European Journal of Operational Research 239 (2014) 323–334

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A scatter search algorithm for the distributed permutation flowshop scheduling problem

Bahman Naderi [a], Rubén Ruiz [b,*]

[a] Department of Industrial Engineering, Faculty of Engineering, University of Kharazmi, Karaj, Iran
[b] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Universitat Politècnica de València, Edifico 8G, Acc. B, Camino de Vera s/n, 46021 València, Spain

ARTICLE INFO

Article history:
Received 11 January 2014
Accepted 15 May 2014
Available online 29 May 2014

Keywords:
Distributed scheduling
Permutation flowshop
Scatter search

ABSTRACT

The distributed permutation flowshop problem has been recently proposed as a generalization of the regular flowshop setting where more than one factory is available to process jobs. Distributed manufacturing is a common situation for large enterprises that compete in a globalized market. The problem has two dimensions: assigning jobs to factories and scheduling the jobs assigned to each factory. Despite being recently introduced, this interesting scheduling problem has attracted attention and several heuristic and metaheuristic methods have been proposed in the literature. In this paper we present a scatter search (SS) method for this problem to optimize makespan. SS has seldom been explored for flowshop settings. In the proposed algorithm we employ some advanced techniques like a reference set made up of complete and partial solutions along with other features like restarts and local search. A comprehensive computational campaign including 10 existing algorithms, together with statistical analyses, shows that the proposed scatter search algorithm produces better results than existing algorithms by a significant margin. Moreover all 720 known best solutions for this problem are improved.

1. Introduction

Scheduling deals with the allocation of resources, typically machines, to tasks (commonly referred to as jobs) over time with the goal of optimizing a given objective (Pinedo, 2012). Scheduling is an important problem that appears mainly in manufacturing industries. It is well known that good schedules contribute greatly to the overall performance of a company (McKay, Pinedo, & Webster, 2002). The layout of the machines on the production floor, along with the flow of the jobs in the machines, together with a myriad of constraints and real life settings determine the type of scheduling problem to solve. The flowshop scheduling problem (FSP) is arguably the most common processing layout in practice as it is typical for manufacturing plants to manufacture a given family of products that have to visit machines in a known order. For example, in car manufacturing, the painting of the car body must go after the body as been welded and before any assembly operation, hence a flowshop structure. Reisman, Kumar, and Motwani (1997) reviewed practical cases and concluded that the flowshop problem has many real life applications. This applicability of the flowshop is also highlighted in the many exiting reviews from the literature like Framinan, Gupta, and Leisten (2004), Ruiz and Maroto (2005), Hejazi and Saghafian (2005) and Gupta and Stafford (2006). As a matter of fact, once generalized to hybrid flowshops or flexible flowline problems, many production problems can be modeled after a flowshop (Linn & Zhang, 1999; Vignier, Billaut, & Proust, 1999; Wang, 2005; Quadt & Kuhn, 2007; Ruiz & Vázquez-Rodríguez, 2010; Ribas, Leisten, & Framinan, 2010). The FSP can be formally described as follows: A set N of n different and independent jobs has to be scheduled. Jobs usually model client orders or batches of products to be manufactured after a production planning process has been carried out (Pochet & Wolsey, 2006). Each job j, $j \in N$ has to visit, in order, all m machines in the set of machines M. Without loss of generality, each job visits first machine 1, then machine 2 and so on until machine m. A job cannot go to the next machine until it is finished in the current machine and a machine cannot process more than one job at the same time. As a result of the machines being disposed in series, each job is broken down into m tasks, one per machine. Each task from a job j, $j \in N$ needs a given processing time at each machine i, $i \in M$. This processing time is denoted as p_{ij} and it is deterministic, known in advance and usually non-negative, represented by an integer quantity.

The objective in the FSP is to find a schedule or processing sequence of all the jobs in the machines such that a given

* Corresponding author. Tel.: +34 96 387 70 07; fax: +34 96 387 74 99.
E-mail addresses: bahman.naderi@aut.ac.ir (B. Naderi), rruiz@eio.upv.es (R. Ruiz).

http://dx.doi.org/10.1016/j.ejor.2014.05.024
0377-2217/

optimization criterion is optimized. According to the previously cited review papers, the most commonly studied objective is the minimization of the maximum completion time or makespan, denoted as C_{max}. Given the completion time of a job in the last machine m, denoted as C_j, the makespan is then the minimization of the maximal C_j, $j = 1, \ldots, n$. Since there are as many possible schedules as job sequences at each machine, the total number of solutions is $(n!)^m$, i.e., all possible job permutations at each machine, considering that these permutations can change from machine to machine. Given this huge search space, the FSP is usually simplified to what is called the Permutation Flowshop Scheduling Problem or PFSP by forbidding job-passing, i.e., once the production sequence is fixed for a machine, all machines follow the same production sequence. This brings down the total number of solutions to $n!$. Using the well known three field notation for scheduling problems (Graham, Lawler, Lenstra, & Rinnooy Kan, 1979; Pinedo, 2012), the PFSP with makespan criterion is denoted as $F/prmu/C_{max}$.

This problem was first studied almost 60 years ago by Johnson (1954) where the well known Johnson's algorithm was proposed for solving the two machine version. For three or more machines, the problem is known to be $\mathcal{NP}$-Complete in the strong sense Garey, Johnson, and Sethi (1976). Nowadays, the literature on the PFSP is immense and the problem and many variants have been thoroughly studied. The topic is so widely studied that there are even some dedicated monographs such as Chakraborty (2009) and Emmons and Vairaktarakis (2012), or even for some variants, like lot streaming in Sarin and Jaiprakash (2007). However, there is one extension that was only recently presented. In Naderi and Ruiz (2010) and Naderi and Ruiz (2010) studied a variant that was referred to as the Distributed Permutation Flowshop Scheduling Problem or DPFSP. In essence, the regular PFSP considers one single factory where products are manufactured. However, multi-factory enterprises are much more competitive in a globalized economy. The literature on manufacturing systems abounds with examples where it is shown that distributed manufacturing is key for high product quality, low production costs and reduced management risks, among many other benefits (Wang, 1997; Moon, Kim, & Hur, 2002; Kahn, 2004, among many others). Distributed manufacturing is now a topic of interest as the recent editorial in a special issue of a reputable manufacturing journal shows (Chan & Chung, 2013). In that editorial and in many of the papers of the cited special issue the importance and benefits of distributed manufacturing are praised and highlighted. In the DPFSP there is an important added complexity with respect to the PFSP: Jobs need to be assigned to factories and then a schedule must be built for each factory. More formally, the DPFSP extends the regular permutation flowshop in the following way: The set N of n jobs must be processed by a set G of F identical factories. Each factory has the same set M of m machines. The processing times of all the tasks of a given job do not change from factory to factory. Once assigned to a factory, a job has to be completed in that factory. The objective is to minimize the maximum makespan among all factories. Naderi and Ruiz (2010) referred to this problem as $DF/prmu/C_{max}$. The same authors demonstrated that no factory must be left empty with no jobs assigned (given $n > F$) as this does not improve the makespan value. They also concluded that the total number of solutions in the DPFSP is $\binom{n-1}{F-1} n!$ Additionally, since the DPFSP reduces to the regular PFSP if $F = 1$, it is easy to conclude that the DPFSP is also an $\mathcal{NP}$-Hard problem.

From the paper of Naderi and Ruiz (2010), several other authors built upon those results and several methodologies have been proposed to solve this new problem. Naderi and Ruiz (2010) proposed some mathematical models, simple heuristics and local search methods. Therefore, more complex methodologies might reveal new interesting solutions to this hard combinatorial problem. Furthermore, given the existing recent methods proposed, it is also worthwhile comparing the effectiveness and efficiency of existing approaches to ascertain which are the state-of-the-art methods. These are some of the objectives of this paper. When deciding about which advanced techniques could be applied to the DPFSP we observed that simple local search based methods failed to escape strong local optima and therefore we chose a powerful methodology: Scatter Search (Glover, Laguna, & Martí, 2000; Laguna & Martí, 2003; Martí, Laguna, & Glover, 2006, among others). Contrary to many existing metaheuristic frameworks, which have been applied several times to flowshop problems, scatter search (SS) has seldom been used for these scheduling settings. References with applications of scatter search to regular flowshops are scarce. Nowicki and Smutnicki (2006) presented some methods, including ideas from path relinking and scatter search to the regular PFSP with makespan criterion but failed to significantly advance the state-of-the-art. In a short paper, Saravanan, Noorul Haq, Vivekraj, and Prasad (2008) proposed another scatter search method for the same problem and reported average percentage deviations over the best known solutions for the benchmark of Taillard (1993) of a little over 1%. This is clearly not better than the deviations below 0.5% given by the simpler Iterated Greedy (IG) method of Ruiz and Stützle (2007) or the deviations of just 0.22% given in Vallada and Ruiz (2009). As regards the PFSP, it seems that there are no other noteworthy scatter search applications. Therefore, it is plausible to think that scatter search methods for flowshop problems still have some headroom for improvement and therefore we choose them for this paper. Furthermore, the controlled diversification in scatter search shows, as we will empirically demonstrate, great strength in the DPFSP.

The remainder of this paper is organized as follows: Section 2 provides a comprehensive literature review on the DPFSP. Section 3 presents in detail the proposed scatter search approach. This method is calibrated in Section 4. In the same Section, almost all relevant algorithms from the literature on the DPFSP are reimplemented and carefully evaluated. Through comprehensive computational and statistical analyses we show that the presented scatter search algorithm can be considered as the new state-of-the-art method for the DPFSP and makespan minimization. Finally, Section 5 concludes this paper and proposes some avenues for future research.

2. Literature review

In Naderi and Ruiz (2010) the authors presented six different Mixed Integer Linear Programming models for the DPFSP together with 12 heuristics that resulted from applying two different job to factory assignment rules to six famous heuristics for the regular flowshop problem. The two rules are the following:

- Assign a given job j to the factory with the lowest current C_{max}, not including job j.
- Assign job j to the factory which completes it at the earliest time, i.e., the factory resulting in the lowest C_{max} after assigning job j.

The rules are applied each time a job is scheduled. From the six tested heuristics the NEH method of Nawaz, Enscore, and Ham (1983) with the second job to factory assignment rule (referred to as NEH2) resulted in the best heuristic performance. Apart from the heuristic methods, Naderi and Ruiz (2010) presented a simple Variable Neighborhood Descent (VND, Mladenović & Hansen (1997)) starting with the NEH2 solution and with two neighborhoods. One being the insertion local search for all factories (until

local optima at each factory) and the second local search takes the factory generating the makespan value and extracts all of its jobs and testes them in all other factories. Two different acceptance criteria are used: (a) accept the new solution if the critical makespan (the largest makespan among all factories) is reduced and (b) accept the solution if there is a net gain in the makespan values between the involved factories in the local search. More details are given in Naderi and Ruiz (2010). The resulting VND methods with both acceptance criteria were referred to as VND(a) and VND(b), respectively. The experimental results showed that VND(a) produced an average percentage deviation over the best known solutions for large problems of up to 500 jobs, 20 machines and 7 factories of just 0.10%. Note that all presented methods by Naderi and Ruiz (2010) are fast, as the slowest method – VND(a) – needed less than 0.15 seconds on average on a Intel Core 2 Duo computer running at 2.4 Gigahertz with 2 Gigabytes of RAM memory.

As Naderi and Ruiz (2010) pointed out, prior to 2010 there was almost no literature on distributed flowshop scheduling apart from some loosely related papers. However, after the publication of that paper, several authors published follow up studies. The first was the work of Liu and Gao (2010). The authors presented a complex electromagnetism metaheuristic (referred to as EM in this paper). They improved the VND local search of Naderi and Ruiz (2010) and extended it to a more powerful Variable Neighborhood Search (VNS, Mladenović & Hansen (1997)) with several neighborhoods such as insertion within the critical factory (the one generating the makespan), swap in the critical factory and general insertion and swap. In their computational evaluation, Liu and Gao (2010) did not directly compare against the VND(a) but rather pointed out the improvement of 151 best known solutions out of the 720 large instances presented in Naderi and Ruiz (2010). As we will later highlight, these comparisons can be misleading. Furthermore, the CPU times of the EM method are significantly larger than those of VND(a). Therefore, it remains to be seen if EM is competitive with VND(a).

Later, Gao and Chen (2011a) presented a Hybrid Genetic Algorithm with local search (GA_LS) which we simply refer to as HGA. The genetic method is inspired by the GA for the regular permutation flowshop of Ruiz, Maroto, and Alcaraz (2006). The algorithm employs NEH2 and VND(a) as initialization. The local search phase is similar to that of VND(a) but a third neighborhood is included in which exchange of jobs from the critical factory and all other jobs in all other factories are tested. In their experiments, HGA reported better solutions than VND(a) but again at the expense of much larger CPU times. According to the results of Gao and Chen (2011a), their HGA method uses almost 246 times more CPU time than VND(a). In the same paper the authors test their proposed HGA with the same CPU time as VND(a) and the results are quite the contrary with HGA showing apparently worse performance than VND(a). Therefore, another interesting experiment is to test HGA versus VND(a) in a completely comparable scenario.

Gao and Chen (2011b) presented an improvement of the NEH heuristic of Nawaz et al. (1983) and the NEH2 of Naderi and Ruiz (2010). The enhancement consists of inserting F jobs at a time (one to each factory) instead of one job at a time as it is usual in the NEH heuristic. This multi-insertion is carried out through an unspecified branch and bound procedure and the authors also employ the previously commented second job to factory assignment rule as well as other published improvements of the NEH. The best combined proposed method is referred to as NEHdf. In the computational experiments, NEHdf is shown to slightly outperform NEH2 (however, in a provided statistical experiment, NEHdf is not shown to statistically outperform NEH2). Again, this outperformance comes at an additional CPU cost.

More recently, Gao, Chen, and Liu (2012b), presented a genetic algorithm which is shown to slightly outperform the HGA of Gao and Chen (2011a). In their comparisons, the average relative percentage deviation of the new algorithm, referred to as GA_KB, is reduced by 0.3%, which is a rather marginal improvement. The CPU times are also slightly reduced but remain more than 200 times larger than those of VND(a).

In the same year, related authors (Gao, Chen, Deng, & Liu, 2012a) have presented a revised VNS method. Basically, the authors mix VND(a) of Naderi and Ruiz (2010) with their improved NEHdf method presented in Gao and Chen (2011b). The resulting algorithm is refereed to as VNS(B&B). Computational analyses show that VNS(B&B) is superior to VND(a) but obviously at the expense of additional CPU time.

More recently, the rate of publications in the DPFSP area is increasing. Gao, Chen, and Deng (2013) have presented a tabu search method. The proposed algorithm builds upon the local search schemes presented in Gao and Chen (2011a) and includes some more extended local search processes. In the experimental section this new TS method is shown to outperform the HGA of Gao and Chen (2011a) by a good margin, improving also the computational efficiency. However, from the tables given in Gao et al. (2013), the proposed TS is still almost 117 times slower than VND(a).

Also recently, Lin, Ying, and Huang (2013) have presented an iterated greedy method inspired by the work of Ruiz and Stützle (2007). Four IG variants are presented and the best one, denoted as IG_{VST} is compared against the HGA of Gao and Chen (2011a) and the TS of Gao et al. (2013). The results favor the IG_{VST} method by a wide margin and also with greatly reduced CPU times albeit the conditions are not fully comparable and the reported CPU times are still much larger than those of VND(a).

After all experimentation and analyses of this paper had been finished we became aware of a recently published paper (Wang, Wang, Liu, & Xu, 2013). The authors have presented an estimation of distribution algorithm (EDA). While the proposed method is shown to outperform VND(a), the new solutions obtained are not as good as those reported in other recent papers. Furthermore, the presented algorithm is much slower than VND(a), needing no less than almost 788 times more CPU time than VND(a).

As we can see, a myriad of metaheuristic methods have been recently presented for the DPFSP. As our critical review shows, many of these methods have not been compared against each other. Most comparisons are done against VND(a), which is basically a heuristic improved by some local search mechanism. Newer and more advanced methods might improve the solutions much further.

3. Scatter search method

Scatter search is a type of evolutionary algorithm which is strongly based on a principled approach to solution generation and recombination and steers away from the randomness of other evolutionary methods like genetic algorithms. The main characteristic of SS is the diversification of solutions as a means for high quality optimization. Its roots date back to the 1970s with the works of Glover (1977) or Glover (1998), to be later formalized in Glover et al. (2000); Laguna and Martí (2003) or Martí et al. (2006) to name just a few.

The SS employed in this paper follows the basic template given in Laguna and Martí (2003) and in Martí et al. (2006) which is based on the known "'five methods": (1) Diversification generation method. The initial population of the method is created using an input solution. Here a pool P of $PSize$ diverse solutions is created. (2) Improvement method. A mechanism, usually a form of local

search, to improve solutions from any of the working sets. Normally it is also applied to the set P at the beginning of the SS procedure. (3) Reference set update method. In SS the reference set or *RefSet* usually contains the b best solutions of P initially. This set is desired to be as diverse as possible, so selecting not only the best but also the maximally diverse solutions is preferable. *RefSet* is an ordered list with the best solution first. As the SS method iterates, new solutions enter *RefSet* according to their quality and diversity. (4) Subset generation method. Here, some solutions from *RefSet* are selected for later processing. The simplest procedure is to generate all possible pairs of solutions from *RefSet* as subsets. (5) Solution combination method. The selected solutions in the subset generation method are recombined to create new solutions. Normally, new solutions are enhanced with the improvement method and later considered for insertion in *RefSet* in the reference set update method. The entire process iterates while there are changes in *RefSet*, i.e., while new different solutions are being discovered. Let us instantiate all these methods in our proposed SS algorithm.

3.1. Solution representation and diversification generation method

In the PFSP literature, the most common solution representation is a permutation of the n jobs. Since in the DPFSP this permutation is divided among the F factories, the most straightforward representation is to have F lists, one per factory. Each list contains a partial permutation with the order in which the jobs have to be processed at each factory. This is the solution representation that Naderi and Ruiz (2010) and subsequent authors have employed. For example, if we have a problem with 10 jobs ($n = 10$) and three factories ($F = 3$), one possible solution is:

$$\begin{cases} 4, 8, 1 \\ 2, 10, 5 \\ 7, 6, 3, 9 \end{cases}$$

In this solution, jobs 4, 8 and 1 are assigned to factory 1 and follow that order, jobs 2, 10 and 5 to factory 2 and so on. The sequence at each factory is obtained by scanning each job list from left to right.

In our proposed SS procedure for the DPFSP we have two specially constructed sets inside the reference set. The first is set H which contains a number b of the best ever found solutions. The second set, denoted as S, is made up of l factory assignment vectors. The union of these two sets makes the reference set, i.e., $RefSet = H \cup S$ of size $b + l$. The sets are clearly different. Set H contains full solutions according to the aforementioned solution representation. However, set S only contains factory assignments for jobs, i.e., given a 10 job, 3 factory DPFSP instance, a member of the set S could be the following: $\{2, 3, 1, 1, 2, 2, 2, 3, 1, 3\}$ meaning that job 1 is assigned to factory 2, job 2 to factory 3 and so on until job 10 which is assigned to factory 3. These are not complete solutions but just factory assignments as no job ordering at each factory is given. The rationale behind these two distinct sets inside the reference sets will be clear after the solution combination method.

For the initial construction of set H we start with a *Psize* of 25 random job permutations. 24 of these permutations are used as an initial ordering that is passed to the NEH2 method of Naderi and Ruiz (2010). Recall that this is an extension of the NEH method of Nawaz et al. (1983). For the last 25th permutation we use the regular NEH initial ordering instead of random. Basically, in the NEH2, jobs are inserted, one by one and according to the initial ordering into all positions of all factories. The job is finally placed in the position resulting in the minimal partial makespan. The second job to factory assignment rule (see Section 2) is used. Let us give an example following the previous case with 10 jobs and 3 factories. Let us consider the initial ordering of jobs as $\{4, 2, 7, 6, 1, 3, 10, 5, 9, 8\}$. Starting from the following partial solution:

$$\begin{cases} 4, 1 \\ 2 \\ 7, 6 \end{cases}$$

the next job to insert is job 3, as 4, 2, 7, 6 and 1 (the previous jobs in the initial ordering) are already in the solution. Therefore, job 3 has to be inserted in 8 different positions in the previous solution, resulting in the following alternatives:

$$\begin{cases} 3, 4, 1 \\ 2 \\ 7, 6 \end{cases}, \begin{cases} 4, 3, 1 \\ 2 \\ 7, 6 \end{cases}, \begin{cases} 4, 1, 3 \\ 2 \\ 7, 6 \end{cases}, \begin{cases} 4, 1 \\ 3, 2, \\ 7, 6 \end{cases}$$

$$\begin{cases} 4, 1 \\ 2, 3, \\ 7, 6 \end{cases}, \begin{cases} 4, 1 \\ 2 \\ 3, 7, 6 \end{cases}, \begin{cases} 4, 1 \\ 2 \\ 7, 3, 6 \end{cases}, \begin{cases} 4, 1 \\ 2 \\ 7, 6, 3 \end{cases}$$

The alternative resulting in the best partial makespan is selected. In order to speed up the insertion procedure, the well known accelerations of Taillard (1990) are used. This procedure is applied to all job permutations to have 25 NEH2 improved solutions. Then, the best b solutions among these 25 are included in set H. Note that this applies to the initial H set construction. Later, at each iteration of the SS procedure, set H contains the best b visited solutions.

As for set S, used for diversification, we simply initialize it with random job to factory assignments. As we will see, at each iteration of the SS algorithm, sets H and S are combined. Therefore, and in order to keep the diversity, set S is randomly regenerated at each iteration of the SS method.

3.2. Subset generation and solution combination methods

In the proposed SS method, the subset generation method is also different from most scatter search applications given the nature of the two sets H and S inside *RefSet*. The procedure consists of selecting all possible combinations of solutions in set H with factory assignments in set S. Therefore, at each iteration, $b \cdot l$ pairs are considered. For example, let us suppose we have $b = 3$ and $l = 2$, i.e., $H = \{h_1, h_2, h_3\}$ and $S = \{s_1, s_2\}$. Therefore we have six combinations: $(h_1, s_1), (h_1, s_2), (h_2, s_1), (h_2, s_2), (h_3, s_1)$ and (h_3, s_2).

The combination method is crucial in the SS procedure. All pairs selected in the previous subset generation method undergo combination. We refer to the solution selected from set H as $p1$ and to the factory assignment vector selected from S as $p2$. The new combined solution, referred to as pn is at first identical to $p1$. The combination method has n iterations. At each iteration, a job from pn is randomly selected, without repetition, so at the end all jobs have been selected. We refer to this randomly selected job as h. If a random number uniformly distributed between 0 and 1 (*rand*) is less than a given value p the combination method checks if job h is assigned to different factories in pn and in the job to factory assignment vector $p2$. If this is the case, job h is extracted from its current factory in pn and tested in all possible positions of the factory indicated in $p2$. The final placement of job h is the position resulting in the lowest makespan at the factory indicated in $p2$. If *rand* is greater or equal than p then the job is not assigned to another factory and left untouched. Let us further illustrate the combination mechanism by applying it to an example with 10 jobs and 10 factories. Let us suppose the subsets are:

$$h_i = \begin{cases} 2, 5, 6, 1 \\ 10, 3, 7 \\ 4, 9, 8 \end{cases} \quad \text{and } s_j = \{3, 1, 2, 2, 1, 1, 3, 2, 1, 2\}$$

The randomly selected job is job 4 and the random value is 0.12 ($p = 0.2$); hence, we check job 4. This job in h_i is assigned to factory 3 and in s_j in factory 2. Therefore, we remove the job from factory 3 and assign to factory 2. To put this job into the sequence of jobs in factory 2, there are 4 possible positions as follows:

$$\begin{Bmatrix} 2,5,6,1 \\ 4,10,3,7, \\ 9,8 \end{Bmatrix} \begin{Bmatrix} 2,5,6,1 \\ 10,4,3,7, \\ 9,8 \end{Bmatrix} \begin{Bmatrix} 2,5,6,1 \\ 10,3,4,7, \\ 9,8 \end{Bmatrix} \begin{Bmatrix} 2,5,6,1 \\ 10,3,7,4 \\ 9,8 \end{Bmatrix}$$

The makespan of each solution is calculated and the solution resulting in the best makespan marks the new position for job 4. Suppose the next randomly selected job is job 8 and the random value is 0.43. Since this value is greater than $p = 0.2$, we skip changing the position of this job and go to the next job. The procedure repeats for all jobs. Fig. 1 shows a pseudoalgorithmic listing of the proposed combination method.

Note that the parameter p controls the intensity of the diversification. Too low of a p value and pn will be essentially similar to $p1$ whereas if p is large, most jobs will be assigned to different factories. Initial experiments indicated that a low p value of 0.1 sufficed to maintain the diversification. In Section 4.1 we will calibrate, using sound statistical techniques, other more important parameters of the proposed SS method.

3.3. Improvement method

The improvement procedure is applied at each iteration of the SS to each solution pn obtained by the solution combination method. Note that we do not apply it after the *RefSet* initial generation. The proposed method is a simplification of the VND procedure of Naderi and Ruiz (2010). Two local search procedures are iteratively applied until the improved solution is a local optima with respect to both neighborhoods. More precisely, in the first local search, for each factory, each job assigned to that factory is extracted and inserted into all possible positions of the sequence at that factory. The position resulting in the best makespan for that factory is chosen. If there has been an improvement in the makespan value for that factory, the procedure is repeated. Therefore, at the end of this first local search, each factory contains local optima solutions with respect to the insertion neighborhood. Note that the accelerations of Taillard (1990) are also applied here.

In the second local search, each job from the critical factory (the factory with the maximal makespan value) is extracted and inserted into all possible positions of the sequence of all other factories. The procedure continues while no improvements in the maximal makespan are found. However, once the maximal makespan is improved, the second local search terminates and we go back to the first local search scheme as in a Variable Neighborhood Descent (VND) method. Contrarily, the process terminates (and the VND too) if all jobs from the critical factory are inserted into all positions of all other factories unsuccessfully. Again the accelerations of Taillard (1990) are used in the second local search as well. It is important to note also that after an improvement in the maximal makespan, only two factories are affected (the one from which the job has been extracted and the one to which the job has been inserted) therefore, when applying again the first local search procedure, only these two factories are examined.

3.4. Reference set update method and restart procedure

After the improvement method is applied to pn we need to check if this new solution is incorporated into the set H of *RefSet*. Inspired by the generational schemes of Ruiz et al. (2006) and Vallada and Ruiz (2010), pn is included into H if and only if: (1) the makespan of pn is better than the makespan of the worst solution in set H and (2) it is unique, i.e., there are no other identical solutions in set H.

If all conditions are satisfied, pn substitutes the worst solution in set H, otherwise, pn is simply discarded. Note that we tested some other more elaborated diversity mechanisms, like adding a third condition by which pn should not decrease the diversity of set H, even if better than the worst and strictly unique. However, continuously checking for diversity is expensive and after further detailed experiments and calibrations (not shown here due to space constraints) the results were not better. As a result, we dropped diversity checking from the proposed SS. This also simplifies the final algorithm. However, after initial experiments, the removal of the diversity checking also resulted in a fast convergence to local optima solutions. We have to consider that set H contains full solutions and these are never diversified after the initial diversification generation method. Only set S, which contains random job to factory assignments is randomly regenerated at each iteration. Therefore, we include a procedure to restart set H after a number of iterations without improvements in the best solution. The procedure is simple; after a iterations without improvements in the best solution, the worst 50% of solutions in set H are discarded and the diversification generation method is employed to generate new solutions. An important remark is that this restart procedure is applied until the best solution is improved, i.e., the counter of iterations without improvement is not reset after the restart procedure is applied.

The complete proposed SS method is given in pseudoalgorithmic form in Fig. 2.

4. Calibration, computational comparisons and statistical analyses

In this section we first calibrate the presented scatter search. Then we carry out a detailed and comprehensive computational comparison of the proposed scatter search method against the best

```
procedure Solution_Combination_Method(p, p1, p2)
 pn = p1
 for j := 1 to n do
  Take a random job, without repetition, from pn, let this job be h
  if (rand < p) then
   if factory assigned to h in pn <> factory assigned to h in p2 then
    Extract h from its factory in pn and assign it to the factory indicated in p2
    Insert h into all positions of the factory indicated in p2
    Place h at the position with the best C_max at the factory indicated in p2
   endif
  endif
 endfor
 return pn
end
```

Fig. 1. Solution combination method pseudoalgorithm.

```
procedure SS(b, l, a)
 Set p := 0.1; counter := 0
 Generate 25 solutions with the NEH2 heuristic  %diversification gen method
 Initialize set H with the best b solutions among the 25
 while (termination criterion not satisfied) do
  Generate new set S with l vectors randomly   %diversification gen method
  for i := 1 to b do        %subset generation method
   p1 = i-th solution from set H
   for j := 1 to l do       %subset generation method
    p2 = j-th solution from set S
    pn=solution_combination_method(p,p1,p2)
    pn'=solution_improvement_method(pn)
    reference_set_update_method(pn')
   endfor
  endfor
  if best solution in H has improved then counter := 0 else counter++
  if counter > a then apply restart_procedure to set H
 endwhile
end
```

Fig. 2. Proposed Scatter Search (SS) method. Note the parameters *b*, *l* and *a*.

existing methods from the literature. We carefully explain the aspects of the comparison, instances tested and all conditions that facilitate the generalization and replicability of the results obtained.

4.1. Calibration of the proposed scatter search method

We have chosen to calibrate only the most meaningful parameters. The size *b* of *RefSet* is typically not greater than 20 (Martí et al., 2006). According to these and other well known indications, the following factors are tested at the following levels, resulting in 48 combinations: (1) Size *b* of set *H* in the *RefSet*. Tested at four levels: $\{2, 5, 10, 15\}$. (2) Size *l* of set *H* in the *RefSet*. Tested at three levels: $\{2, 5, 10\}$ and (3) Number of iterations *a* before restart occurs. Tested at four levels: $\{10, 20, 30, 40\}$.

Naderi and Ruiz (2010) presented two sets of instances for the DPFSP. The first set contains 420 small instances of up to 16 jobs, 5 machines and 4 factories. These small instances were used for solving the proposed MILP models in that paper and are deemed too easy for calibration and testing. They are therefore not used in the remainder of this paper. Naderi and Ruiz (2010) also presented a set of 720 large instances based on the 120 instances of Taillard (1993) which has 12 sets with the following different combinations of number of jobs *n* and number of machines *m* ($n \times m$): $\{(20, 50, 100) \times (5, 10, 20)\}, \{200 \times (10, 20)\}$ and 500×20. Each combination has 10 replicates and therefore the 120 instances in total. All these 120 instances are considered with a different number of factories. We have $F = \{2, 3, 4, 5, 6, 7\}$, which gives us 720 instances in total. All instances are available from http://soa.iti.es.

It has to be noted that calibrating the proposed scatter search using the 720 instances of Naderi and Ruiz (2010) would result in an over fitting or over calibration. Calibrating methods on the same instances on which they are going to be tested later is bad practice and potentially unfair. Instead, we present a set of 50 random instances. In this set *n*, *m* and *F* are randomly chosen from the previous combinations. Once chosen, the processing times are randomly sampled from a uniform distribution in the range [1,99] as it is common in the scheduling literature. Therefore, the 50 calibration instances are different from the 720 test instances. These calibration instances are also available online.

We have used the Design of Experiments (DOEs) approach (Montgomery, 2012) for the calibration. The experimental configuration is a full factorial experiment with as many treatments as the previous combinations (48). *b*, *l* and *a* are controlled factors. The response variable is the relative percentage deviation over the best solution known for each instance, calculated as follows: $\text{RPD} = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$. $Some_{sol}$ is the solution obtained by any of the 48 SS configurations over a given instance and $Best_{sol}$ is the lowest makespan known for that instance. In order to increase the power of the experiment we used 5 replicates raising the total number of treatments to $48 \times 50 \times 5 = 12,000$. With such a large number of results the power of the experiment is expected to be high.

The results of the experiment are analyzed using the Analysis of Variance (ANOVA) technique. ANOVA is a parametric statistical tool and three hypotheses must checked. From more to less important these are independence of the residuals, homoscedasticity of the factor's levels (also known as homogeneity of variance) and normality of the residuals. After careful checking we found no significant deviations from the hypotheses. Note that a screening full factorial experimental design is by no means an elaborated and fine-tuned calibration process. Actually, a full factorial design analyzed by means of ANOVA can be considered as the first step in an algorithm calibration. For more exhaustive approaches, the reader is referred to Bartz-Beielstein, Chiarandini, Paquete, and Preuss (2012) where advanced techniques are shown. The reason behind our choice of a simple calibration is none other than to avoid an unfair comparison with existing approaches. After all, if a thorough and extensive fine tuning calibration was carried out over the proposed scatter search methods, we would not be able to ascertain in the computational evaluation if a better performance is obtained because of good algorithm constructs and operators or just because of a better calibration process.

For the computational experiments we have at our disposal a cluster with 30 computing blades, each one has two Intel XEON E5420 processors running at 2.5 Gigahertz. with 4 cores each (which makes 8 cores per blade). Each blade has 16 Gigabytes of RAM memory. Therefore, in total we have 240 cores and 480 Gigabytes. This cluster allows us to use many different virtual machines for the experiments, each one running Windows XP operating system with one single virtualized processor and 2 Gigabytes of RAM memory each. These virtual machines were used for the computational experiments to distribute the computational load.

It is important to set a meaningful stopping criterion for each scatter search configuration. A common error in the literature when calibrating algorithms is to give a fixed number of iterations to each combination of factors. Obviously, a larger *RefSet* needs substantially larger CPU times and one can finally conclude that a configuration with larger sets is better while the real truth is that it is better just because more CPU time was allowed. Therefore, we

use an elapsed CPU time termination criterion that is a function of the number of jobs n, number of machines m and number of factories F. This is needed in order to observe the statistical effect of the tested factors. If a fixed CPU time was used, smaller instances would end up with very good results as a relatively large CPU time would have been employed. The effect would be the contrary for large instances where the same CPU time would probably be not enough. This scenario would be disastrous as a lurking variable "CPU time" would mask the effect of the factors. As a result, we employ the following expression as a CPU time termination criterion for each run of the proposed scatter search configurations: $n \times m \times F \times C$, where C is set to 10 and the whole expression is in milliseconds. This is a moderately short CPU time as for the largest instances of 500 jobs, 20 machines and 7 factories the total elapsed CPU time will be 700 seconds and just of 2 seconds for the smallest instances of $20 \times 5 \times 2$.

The results of the ANOVA are summarized as follows (the ANOVA table is not reproduced here due to space constraints but it is available upon request from the authors). All three factors b, l and a are statistically significant with high F-Ratios and p-values very close to zero. Therefore, there are statistically significant differences in the response variable between the levels of the studied factors. In more detail, the most significant factor is the size b of set H. The second most significant factor is the size l of set H and the third the number of iterations after which restart is applied (a). The means plots of these factors, along with 95% Tukey's Honest Significant Difference (HSD) confidence intervals are shown in Fig. 3. It has to be noted that overlapping confidence intervals signify that observed differences in the response variable (RPD) of the overlapped means are statistically not significant.

The 2 level interactions between the factors are not significant. From the plots we see that the levels 10 and 15 are statistically equivalent for factor b. The same applies to levels 5 and 10 for factor l. We choose the values 10 and 10, respectively as together they make 20, an ideal size for *RefSet* according to Martí et al. (2006). For the number of iterations before restart, the level of 40 is equivalent to 20 and 30 but it results in a lower average. Although not shown here, values larger than 10 for l and larger than 40 for a resulted in worse performance in confirmation experiments. As a result we fix b and l to 10 and a to 40.

4.2. Methods compared and experimental setting

We now detail the experimental setting for the computational campaign. The following methods are included in the comparison:

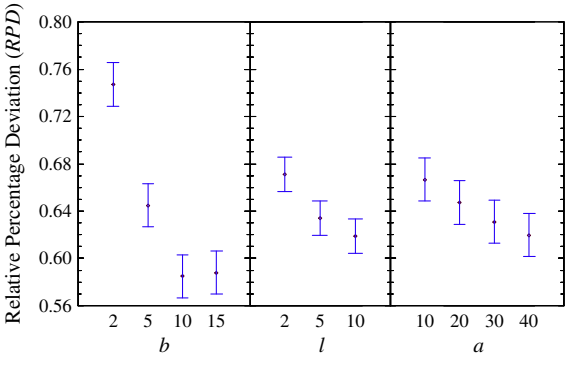

Fig. 3. Means plot for the size b of set H, size l of set S and number of iterations before restart a for the SS ANOVA calibration experiment. All means have Tukey's Honest Significant Difference (HSD) intervals at the 95% confidence level.

- The discrete electromagnetism metaheuristic of Liu and Gao (2010), referred to as EM. This algorithm includes four neighborhoods in a VNS local search phase. Random initialization as per the authors' design.
- Hybrid Genetic Algorithm with local search (GA_LS) of Gao and Chen (2011a), referred to as HGA. This algorithm employs NEH2 and VND(a) as initialization.
- The improved NEH of Gao and Chen (2011b), referred to as NEHdf. Note that in the original paper, the details of the employed branch and bound procedure used inside NEHdf are not given. We contacted the authors for help and source codes. Source codes were not given to us. Instead, the original authors provided us with a slightly extended paper version (Gao & Chen, 2011c). However, this paper did not contain sufficient explanations either. In the end, since the branch and bound enumerates all possible factory assignments and the maximum value of F is 7 in the benchmark, we found out that it was actually faster, using all possible accelerations, to try all 7! possible solutions at each step of the NEHdf. With this we got comparable, if not faster, CPU times that those reported in Gao and Chen (2011b).
- The improved VND of Gao et al. (2012a), referred to as VNS(B&B). Note that this algorithm is basically a mixture of VND(a) and the previous NEHdf and we faced the same reimplementation issues.
- The tabu search method of Gao et al. (2013), referred to as TS.
- The best iterated greedy algorithm of Lin et al. (2013) which the authors called IG_{VST} and is simply referred to as IG here.
- The comparison also includes the original methods presented in Naderi and Ruiz (2010), namely NEH1, NEH2, VND(a) and VND(b). Note that VND(a) and VND(b) have been slightly modified so to stop at a given specified CPU time and not after local optimality is reached. It has to be stressed though that neither method has any diversification mechanism so they eventually get stuck at a local optima from which they cannot escape. In any case, this change in the stopping criterion has been introduced in order to ease the comparisons among methods.
- We finally include in the comparison the proposed scatter search method SS.

In total we are comparing 11 methods. As we can see from the previous list and from the literature review of Section 2, only two algorithms have not been included in the computational comparison. We did not reimplement and test the GA_KB of Gao et al. (2012b) as according to the authors the performance is very similar to that of the HGA of Gao and Chen (2011a). Also, the paper is scant in details and an independent reimplementation of GA_KB is unlikely to succeed without access to the source codes. As commented in Section 2, the paper Wang et al. (2013) was published after all experimentation in this paper was finished. In any case, and as it was mentioned, the EDA method proposed in that paper is not competitive, being somewhat better than VND(a) but needing much more CPU time. It is clear that this method is much worse than other recent methods like the IG or TS above and therefore we have chosen not to reimplement it. We will provide, however, indirect comparisons against EDA later in this section.

Note that all methods have been carefully coded in C++ following the original author's explanations in their respective papers. The stopping criterion of all methods has been modified so that all algorithms will be using the same CPU time in all experiments. This CPU time follows the same expression as in the calibration of the proposed scatter search method ($n \times m \times F \times C$). However, in this case, C has been tested at several values, namely 20, 40, 60, 80 and 100. This means that the CPU time employed by all methods ranges from 4 seconds for the smallest instances of $n = 20$, $m = 5$, $F = 2$ and the shortest tested time of $C = 20$–7000 seconds for the largest instances of $500 \times 20 \times 7$ and

$C = 100$. Note that we do not test each method for $C = 100$ and record the times at 20, 40, 60 and 80. In each test, each algorithm is restarted from the beginning. This helps in avoiding self correlation in the results which would be problematic for later statistical testing. Testing all methods with 720 instances and with so many stopping times that range from a few seconds to almost 2 hours guarantees a full range of results and a sound statistical analysis. Furthermore, since all algorithms have been coded in the same language and are run on the same computers with the same CPU time stopping criterion we have a completely comparable computational campaign. Note that the makespan evaluation, most local search operators and initialization procedures are shared among the methods. If a given method works better than another it can only be attributed to the method itself and not to a faster computer, better coding or different stopping times.

In total we have tested 11 methods. NEH1, NEH2 and NEHdf are heuristics and do not have a stopping criterion and therefore are only tested once with each instance. All other 8 methods are tested with the 720 instances and with the 5 aforementioned different stopping times which means that we have $3 \times 720 = 2160$ results for the heuristics and $8 \times 5 \times 720 = 28,800$ results for the metaheuristics. Given the large number of results we have not used replicates. The total CPU time needed for the metaheuristic results (not considering the calibration of the scatter search or the heuristics) is almost 165 days. The same cluster of computers used for the SS calibration is employed for the comparisons.

4.3. Heuristic results for large instances

Table 1 shows the average relative percentage deviation for the three tested heuristic methods, grouped by the number of factories. Each cell contains the average of the 120 instances per value of F. The CPU times (in seconds) are also provided.

NEH1 is inferior to NEH2 which confirms the previous results of Naderi and Ruiz (2010). At the same time, it has to be considered that, on average, NEH1 is almost 4 times faster. NEHdf is only slightly better than NEH2 and also about 50% slower. All three heuristics are in any case incredibly fast, needing in the worst case less than 0.3 seconds (NEHdf for instances of size $500 \times 20 \times 7$). Recall that Gao and Chen (2011b) showed NEHdf not to be statistically better than NEH2. Let us check if this is the case. We carry out a multifactor ANOVA where n, m, F and the heuristics are controlled factors and the average relative percentage deviation is the response variable. We are only interested in the means plot of the factor algorithm, which is given in Fig. 4.

Note that the means plotted are actually the average relative percentage deviations for all 720 large instances. As we can see, we confirm the previous results of Gao and Chen (2011b) and conclude that while NEHdf obtains slightly better results than NEH2, these differences are not large and/or consistent enough so to be statistically significant. As a conclusion, NEH2 is a preferred method given also that NEHdf is difficult to reimplement and slower than NEH2.

Table 1
Average Relative Percentage Deviation (AVRPD) and CPU time needed (in seconds) for the three tested heuristics grouped by number of factories F.

	AVRPD			CPU time (seconds)		
F	NEH1	NEH2	NEHdf	NEH1	NEH2	NEHdf
2	6.35	4.58	4.29	0.007	0.015	0.029
3	7.17	4.82	4.45	0.006	0.018	0.030
4	8.21	5.00	4.51	0.005	0.022	0.032
5	8.51	5.03	5.01	0.005	0.026	0.035
6	9.32	5.40	5.27	0.005	0.028	0.038
7	10.28	6.04	5.91	0.004	0.032	0.049
Average	8.31	5.15	4.91	0.006	0.023	0.035

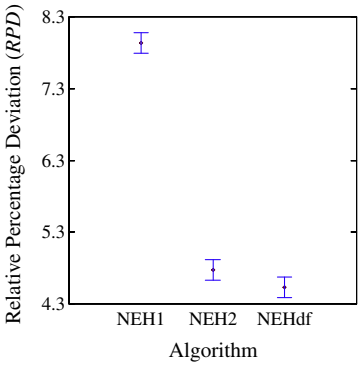

Fig. 4. Means plot for the heuristic algorithms. All means have Tukey's Honest Significant Difference (HSD) intervals at the 95% confidence level.

4.4. Metaheuristic results for large instances

Table 2 summarizes the results of the 8 tested metaheuristics. The results are grouped by each CPU time stopping criterion level (C) as well as per number of factories F. Again, each cell contains the average of 120 results. Even though the stopping time is a fixed equation for each instance and method ($n \times m \times F \times C$), the last column gives the average CPU time (in seconds) as a guidance.

The results of the computational evaluation contain some important findings. First of all, we confirm the better performance of VND(a) versus VND(b) as was explained in Naderi and Ruiz (2010). However, being just local search methods that stop at a local optima and without any diversification method, VND(a) and VND(b) do not improve their performance with additional CPU time. Both methods get stuck way before the shortest CPU times of $C = 20$ are reached. Actually, and as shown in Naderi and Ruiz (2010), both methods find their solutions in 0.147 and 0.096 seconds, on average, respectively. Other methods also get stuck as their solutions do not improve with additional times. An example is the VNS(B&B) of Gao et al. (2012a). In any case, the average deviation at 2.68% is clearly below VND(a) and VND(b), which confirms the results reached by the original authors.

An important finding resulting from the evaluation is that the EM method of Liu and Gao (2010), apart from being stuck as no better solutions are found with additional CPU time, is that its performance is below all other tested metaheuristics at 5.21% relative percentage deviation. As a matter of fact, this deviation is larger than that of NEHdf and NEH2 from Table 2. If we take the average CPU times in the shortest experiment of $C = 20$, EM needs 164.63 seconds on average, while NEHdf and NEH2 need just 0.035 and 0.023 seconds, respectively. This means that EM obtains a similar performance but at the same time needs an exorbitant amount of CPU time that is between 4704 and 7157 times longer. Note that in their paper, Liu and Gao (2010) claimed to have improved 151 best known solutions out of the 720 of Naderi and Ruiz (2010). We would like to stress, that these comparisons are often misleading. We do not claim that their results did not improve the best known solutions. As a matter of fact, our reimplementation of EM improves not 151 but 161 best known solutions when compared to the original best solutions given in Naderi and Ruiz (2010) (so it seems that our implementation of EM is actually slightly more efficient). The fact is that there are another 559 instances in which EM does not improve the best known solutions. Herein lies the problem, as the solutions given by EM in these 559 cases are not good. The result is that even though

Table 2
Average Relative Percentage Deviation (AVRPD) and CPU time used (in seconds) for the tested metaheuristics grouped by CPU time limit C and number of factories F. Bold values represent best results.

C	F	EM	HGA	IG	SS	TS	VND(a)	VND(b)	VNS(B&B)	CPU time
20	2	4.33	2.72	2.51	**0.98**	1.66	2.77	3.02	2.56	73.17
	3	4.92	3.09	2.69	**1.02**	2.08	3.02	3.43	2.44	109.75
	4	4.90	3.34	2.74	**1.18**	2.82	3.27	3.27	2.44	146.33
	5	5.15	3.62	2.66	**1.50**	3.48	3.72	3.72	2.47	182.92
	6	5.42	3.95	2.50	**1.87**	4.05	4.08	4.08	2.77	219.50
	7	6.18	4.76	2.49	**2.44**	4.98	4.92	4.77	3.41	256.08
Average		5.15	3.58	2.60	**1.50**	3.18	3.63	3.78	2.68	164.63
40	2	4.68	2.63	2.37	**0.88**	1.54	2.77	3.02	2.57	146.33
	3	4.86	3.00	2.57	**0.96**	1.99	3.02	3.43	2.56	219.50
	4	5.00	3.32	2.45	**1.06**	2.76	3.27	3.56	2.45	292.67
	5	5.19	3.56	2.40	**1.29**	3.46	3.72	3.79	2.40	365.83
	6	5.46	3.87	2.30	**1.75**	3.96	4.08	4.08	2.73	439.00
	7	6.16	4.74	2.25	**2.25**	4.97	4.92	4.77	3.32	512.17
Average		5.23	3.52	2.39	**1.36**	3.11	3.63	3.78	2.67	329.25
60	2	4.73	2.56	2.27	**0.80**	1.46	2.77	3.02	2.62	219.50
	3	4.87	2.99	2.41	**0.81**	1.93	3.02	3.43	2.49	329.25
	4	5.00	3.26	2.46	**0.95**	2.70	3.27	3.56	2.50	439.00
	5	5.40	3.53	2.34	**1.25**	3.42	3.72	3.79	2.43	548.75
	6	5.54	3.85	2.25	**1.66**	3.84	4.08	4.08	2.66	658.50
	7	6.03	4.72	**2.07**	2.16	4.88	4.92	4.77	3.38	768.25
Average		5.26	3.49	2.30	**1.27**	3.04	3.63	3.78	2.68	493.88
80	2	4.52	2.56	2.21	**0.76**	1.43	2.77	3.02	2.63	292.67
	3	4.98	2.99	2.42	**0.75**	1.92	3.02	3.43	2.52	439.00
	4	4.93	3.22	2.43	**0.93**	2.66	3.27	3.56	2.41	585.33
	5	5.18	3.52	2.32	**1.16**	3.44	3.72	3.79	2.45	731.67
	6	5.58	3.83	2.11	**1.64**	3.94	4.08	4.08	2.78	878.00
	7	5.81	4.67	**1.96**	2.20	4.93	4.92	4.77	3.30	1024.33
Average		5.17	3.47	2.24	**1.24**	3.05	3.63	3.78	2.68	658.50
100	2	4.65	2.49	2.10	**0.70**	1.42	2.77	3.02	2.65	365.83
	3	4.84	2.90	2.34	**0.69**	1.96	3.02	3.43	2.50	548.75
	4	5.11	3.18	2.34	**0.90**	2.62	3.27	3.56	2.46	731.67
	5	5.32	3.52	2.22	**1.14**	3.28	3.72	3.79	2.41	914.58
	6	5.48	3.80	1.94	**1.57**	3.89	4.08	4.08	2.69	1097.50
	7	6.09	4.67	**1.99**	2.14	4.92	4.92	4.77	3.31	1280.42
Average		5.25	3.43	2.16	**1.19**	3.02	3.63	3.78	2.67	823.13
Tot. average		5.21	3.50	2.34	**1.31**	3.08	3.63	3.78	2.68	493.88

Liu and Gao (2010) improved 151 of the original best known solutions of Naderi and Ruiz (2010), their average performance, when compared in an apples to apples scenario, is poor. Summing up, improving a fraction of the best known solutions is not indicative of good performance. Had Liu and Gao (2010) compared their EM against VND(a) of Naderi and Ruiz (2010) they would have found out that VND(a) is about 43% better in performance and about 1120 times faster according to the results in this paper and in Naderi and Ruiz (2010).

The HGA method of Gao and Chen (2011a), as commented in Section 2 was shown to outperform VND(a) but at an unfair CPU time advantage. In the experiments in this paper the same CPU times are employed and we confirm that indeed HGA results in a slightly better average relative percentage deviation than VND(a) (3.50% versus 3.63%). It remains to be seen, however, if this small difference in performance is indeed statistically significant. At the end of this section we will carry out additional statistical analyses that will confirm this question. The TS of Gao et al. (2013) is confirmed to outperform VND(a) and HGA, which ratifies the results of the original paper. It is, however, quite interesting that our implementation of the VNS(B&B) of Gao et al. (2012a) seems to be much better than both HGA and TS albeit VNS(B&B) is not mentioned or used in the comparisons of this last paper of Gao et al. (2013).

We can also comment on the recent IG method of Lin et al. (2013). We can see that the IG beats all other existing metaheuristics clearly, bringing down the relative percentage deviation to just 2.34%. IG are simple methods and therefore we can safely recommend IG over EM, HGA, TS and VNS(B&B). VND(a) and VND(b) are actually much faster and therefore should be considered separately. Additionally, and similarly to HGA and TS, the results of IG steadily improve as more CPU time is given. For example, IG with $C = 20$ has an average deviation of 2.60% compared to a deviation of 2.16% for $C = 100$.

We finally comment on the results of the proposed scatter search method SS. We can see that the overall relative percentage deviation is just 1.31% which is almost 79% better than the closest competitor IG. Except in some isolated cases with $F = 7$ and large CPU times where IG manages slightly better solutions, SS obtains, by far, the lowest deviations in all cases. From the reported results, all indications are that the proposed SS is a much better performer after being tested in a wide range of CPU times and instances. Furthermore, it has to be noted that the solutions given by the proposed SS method when tested with the shortest CPU time of $C = 20$ already improve 719 out of the 720 best known solutions reported in Naderi and Ruiz (2010). For the single instance where the solution is not improved, the reported makespan is just one unit larger. For $C = 40$ all 720 best known solutions are already improved. Comparatively, the recently proposed IG algorithm of Lin et al. (2013) was reported in that paper to have improved "almost half" of the instances of Naderi and Ruiz (2010). A similar indirect test can be carried out with the untested EDA algorithm of Wang et al. (2013). Recall that this algorithm was not reimplemented because it was just recently proposed. However, the authors report their best found solutions, so an indirect comparison is possible. In their paper the authors Claim 589 best new

solutions. In an appendix, they report just 585 best solutions. Comparing all these values against our new best known solutions obtained in this paper we conclude that their EDA method produces an average relative percentage deviation of 2.28% in these 585 instances that they improved. It has to be mentioned that this is a best case scenario, since we are only considering the 585 instances the authors improved in respect to the original best known values of Naderi and Ruiz (2010). Comparatively, the SS proposed in this paper, for the shortest tested CPU time of $C = 20$ results in a deviation of just 1.62% for the same 585 instances. Actually, for these 585 instances, the proposed SS ($C = 20$) is better than or equal to the EDA in 508 cases. Although the tests have been carried out on different computers (the computer used by Wang et al. (2013) being faster at 3.2 Gigahertz than ours which runs at 2.5 Gigahertz), our proposed SS with $C = 20$ has comparable CPU time demands to the EDA of Wang et al. (2013). As a result, it is safe to state that even indirectly compared, the proposed SS obtains much better solutions than the EDA of Wang et al. (2013). The new improved solutions obtained in this paper are available at http://soa.iti.es.

While the differences between the proposed SS and the existing metaheuristics reported in Table 2 are clearly large enough as to be statistically significant, we still carry out an ANOVA to check if the observed differences are indeed statistically significant. In a first summarized ANOVA we check the EM method against NEH2 and NEHdf. Due to reasons of space we do not report the means plots but we confirm our suspicions that the overall average relative percentage deviation of EM is not statistically better than that of NEH2 or NEHdf.

In order to have a clearer picture, NEH1, NEH2, NEHdf and EM are removed for subsequent statistical analyses. Another ANOVA is carried out where F and C, together with the type of algorithm, are controlled factors. By far, the most significant effect comes from the algorithm factor with an F-Ratio of almost 423 and a p-value very close to zero. The factor F (as well as n and m if we augmented the experiment) are very significant. Conversely, the factor C is not very significant. This is a result of many algorithms showing the same performance regardless of the CPU time employed. Fig. 5 shows the means plot of the factor algorithm averaged across all 720 instances and all C values (3600 data points averaged at the center of each interval).

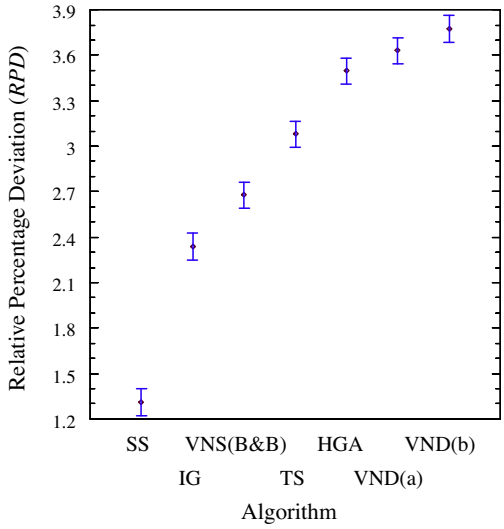

Fig. 5. Means plot for the metaheuristic algorithms (excluding EM). All means have Tukey's Honest Significant Difference (HSD) intervals at the 95% confidence level.

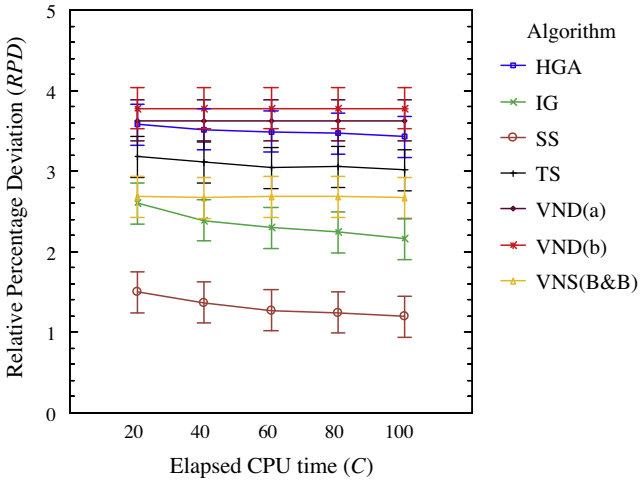

Fig. 6. Means plot for the interaction between the CPU time limit C and the metaheuristic algorithms (excluding EM). All means have Tukey's Honest Significant Difference (HSD) intervals at the 95% confidence level.

As we can see, the differences depicted in Table 2 are, for most methods, statistically significant. SS is statistically better than IG which in turn is better than VNS(B&B) which improves on TS which again is better than HGA. However, HGA is statistically equivalent to VND(a). This confirms the experiments of Gao and Chen (2011a), which showed that with similar CPU time, HGA was actually worse than VND(a). Our implementation of HGA shows slightly better results than VND(a) when run at the same CPU time but the difference is not statistically significant. Furthermore, recall that even though throughout this paper VND(a) and VND(b) are run for the same CPU time as all other methods, in reality they take a fraction of the time. As a result, HGA cannot be recommended over VND(a). Finally, VND(a) is not statistically better than VND(b), which again concurs with the experiments carried out in Naderi and Ruiz (2010).

It is also interesting to study the interaction between the CPU time limit C and algorithm, shown in Fig. 6.

We confirm that for many algorithms better solutions are not obtained with additional CPU time. SS is statistically better than IG for all time periods and it is seen that among all methods, the only ones that progressively improve as more CPU time is allowed are IG and SS although most of the time the differences are not large enough to be statistically significant. Note however that the width of the Tukey's Honest Significant Difference intervals shortens as the number of results increase. Had we run each algorithm more times (replicates) for each instance and C value, the differences for IG and SS would have resulted as significant as C increases.

As a final note, small focused experiments between IG and SS for $C = 60$, 80 and 100 and $F = 7$ (the cases in which, according to the reported averages in Table 2, IG is better than SS) show that the small differences are not enough to be statistically significant. As a rule of thumb, a difference between two averages has to be consistent over a large number of cases and/or large enough so as to be statistically significant. This fact, far from being undesirable in statistical testing, is the real backbone of the generalization capabilities of the ANOVA. A method A has to be substantially better than another method B and over a large number of test cases in order to generalize results to other cases and populations. Otherwise, a 0.1% better performance over a small number of lab cases in a method A over another method B would not guarantee that outside the lab these differences would hold true.

5. Conclusions and future research

The Distributed Permutation Flowshop Problem (DPFSP) is an interesting multi-factory extension of the regular flowshop recently proposed by Naderi and Ruiz (2010). The authors initially proposed six alternative Mixed Integer Linear Programming models as well as two simple heuristics (NEH1 and NEH2) based on the well known high performing flowshop heuristic of Nawaz et al. (1983) augmented with efficient job to factory assignment rules. The authors also presented two simple Variable Neighborhood Descent algorithms VND(a) and VND(b). After this initial work, a number of authors have proposed a number of methods and have compared mainly against the best performing method at the time – VND(a). In this follow up research we have studied again the DPFSP and have proposed an effective Scatter Search (SS) procedure. The main characteristic of the presented SS is a hybrid *RefSet* made up of full solutions as well as job to factory assignment vectors. The solution combination method combines all full solutions with all job to factory assignment vectors. This results in an effective strategy as the solution improvement procedure works in the job permutations at each factory and the combination method explores different effective job to factory assignments. Together with a stringent reference set update procedure and a restart mechanism, the proposed SS results in state-of-the-art performance.

We have carried out a thorough computational analysis where most existing methods from the literature have been carefully reimplemented and tested in a comprehensive set of 720 instances. Almost 165 days of CPU time have been employed in the tests where all algorithms have been tested at 5 different stopping times. The computational results are accompanied by sound statistical analysis using design of experiments and analysis of variance techniques. Results indicate that the proposed SS outperforms all existing methods by a wide statistical margin, including methods that have been proposed very recently. Another contribution of this paper is the comparison among the other existing methods. Many algorithms, when tested in a completely comparable scenario frequently show a performance that was not observed in the original experiments. For example, we have shown that the performance of the EM algorithm, despite claims from the original authors, is not competitive. On the other hand, our reimplementation of the VNS(B&B) method shows a promising performance even though the original authors have not compared this method in their latest published study.

The DPFSP is a recently proposed scheduling problem and many avenues for future research lay open before us. There is no reported research on the DPFSP with other objectives apart from makespan. Furthermore, the problem should be generalized, as not all factories are often completely identical. Other aspects could include leveling the load among the factories or considering important real-life constraints such as assembly operations and setup times.

Acknowledgements

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.

References

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., & Preuss, M. (Eds.). (2012). *Experimental methods for the analysis of optimization algorithms*. New York: Springer.

Chakraborty, U. (Ed.). (2009). *Computational intelligence in flow shop and job shop scheduling. Studies in computational intelligence*. New York: Springer.

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

Emmons, H., & Vairaktarakis, G. (2012). *Flow shop scheduling: theoretical results, algorithms, and applications. International series in operations research & management science*. New York: Springer.

Framinan, J. M., Gupta, J. N. D., & 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.

Gao, J., & Chen, R. (2011c). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. Technical Report SRE-10-1014. College of Information Science and Technology, Dalian Maritime University, Dalian, Liaoling Province, 116026, China.

Gao, J., & 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., & 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., & 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., & 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., & 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., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences, 8*(1), 156–166.

Glover, F. (1998). A template for scatter search and path relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, & D. Snyers (Eds.), *Artificial evolution. Third European conference AE '97 Nîmes, France, October 22–24, 1997 selected papers. Lecture notes in computer science* (vol. 1363, pp. 13–54). New York: Springer.

Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 29*(3), 652–684.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & 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., & Stafford, E. F. Jr, (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research, 169*(3), 699–711.

Hejazi, S. R., & 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.

Kahn, K. B. (Ed.). (2004). *The PDMA handbook of new product development*. New York: John Wiley & Sons.

Laguna, M., & Martí, R. (2003). *Scatter search: Methodology and implementations in C. Operations research/computer science interfaces series*. Dordrecht, Netherlands: Kluwer Academic Publishers.

Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering, 37*(1–2), 57–61.

Lin, S.-W., Ying, K.-C., & 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., & 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* (pp. 156–163). Hong Kong, China: IEEE Computer Society.

Martí, R., Laguna, M., & Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research, 169*(2), 359–372.

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

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

Montgomery, D. C. (2012). *Design and analysis of experiments* (8th ed.). Wiley.

Moon, C., Kim, J., & Hur, S. (2002). Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers & Industrial Engineering, 43*(1–2), 331–349.

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

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

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

Pinedo, M. (2012). *Scheduling: Theory, algorithms and systems* (4th ed.). New York: Springer.

Pochet, Y., & Wolsey, L. A. (2006). *Production planning by mixed integer programming. Springer series in operations research and financial engineering*. New York: Springer.

Quadt, D., & Kuhn, D. (2007). A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research, 178*(3), 686–698.
Reisman, A., Kumar, A., & 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., Leisten, R., & Framinan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research, 37*(8), 1439–1454.
Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494.
Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, The International Journal of Management Science, 34*(5), 461–476.
Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.
Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research, 205*(1), 1–18.
Saravanan, M., Noorul Haq, A., Vivekraj, A., & Prasad, T. (2008). International journal of advanced manufacturing technology. *International Journal of Advanced Manufacturing Technology, 37*(11–12), 1200–1208.
Sarin, S. C., & Jaiprakash, P. (2007). *Flow shop lot streaming problems*. New York: Springer.
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.
Vallada, E., & Ruiz, R. (2009). Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research, 193*(2), 365–376.
Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA, The International Journal of Management Science, 38*(1–2), 57–67.
Vignier, A., Billaut, J.-C., & Proust, C. (1999). Les problèmes d'ordonnancement de type flow-shop hybride: État de l'art. *RAIRO Recherche opérationnelle, 33*(2), 117–183. in French.
Wang, B. (Ed.). (1997). *Integrated product, process and enterprise design. Manufacturing systems engineering series*. London: Chapman & Hall.
Wang, H. (2005). Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. *Expert Systems, 22*(2), 78–85.
Wang, S.-Y., Wang, L., Liu, M., & 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.