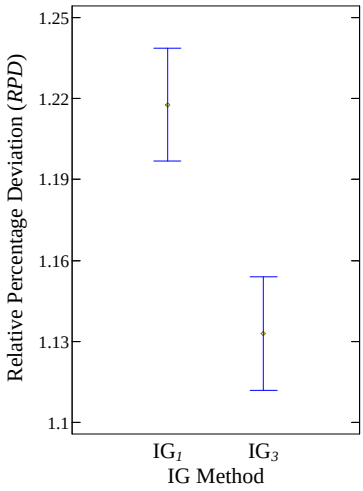

Figure 5: Means plot and 99% confidence level Tukey's HSD intervals for the type of Iterated Greedy method in the final test experiments.

known solutions. When doing so they require a larger, but still acceptable CPU time. The presented Iterated Greedy algorithms are of a much higher quality but need more CPU time. This time, however, can be set by the decision maker. With all these tools, plant managers have a wide range of algorithms with different CPU time demands and solution qualities to suit the needs of each moment.

8. Conclusions and future research

We have addressed the addressed Distributed Assembly Permutation Flowshop Scheduling Problem with the additional consideration of sequence dependent setup times at both production and assembly stages. This results in a considerably more realistic and applicable problem setting. The objective is the minimization of the makespan at the assembly stage.

We have presented two constructive heuristics, which are combined with two existing job to factory assignment rules. Furthermore, a simple and relatively fast metaheuristic based on Variable Neighborhood Descent (VND) is proposed, calibrated and analyzed. Additionally, we present an Iterated Greedy (IG) algorithm that has also been extensively analyzed. While IG is a very simplistic metaheuristic, we have simplified it further by proposing an acceptance criterion that does not consider a simulated annealing-like temperature as is common in the IG literature (20). The result is a parameter-less acceptance criterion.

Sound and detailed statistical techniques have been employed to calibrate and to analyze the performance of all presented methods. The result is a battery of approaches that range from very fast (almost instantaneous) constructive heuristics that produce reasonably good results to more time consuming methods like VND or IG that reach close to optimality performance. Given the applicability of the researched problem and the range of proposed approaches, this work represents a solid step forward in solving more realistic distributed scheduling problems.

Future research includes the consideration of additional characteristics in the problem setting to make it even more realistic. For example, a transportation time or stage in between the production and assembly stages might prove useful. Additionally, heterogeneous distributed factories could account for more complex scenarios. Furthermore, the assembly stage could be made more complex with parallel machines, for example. Of course, more elaborate solution approaches could further improve the results obtained in this paper.

Acknowledgments

Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "RESULT - Realistic Extended Scheduling Using Light Techniques" with reference DPI2012-36243-C02-01 co-financed by the European Union and FEDER funds and by the Universitat Politècnica de València, for the project MRPIV with reference PAID/2012/202. Carlos Andrés is partially supported by the Spanish Ministry of Science and Innovation, under the project "INSAMBLE - Scheduling at assembly/disassembly synchronized supply chains" with reference DPI2011-27633.

References

[1] C. Koulamas, G. Kyparisis, The three stage assembly flowshop scheduling problem, Computers & Operations Research 28 (2001) 689–704.

[2] C. Lee, T. Cheng, B. Lin, Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem, Management Science 39 (1993) 616–625.

[3] C. N. Potts, S. V. S. Janov, V. A. Strusevich, L. N. V. Wassenhove, C. M. Zwaneveld, The two-stage assembly scheduling problem: Complexity and approximation, Operations Research 43 (1995) 346–355.

[4] F. T. S. Chan, S. H. Chung, P. L. Y. Chan, An adaptive genetic algorithm with dominated genes for distribute scheduling problems, Expert Systems with Applications 29 (2005) 364–371.

[5] C. Moon, J. Kim, S. Hur, Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain, Computers & Industrial Engineering 43 (2002) 331–349.
[6] B. Wang, Integrated product, process and enterprise design, Chapman & Hall, London, 1997.
[7] H. Jia, A. Nee, J. Fuh, Y. Zhang, A modified genetic algorithm for distributed scheduling problems, Journal of Intelligent Manufacturing 14 (2003) 351–362.
[8] K. B. Kahn, G. A. Castellion, A. Griffin, The PDMA handbook of new product development, Wiley, New York, second edition, 2004.
[9] W. Yang, C. Liao, Survey of scheduling research involving setup times, International Journal of Systems Science 30 (1999) 143–155.
[10] A. Allahverdi, J. N. D. Gupta, T. Aldowaisan, A review of scheduling research involving setup considerations, Omega, The International Journal of Management Science 27 (1999) 219–239.
[11] A. Allahverdi, C. Ng, T. Cheng, M. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal of Operational Research 187 (2008) 985–1032.
[12] T. Cheng, J. N. D. Gupta, G. Wang, A review of flowshop scheduling research with setup times, Production and Operations Management 9 (2000) 262–282.
[13] K. Baker, Introduction to sequencing and scheduling, Wiley, New York, 1974.
[14] J. M. Framinan, J. N. D. Gupta, R. Leisten, A review and classification of heuristics for permutation flow-shop scheduling with makespan objective, Journal of the Operational Research Society 55 (2004) 1243–1255.
[15] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2005) 479–494.
[16] S. Hejazi, S. Saghafian, Flowshop-scheduling problems with makespan criterion: a review, International Journal of Production Research 43 (2005) 2895–2929.
[17] J. N. D. Gupta, E. F. Stafford, Jr, Flowshop scheduling research after five decades, European Journal of Operational Research 169 (2006) 699–711.
[18] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Computers & Operations Research 37 (2010) 754–768.
[19] S. Hatami, R. Ruiz, C. Andrés-Romano, The distributed assembly permutation flowshop scheduling problem, International Journal of Production Research 51 (2013) 5292–5308.
[20] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (2007) 2033–2049.
[21] R. Ruiz, T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, European Journal of Operational Research 187 (2008) 1143–1159.
[22] A. Hariri, C. Potts, A branch and bound algorithm for the two-stage assembly scheduling problem, European Journal of Operational Research 103 (1997) 547–556.
[23] O. Tozkapan, A. Kırca, C.-S. Chung, A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem, Computers & Operations Research 30 (2003) 309–320.
[24] F. Al-Anzi, A. Allahverdi, A hybrid tabu search heuristic for the two-stage assembly scheduling problem, The International Journal of Operational Research 3 (2006) 109–119.
[25] F. Al-Anzi, A. Allahverdi, Heuristics for a two-stage assembly flowshop with bicriteria of maximum lateness and makespan, Computers & Operations Research 36 (2009) 2682–2689.
[26] X. Sun, k. Morizawa, H. Nagasawa, Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling, European Journal of Operational Research 146 (2003) 498–516.
[27] J. Gao, R. Chen, D. W., An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, International Journal of Production Research 51 (2013) 641–651.

[28] S.-W. Lin, K.-C. Ying, C.-Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, International Journal of Production Research 51 (2013) 5029–5038.
[29] S.-Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, International Journal of Production Economics 145 (2013) 387–396.
[30] M. Yokoyama, Scheduling for two-stage production system with setup and assembly operations, Computers & Operations Research 31 (2004) 2063 – 2078.
[31] M. Yokoyama, Flow-shop scheduling with setup and assembly operations, European Journal of Operational Research 187 (2008) 1184–1195.
[32] F. Al-Anzi, A. Allahverdi, A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times, European Journal of Operational Research 182 (2007) 80–94.
[33] A. Allahverdi, F. Al-Anzi, The two-stage assembly scheduling problem to minimize total completion time with setup times, Computers & Operations Research 36 (2009) 2740–2747.
[34] S. Hatami, S. Ebrahimnejad, R. Tavakkoli-Moghaddam, Y. Maboudian, Two meta-heuristics for the three-stage assembly flowshop scheduling with sequence-dependent setup times, The International Journal of Advanced Manufacturing Technology 50 (2010) 1153–1164.
[35] A. Mozdgir, S. F. Ghomi, F. Jolai, J. Navaei, Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times, International Journal of Production Research 51 (2013) 3625–3642.
[36] P. Hansen, N. Mladenović, Variable neighborhood search: Principles and applications, European Journal of Operational Research 130 (2001) 449–467.
[37] I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for the flowshop scheduling problem with blocking, OMEGA, The International Journal of Management Science 39 (2011) 293–301.
[38] Q.-K. Pan, L. Wang, B. H. Zhao, An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion, International Journal of Advanced Manufacturing Technology 38 (2008) 778–786.
[39] Q.-K. Pan, R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle flowshop scheduling problem, OMEGA, The International Journal of Management Science 44 (2014) 41–50.
[40] J. M. Framinan, R. Leisten, Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm, International Journal of Production Research 46 (2008) 6479–6498.
[41] Q.-K. Pan, R. Ruiz, Local search methods for the flowshop scheduling problem with flowtime minimization, European Journal of Operational Research 222 (2012) 31–43.
[42] G. Minella, R. Ruiz, M. Ciavotta, Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems, Computers & Operations Research 38 (2011) 1521–1533.
[43] M. Ciavotta, G. Minella, R. Ruiz, Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study, European Journal of Operational Research 227 (2013) 301–313.
[44] I. Osman, C. Potts, Simulated annealing for permutation flow-shop scheduling, Omega, The International Journal of Management Science 17 (1989) 551–557.
[45] D. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, eight edition, 2012.
[46] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, New York, 2010.
[47] D. Basso, M. Chiarandini, L. Salmaso, Synchronized permutation tests in replicated $i \times j$ designs., Journal of Statistical Planning and Inference 137 (2007) 2564–2578.
[48] D. Rasch, V. Guiard, The robustness of parametric statistical methods, Psychology

Science 46 (2004) 175–208.
[49] E. Ridge, D. Kudenko, Tuning an algorithm using design of experiments, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, New York, 2010, pp. 265–286.