A Decomposition-based Local Search Algorithm for Multi-objective Sequence Dependent Setup Times Permutation Flowshop Scheduling

Murilo Zangari*, Ademir Aparecido Constantino* and Josu Ceberio†
*Computer Science Department, State University of Maringá, 87020-900, Maringá, Brazil
Email: {murilo.zangari, ademir.uem}@gmail.com
†Intelligent Systems Group, Department of Languages and Computer Systems,
University of the Basque Country (UPV/EHU), 20018 Donostia, Spain
Email: josu.ceberio@ehu.es

***Abstract*—The flowshop scheduling problem (FSP) has been widely studied in the last decades, both in the single objective as well as in the multi-objective scenario. Besides, due to the real-world considerations on scheduling problems, the concern regarding sequence-dependent setup times has emerged. In this paper, we present a decomposition-based iterated local search algorithm (MOLS/D) to deal with the multi-objective sequence-dependent setup times permutation FSP. In order to demonstrate the validity of the proposed algorithm, we have conducted an experimental study on a set of 220 benchmark instances minimizing the criteria makespan and total weighted tardiness. The results, according to various performance metrics and statistical analysis, show that MOLS/D significantly outperforms a tailored MOEA/D variant and the best-known reference sets from the literature. Thus, we have established a state-of-the-art approach for the problem considered.**

I. Introduction

In the last decades, the flowshop scheduling problem (FSP) has been widely studied since it serves as a model for several real-world industry environments from manufacturing and engineering [1]. FSP is characterized by a set of jobs that have to be processed by a set of machines sequentially. Each job has a fixed non-negative processing time at each machine, which is known in advance. A machine can only process one job at the time, and the order of the machines is the same for all jobs.

For FSP, various objectives have been considered as optimization goals. The most studied criteria in this sense are those based on the completion time, such as the minimization of the maximum completion time of the last job (makespan), and the minimization of the sum of the completion time of all jobs (total flowtime). Objectives that involve due dates, such as the total (weighted) tardiness, have also received increasing attention [1], [2]. Each one of these criteria is *NP*-hard from a specific number of machines [3]. As a consequence, exact methods have not proved to be suitable for solving large problem instances due to the required computational effort. Hence, optimization (heuristic and meta-heuristic) algorithms have attracted a substantial research effort for solving FSP, and have proved to be particularly effective [1], [4], [5].

Most literature referring to FSP focuses on the optimization of a single objective. However, most real-life scheduling problems naturally involve multiple criteria. For instance, the optimization of the makespan deals with a very high throughput and machine utilization. Although, most of the due dates related to the satisfaction of the costumers are likely to be violated [6]. In this case, the optimization of a combination of the objectives is often of great interest for decision makers. Over the years, several multi-objective meta-heuristic algorithms (such as those based on simulated annealing [7], iterative local search [6], and evolutionary computation [8], [9]) have been developed for solving multi-objective FSP. Most of them are classified as *a posteriori* approach, in which the goal is to find a set of solutions representing a *trade-off* among the objectives. A literature review on multi-objective FSP is found in [1].

Setup times are very common in most industrial environments. In FSP, these setups can reflect some non-productive operations that have to be processed on machines that are not part of the processing times of the jobs, such as cleaning, replacement of tools, transportation of jobs from one machine to the next machine, fixing, and so on [10]. Most of the studies on FSP just ignore the setup times or consider them as *sequence-independent* and as part of the processing times. However, it leads to a very theoretical formulation of the problem [6], [11]. Alternatively, when the setup times are considered, and they depend both on the current job being processed and on the next job in the sequence, the problem is called *sequence-dependent setup times* (SDST).

Even though SDST and multi-criteria are important in industry, few studies have focused *on multi-objective sequence-dependent setup times permutation flowshop scheduling* [6], [11]. In [6], the authors were the first to attempt to solve it. They approached the problem with a Pareto-based metaheuristic algorithm, called Restarted Iterated Pareto Greedy (RIPG). The algorithm is characterized by a heuristic initialization of the population and a tailored local search. They focused on two bi-objective cases, i) makespan and total flowtime, and ii) makespan and total weighted tardiness. The authors also presented a set of 220 benchmark instances. In that paper,

RIPG achieved the best results in comparison to several multi-objective metaheuristic algorithms. In [12], the authors attempt to solve SDST minimizing makespan and total flow time with a two-step local search algorithm. The first step is a Pareto local search embedded with a perturbation operator. In the second step, the algorithm deals with several scalar single-objective subproblems. The authors reported significant results compared to some state-of-the-art algorithms. In [11], the authors proposed a Pareto-based iterated local search algorithm, called MOILS, for minimizing makespan and total weighted tardiness. The algorithm achieved competitive results compared to some other approaches, such as the multi-objective simulated annealing (MOSA) [7].

Since this topic is an attractive research area and is still at an early stage, in this paper, we propose a decomposition-based iterated local search algorithm, called MOLS/D, to solve SDST flowshop scheduling minimizing makespan and total weighted tardiness. The algorithm consists of the following: (i) A population of solutions, in which each solution corresponds to a scalar single-objective subproblem. The subproblems are optimized simultaneously in a collaborative manner using the concept of neighborhood between them. (ii) A heuristic step based on the NEH algorithm [4] to initialize the population on promising regions of the search space. (iii) A simple local search operator to search new solutions. (iv) Lastly, a controlled shaking procedure is used when the algorithm gets stuck on local optimum regions of the search space.

The idea of combining these ingredients is to have an efficient management of the search process regarding the balance between exploration and exploitation. The decomposition approach has been recognized as one of the main strategies in traditional multi-objective optimization [13]. Constructive heuristic methods, such as the NEH algorithm [4] with different dispatching rules, are usually employed to initialize promising solutions [2], [7]. Advanced local search methods are embedded with exploratory mechanisms to escape from local optima, such as short memory (tabu search) [14], acceptance criterion (simulated annealing) [7], restart mechanisms (iterated greedy methods) [6], and so on. Moreover, several decomposition-based algorithms, mainly those implemented using the concepts of the multi-objective evolutionary algorithm based on decomposition (MOEA/D framework) [15], have been designed to solve FSP optimizing various combination of objectives [8], [14], [16]. In fact, the decomposition approach provides a simple, yet efficient, framework for using single-objective approaches, such as genetic search and local search.

In order to demonstrate the validity of the proposed MOLS/D, we conducted an experimental study on 220 SDST benchmark instances. We have compared MOLS/D against different approaches from the specific literature. The first algorithm to be compared is the improved MOEA/D variant [8]. This algorithm employs genetic search, specially tailored for PFSP. For a fair comparison, we also incorporate the heuristic initialization and the shaking procedure into MOEA/D. The remaining algorithms to be compared are RIPG and MOSA_VM, which were reported in [6] as the two best-performer approaches for the benchmark considered in this paper. The results, according to different performance assessments and statistical tests, shows that the proposed MOLS/D significantly outperforms the other approaches..

The rest of the paper is organized as follows: Section II introduces the problem and the preliminary concepts. Section III describes the proposed algorithm. Section IV details the experimental study. The final remarks are given in Section V.

II. Preliminaries

A. Sequence-dependent setup times flowshop scheduling

In this problem, there is a set of n jobs to be processed on a given set of m machines. Each job $i \in \{1, ..., n\}$ has a fixed processing time on machine $k \in \{1, ..., m\}$, denoted as $p_{i,k}$. The problem has the following assumptions: i) it assumes that all jobs are available at time zero, ii) each job can only be processed on a single machine at the same time, iii) each machine can process only one job at a time, and iv) we are restricted to a version of the problem where the sequence of jobs is the same for all machines. In this case, a candidate solution is represented as a permutation of jobs, and the number of possible solutions is $n!$ (all permutations of size n).

If the setup times depend on the current job being processed and the next job waiting, the problem is called *sequence-dependent setup times*. Setups can be anticipatory, i.e., they can be performed as soon as the machine is free and before the next job in the sequence is loaded. In this case, $S_{i,j,k}$ denotes the setup time between the jobs i and j ($i \neq j$) on machine k when processing job j after having processed job i.

A permutation is represented as $\sigma = (\sigma(1), ..., \sigma(n))$, where $\sigma(i)$ represents the job to be processed in the ith position of the solution σ. Hence, the completion time of job $\sigma(i)$ on machine k is denoted as $C_{\sigma(i),k}$, and it is recursively calculated as

$$C_{\sigma(i),k} = \max\{C_{\sigma(i),k-1}, C_{\sigma(i-1),k} + S_{\sigma(i-1),\sigma(i),k}\} + p_{\sigma(i),k} \quad (1)$$

where $C_{\sigma(0),k} = 0$, $C_{\sigma(i),0} = 0$, and $S_{\sigma(0),\sigma(i),k} = 0$.

The completion time of the last job on the last machine is called maximum completion time or *makespan*, and it is denoted as $C_{max} = C_{\sigma(n),m}$. The minimization of makespan corresponds to an overall higher throughput [17].

Jobs often represent client orders that have a specific delivery date, represented as d_i. If a job i is finished after its due date d_i, i.e., $C_{i,m} > d_i$, then it is said to be tardy. The tardiness of a job i is calculated as $T_{\sigma(i)} = \max\{0, C_{\sigma(i),m} - d_{\sigma(i)}\}$. Moreover, not all deliveries are equally important. Thus, a weight w_i is also given for each job i. The total weighted tardiness of all jobs for a given permutation is calculated as $TWT = \sum_{i=1}^{n} w_{\sigma(i)}.T_{\sigma(i)}$.

B. Multi-objective optimization

In terms of minimization, a multi-objective optimization problem with q objectives can be stated as

$$\text{Minimize } F(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_q(\mathbf{x})] \quad \text{subject to } \mathbf{x} \in \Omega \quad (2)$$

where $f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_q(\mathbf{x})$ are q objective functions to be minimized, $\mathbf{x}$ is a decision vector, and Ω is the set of feasible solutions (decision space).

The concept of Pareto dominance [18] is used to represent a set of solutions that compromises a *trade-off* among all the objectives. Let $\mathbf{x}, \mathbf{y} \in \Omega$, $\mathbf{x}$ is said to *dominate* $\mathbf{y}$ if and only if $f_l(\mathbf{x}) \leq f_l(\mathbf{y})$ for all $l \in \{1, .., q\}$ and $f_l(\mathbf{x}) < f_l(\mathbf{y})$ for at least one l. A solution $\mathbf{x}^* \in \Omega$ is called *Pareto optimal* if there is no other $\mathbf{x} \in \Omega$ which *dominates* $\mathbf{x}^*$. The set of all the *Pareto* optimal solutions is called the *Pareto set (PS)* and the solutions mapped in the *objective space* are called *Pareto front (PF)*, i.e., $PF = \{F(\mathbf{x}) | \mathbf{x} \in PS\}$.

When a *PF* is given, the decision maker picks one satisfactory solution from it. Although, in several hard problems, achieving the exact *PF* might be computationally intractable. However, a good approximation to the *PF*, in a reasonable time, is of considerable interest.

C. Iterated local search

A solution $\mathbf{x}'$ is a neighbour solution of $\mathbf{x}$ (i.e., $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$) if $\mathbf{x}'$ can be achieved by a single *move* from $\mathbf{x}$, and it depends on a basic underlying *operator* and a given *distance* between any two solutions. Several studies have shown that a simple local search procedure is very helpful in improving the quality of solutions [6], [11].

Iterated local search (ILS) is a simple, yet powerful, metaheuristic which consists of repeatedly applying local search procedures. When the search is trapped in a local optimal solution, ILS can perturb the solution to allow the search to escape from the trap without losing many of the good properties of the current solution [19].

III. Multi-objective Iterated Local Search based on Decomposition

In algorithms based on the traditional MOEA/D framework [15], a set of uniformly spread weight vectors $\{\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^N\}$ is employed, where a weight vector contains weights as the number of objectives $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_q)$. A weight is subject to $\sum_{l=1}^{q} \lambda_l = 1$, and $\lambda_l \geq 0$.

The idea is to decompose a multi-objective problem into a set of N single-objective subproblems, where each subproblem is associated to a weight vector $\boldsymbol{\lambda}^k$, $k = 1, ..., N$, and it is defined by a given scalarizing function method $g(\sigma|\boldsymbol{\lambda}^k)$.

According to a preliminary study [16], the best-suited scalarizing function for bi-objective FSP problems is the *Weighted Sum* approach. Since the objectives C_{max} and *TWT* have different magnitudes, in this paper, we use the normalized *Weighted Sum* approach, and it is calculated as

$$\textit{minimize} \quad g^{ws}(\sigma|\boldsymbol{\lambda}^k, \mathbf{z}^*, \mathbf{w}^*) = \sum_{l=1}^{q} \lambda_l^k \cdot \frac{f_l(\sigma) - \alpha.z_l^*}{w_l^* - z_l^*} \quad (3)$$
$$\sigma \in \mathbb{S}_n$$

where $\mathbb{S}_n$ denotes the search space of solutions composed by $n!$ permutations, $\mathbf{z}^* = (z_1^*, .., z_q^*)$ is the minimum (best) values found so far for each objective, and $\mathbf{w}^* = (w_1^*, .., w_q^*)$ is the maximum (worst) values found so far for each objective. Besides, $\mathbf{z}^*$ does not guarantee a lowest reference point. So $\mathbf{z}^*$ is multiplied (decreased) by a factor α. In accordance with [8], [16], we set $\alpha = 0.6$. For simplicity, we represent $g^{ws}(\sigma|\boldsymbol{\lambda}^k, \mathbf{z}^*, \mathbf{w}^*)$ as $g(\sigma|\boldsymbol{\lambda}^k)$.

During the search process, the algorithm maintains a population $Pop = \{\sigma^1, ..., \sigma^N\}$, and the correspondents function values $F(\sigma^1), ..., F(\sigma^N)$, where σ^k is the best solution found so far for subproblem $g(\sigma|\boldsymbol{\lambda}^k)$. The algorithm also maintains a set, called *external Pareto* (EP), to store all non-dominated solutions found so far according to the Pareto dominance. In Algorithm 1, we present the MOLS/D pseudo-code, and afterward, we describe its steps.

Algorithm 1 MOLS/D for multi-objective flowshop scheduling
Input:
N: population size.
$\{\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^N\}$: N distributed weight vectors.
T: neighborhood size.
n_r: maximum replacements allowed by a new solution.
n_{sh}: number of random *insert moves*.
Output:
Pop: set of the N current solutions;
EP: external set of non-dominated solutions;
/ Initialization */*
1: For every subproblem k, determine the T closest weight vectors $B(k)$;
2: Initialize $EP = \emptyset$;
3: Initialize $Pop := \{\sigma^1, ..., \sigma^N\}$ according to the heuristic method, and compute every $F(\sigma^k)$;
4: Initialize the reference points $\mathbf{z}^*, \mathbf{w}^*$;
/ Search process */*
5: **while** a termination condition is not met **do**
6: **for** each subproblem index $k \in 1, ..., N$ **do**
7: Obtain σ' from $\mathcal{N}(\sigma^k)$ based on a *single random move*, and compute $F(\sigma')$;
8: Update the reference points $\mathbf{z}^*, \mathbf{w}^*$;
9: Update Pop with σ' according to $B(k)$ and n_r;
10: **if** σ^k has not been improved after n iterations **then**
11: Apply a perturbation on σ^k with n_{sh} *insert moves*;
12: **end if**
13: Update EP with σ^k
14: **end for**
15: **end while**
16: **return** Pop, EP

Initialization: The number of subproblems (N) and the correspond weight vectors are defined as in [15], in which each individual λ_l^k takes a value from $\{\frac{0}{H}, \frac{1}{H}, ..., \frac{H}{H}\}$, where H is a user-specified parameter, and $N = C_{H+q-1}^{q-1}$. Next, for each subproblem index k, the indexes of the T closest weight vectors $B(k)$ are defined according to the Euclidean distance between them. The *EP* is initialized empty. Pop is initialized using a heuristic method in which two promising solutions are generated (one for each specific criterion). The initialization of Pop works as follows:

1) First, we consider the minimization of C_{max}. In this case,

we generate a permutation π' using the NEH heuristic with the improved priority rule proposed by [20].
2) Next, we regard the minimization of *TWT*. We employ the NEHedd heuristic [2] to generate π'', in which different priority rules are evaluated regarding the influence of the due dates of the jobs in each specific instance.
3) Then, we employ the three schemes presented in [7] to improve π' and π''.
4) The subproblem associated to the weight vector $\boldsymbol{\lambda} = (1, 0)$ (respectively $\boldsymbol{\lambda} = (0, 1)$) is initialized with π' (respectively π''). The remaining subproblems are initialized with perturbed solutions based on n_{sh} random *insert moves* on π' and π''.

Search process: The search process has tree steps: i) generation, ii) update, and iii) perturbation. At each iteration, for $k \in 1, ..., N$:

1) A single neighbourhood operator (e.g., *1-insert* or *1-interchange*) is applied to σ^k to generate σ', and its $F(\sigma')$ is computed.
2) Next, $F(\sigma')$ is used to update the reference points. Then, the algorithm tries to update the current Pop with σ' according to the closest neighbor subproblems in $B(k)$. For each index $j \in B(k)$, if $g(\sigma'|\boldsymbol{\lambda}^j) \leqslant g(\sigma^j|\boldsymbol{\lambda}^j)$, then set $\sigma^j = \sigma'$ and $F(\sigma^j) = F(\sigma')$. To avoid many copies in the population, σ' can only update a maximum of n_r subproblems.
3) If a subproblem solution σ^k has not been updated after n consecutive iterations, then σ^k is shaken with n_{sh} random *insert moves*, even if it computes a worse $g(\sigma^k|\boldsymbol{\lambda}^k)$.

The *EP* is updated with the non-dominated solutions from Pop according to the Pareto-dominance, i.e., if no solution in EP dominates $F(\sigma^k)$, $F(\sigma^k)$ is added to EP, and all the solutions dominated by $F(\sigma^k)$ are removed.

We have some comments about the design of the algorithm: i) As in the traditional MOEA/D framework, all the subproblems receive the same computational effort. We know that the shape of the *PF* for SDST flowshop minimizing (C_{max},*TWT*) is slightly continuous and convex. Thus, if we use a well-distributed set of weight vectors that covers the *PF*, the algorithm can efficiently manage the spread of the solutions along the *PF*. ii) The maximum number of updates (n_r) by a new solution is bounded by a small value, which contributes to the diversity in Pop. iii) Each subproblem index k has an extra parameter that counts the number of consecutive iterations in which its solution σ^k has not been improved. When this value reaches a user-specified bound, the shaking procedure is performed to move its solution to another region of the search space.

IV. Experimental study

A. Benchmark description and reference sets for comparison

The instances of the SDST flowshop used in the experiments have been created by Ciavotta et al. [6], which is based on the well-known Taillard's benchmark [21]. That benchmark varies according to the number of jobs (n) and number of machines (m). The 11 different combinations are $n = \{20, 50, 100\} \times m = \{5, 10, 20\}$ and $n = 200 \times m = \{10, 20\}$, being 10 instances for each combination (110 instance in total). The processing times $p_{i,k}$ were randomly generated in the range $[0, 99]$. The extended benchmark of [6] contains 220 instances, composed of two sets, referenced as SSD50 and SSD125. Setup times in SSD50 and SSD125 are generated to be 50% and 125% of the processing times $(p_{i,k})$, respectively. The due dates (d_i) were generated according to $d_i = P_i \times (1 + random.3)$, where $P_i = \sum_{k=1}^{m} p_{i,k}$, and $random$ is a uniform random number between $[0, 1]$. The instances are also assigned with a set of weights for each job in the range $[1, 10]$. Moreover, the authors of the benchmark have provided a third set, referenced as SSDTest, containing only four instances for each combination (44 instances in total), which have been used to calibrate the algorithms. The benchmark is available at the repository website[1].

As mentioned in Section I, few studies have dealt with multi-objective SDST minimizing makespan and total weighted tardiness. The same authors of the benchmark adapted and evaluated a total of 17 algorithms (both general and FSP specific). The results of that study are available and well documented at the same repository of the benchmark [6]. In our experimental study, we have selected the results (in the form of approximated *PFs*) obtained by RIPG and MOSA_VM because, in that paper, they were stated as the two best-performer approaches.

B. Performance assessment and experimental setup

We assess the performance of the algorithms by means of three different multi-objective performance indicators.

The *Pareto-compliant Hypervolume* indicator (*HV*) [22] measures the size (volume) of the objective space dominated by all solutions in a given approximated *PF* and bounded by a given reference point $\boldsymbol{z}^r = (z_1^r, \ldots, z_q^r)^T$. The higher the hypervolume value, the better the approximated *PF*. In order to compute *HV*, the obtained approximated *PFs* are normalized between $[0, 1]$ using the *max-min* normalization. Since we are dealing with the minimization of a bi-objective problem, the reference point is set to $\boldsymbol{z}^r = (1.01, 1.01)$.

The *set coverage* indicator (*C-metric*) [23] measures the "degree" of dominance of an approximation *PF* over another by computing the proportion of solutions in *B* that are dominated by at least one solution in *A*. *C(A,B) = 1* means that all solutions in *B* are dominated by some solution in *A*, while *C(A,B)=0* means that no solution in *B* is dominated by a solution in *A*. Formally,

$$C(A, B) = \frac{|\{u \in B | \exists v \in A : v \ dominates \ u\}|}{|B|} \tag{4}$$

Furthermore, $C(A, B)$ is not necessarily equal to $1 - C(B, A)$.

The *empirical attainment function* (EAF) allows us to graphically analyze the distribution of the approximation Pareto sets in the objective space over several runs [24]. An *attainment surface* delimits the region of the objective space attained by an

[1] Available at http://soa.iti.es/problem-instances

algorithm with a certain minimum frequency. For instance, the median attainment surface delimits the region of the objective space attained by half of the runs of the algorithm. Examining the empirical attainment surfaces, we can assess the likely location of the outcomes of an algorithm. The graphical tool *Differential* EAF (Diff-EAF) [25] compares EAFs of pairs of algorithms, identifying the regions of the objective space where one algorithm performs better than another. The difference between two EAFs is shown side-by-side. Each plot represents the difference in favor of one algorithm, in which the proportion of the difference is expressed in gray levels, i.e., the darker the color, the stronger the difference at that location of the objective space in favor of that algorithm. The main drawback of the Diff-EAF is that one plot is generated for each instance and pair of algorithms.

C. Experimental setup and parameter settings

MOLS/D and the MOEA/D variant have been instantiated from the same framework written in C++ using Ubuntu 16.04. The experiments have been conducted on a PC with Intel Xeon E5-1650 3.5 GHz $\times 12$ processor and 32 GB memory.

The general parameters for both are the same, and they are listed in Algorithm 1. According to a preliminary study, the parameter settings are $N = 100$, $T = 20$, and $n_r = 2$. For the shaking procedure, a subproblem solution is perturbed with $n_{sh} = 14$ random *insert moves* when it reaches n consecutive iterations without an improvement.

The remaining parameter settings of MOEA/D are in accordance to [8], in which the neighborhood size for selection is 10, and the genetic operators employed are the *two-point crossover* and the *1-insert mutation* (both with 100% probability).

The algorithms stop after $MaxIte = 1000n$ iterations. Each algorithm was run 20 independent times for each problem instance. In all comparison results, we have carried out the *Friedman's* ranked-based test at 95% of confidence level ($\alpha = 0.05$) to assess whether the difference in the results is statistically significant. The *post-hoc* Nemenyi is used to rank the results if a statistical difference is found.

D. Analysis of MOLS/D components

We have conducted a sensitive analysis of the MOLS/D components. For this purpose, we have used the *SSDTest*, which contains 44 instances. First, we have studied the influence of the local search operator according to the *1-interchange* and *1-insert* moves. Figure 1 depicts the averaged *HV* values obtained by MOLS/D using the *1-insert* and *1-interchange* over the 20 runs $\times$ 44 instances (880 data points in total). We can notice that the algorithm with the *1-insert* operator achieves better *HV* results compared to *1-interchange*. According to the statistical test over 880 data points per instantiated algorithm, MOLS/D with *1-insert* is significantly superior.

We have also analyzed the influence of the heuristic initialization and the controlled shaking procedure. Figure 2 shows the *HV* values achieved by MOLS/D, over the 44 instances, with four different configurations: (i) without both the heuristic initialization and the shaking procedure (referenced as LS),

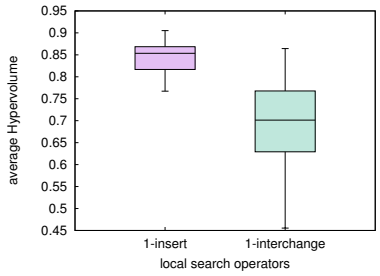

Fig. 1. Boxplot of the average *HV* values obtained by MOLS/D using the *1-insert move* and the *1-interchange move*.

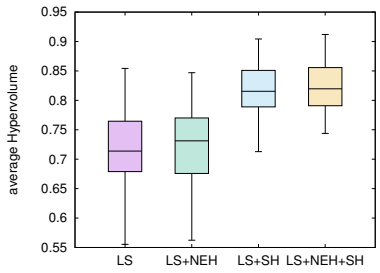

Fig. 2. Boxplot of the average *HV* values obtained by four algorithm configurations: (i) without both heuristic initialization and shaking procedure (LS), (ii) only with the heuristic initialization (LS+NEH), (iii) only with the shaking procedure (LS+SH), and (iv) with all the components together (LS+NEH+SH).

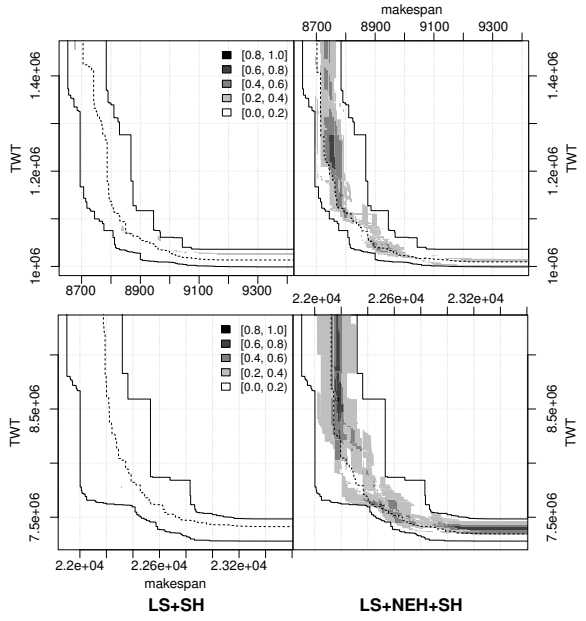

Fig. 3. Diff-EAF between MOLS/D without the heuristic initialization (left) vs. with the heuristic initialization (right) for SSDTest_034 (100×20) (top) and SSDTest_041 (200×20) (bottom).

(ii) only with the heuristic initialization (called as LS+NEH), (iii) only with the controlled shaking procedure (referenced as LS+SH), and (iv) with all components together (defined as LS+NEH+SH). We can notice that the shaking mechanism contributes much more to better results than the heuristic initialization. According to the statistical test aforementioned, LS+SH and LS+NEH+SH have not achieved a significant difference. Overall, the main reason for this behavior can

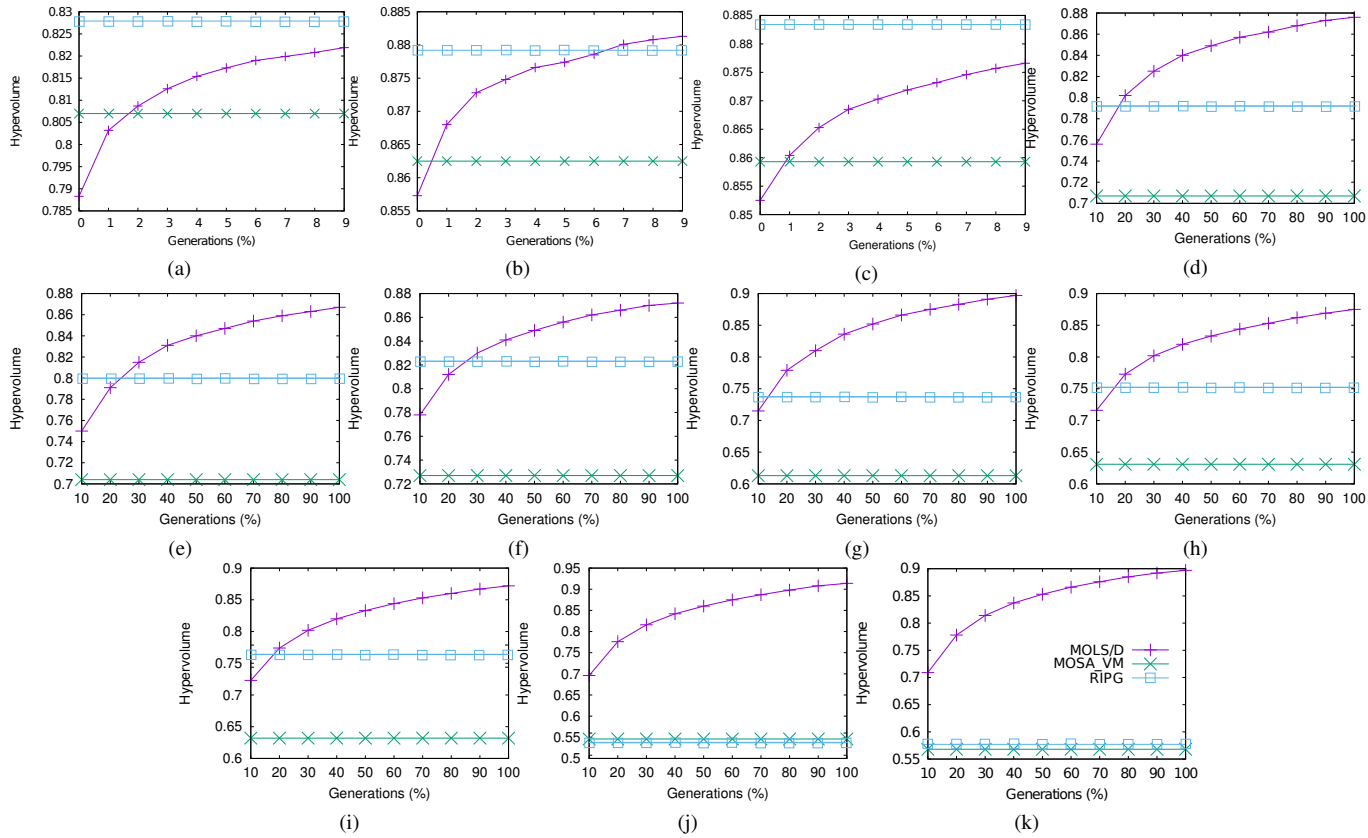

Fig. 4. Average *HV* values obtained by MOLS/D throughout 10 different stages of the search compared to the *HV* obtained by the reference sets of MOSA and RIPG (constant lines) for the problem scales (a) 20×05, (b) 20×10, (c) 20×20, (d) 50×05, (e) 50×10, (f) 50×20, (g) 100×5, (h) 100×10, (i) 100×20, (j) 200×10, and (k) 200×20.

be attributed to the search process of the algorithm which can easily reach solutions close to those provided by the constructive heuristic method. In order to better understand the algorithms behavior with the heuristic initialization, we have analyzed the outcomes using the Diff-EAF toll. Due to space limitations, in Figure 3, we show the Diff-EAF for two representative large problem instances. In each comparison, the continuous lines are the same on each side of the plot, and they represent the overall best and worst *attainment surfaces* (best and worst overall outputs). Any difference between the algorithms is expressed between these two lines. The dashed lines on each side correspond to the median *attainment surface* of each algorithm. The gray levels allow us to identify which algorithm prevails in each region of the objective space. Observing these figures, we can notice that MOLS/D with the heuristic initialization provides better results, mainly towards the extreme regions of the objective space, especially for makespan.

E. Comparison results and discussion

In this section, we compare MOLS/D to the improved MOEA/D and the reference sets from RIPG and MOSA_VM. Since the reference sets were obtained with different stopping conditions, firstly, we have tracked the approximated *PFs* obtained by MOLS/D throughout several stages of the search process in order to have a fair comparison. Due to limitation of space, we present the results for *SSD50*, grouped by the 11 different problem-scales $(n \times m)$. Figure 4 shows the averaged *HV* values reached MOLS/D throughout 10 stages of the search compared to those *HVs* obtained by RIPG and MOSA_VM (which are represented by constant lines). We can see that the MOLS/D is able to keep improving throughout all the process. MOLS/D outperforms MOSA_VM, in terms of *HV*, for all group of instances. MOLS/D outperforms RIPG for all groups of instances, except for $n = 20$. For the problem instances with $n = 50$ (respectively, $n = 100$, and $n = 200$), MOLS/D needs only an average of 30% (respectively, 20%, and 10%) of the total iterations to outperform RIPG.

All comparative results (4 algorithms $\times$ 220 instances $\times$ 2 performance indicators) are given in Table I and Table II. The result in each cell represents a problem-scale which reflects the average over 20 runs $\times$ 10 instances (200 data points in total). The best results, according to the statistical test, are highlighted in bold.

We can observe in Table I that, in terms of *HV* indicator, MOLS/D achieves the best results for most cases. According to the statistical test, MOLS/D is significantly better than RIPG in 9 out of 11 groups of instances. In general, the remaining order of the algorithms is RIPG, MOSA_VM, and lastly MOEA/D.

In terms of *C-metric* (Table II), MOLS/D outperforms MOSA_VM and MOEA/D for all groups of instances and sets of benchmarks. For the problem instances with

TABLE I
AVERAGE *HV* VALUES OBTAINED BY MOLS/D, MOEA/D, MOSA_VM, AND RIPG

problem $n \times m$	SSD50				SSD125			
	MOLS/D	MOEA/D	MOSA_VM	RIPG	MOLS/D	MOEA/D	MOSA_VM	RIPG
20×05	**0.847**	0.649	0.832	**0.852**	**0.822**	0.542	0.799	**0.835**
20×10	**0.893**	0.774	0.874	**0.890**	**0.867**	0.684	0.852	**0.871**
20×20	**0.874**	0.712	0.856	**0.881**	**0.863**	0.668	0.834	**0.870**
50×05	**0.877**	0.413	0.704	0.788	**0.869**	0.357	0.656	0.812
50×10	**0.869**	0.440	0.706	0.801	**0.856**	0.412	0.659	0.816
50×20	**0.867**	0.487	0.717	0.817	**0.872**	0.445	0.678	0.831
100×05	**0.897**	0.348	0.612	0.735	**0.874**	0.251	0.509	0.764
100×10	**0.875**	0.382	0.632	0.752	**0.862**	0.309	0.517	0.767
100×20	**0.873**	0.377	0.633	0.763	**0.885**	0.347	0.554	0.787
200×10	**0.914**	0.320	0.549	0.538	**0.906**	0.285	0.420	0.549
200×20	**0.896**	0.375	0.565	0.574	**0.906**	0.369	0.479	0.627

TABLE II
AVERAGE *C-metric* VALUES REACHED BETWEEN A = MOLS/D AND B = {MOEA/D, MOSA_VM AND RIPG}

problem $n \times m$	SSD50						SSD125					
	B=MOEA/D		B=MOSA_VM		B=RIPG		B=MOEA/D		B=MOSA_VM		B=RIPG	
	C(A,B)	*C(B,A)*	*C(A,B)*	*C(B,A)*	*C(A,B)*	*C(A,B)*	*C(A,B)*	*C(B,A)*	*C(A,B)*	*C(B,A)*	*C(A,B)*	*C(B,A)*
20×05	**0.881**	0.140	**0.624**	0.483	0.524	**0.658**	**0.953**	0.052	**0.613**	0.487	0.474	**0.643**
20×10	**0.866**	0.338	**0.749**	0.551	0.651	**0.689**	**0.930**	0.109	**0.647**	0.483	0.531	**0.627**
20×20	**0.873**	0.502	**0.817**	0.590	0.715	**0.794**	**0.878**	0.519	**0.789**	0.644	0.698	**0.776**
50×05	**1.000**	0.002	**0.947**	0.091	**0.856**	0.431	**1.000**	0.000	**0.958**	0.124	**0.846**	0.537
50×10	**1.000**	0.003	**0.951**	0.064	**0.809**	0.531	**1.000**	0.008	**0.876**	0.144	0.592	**0.665**
50×20	**1.000**	0.008	**0.950**	0.065	**0.793**	0.480	**1.000**	0.000	**0.963**	0.095	**0.767**	0.605
100×05	**1.000**	0.000	**0.989**	0.016	**0.945**	0.235	**1.000**	0.000	**0.983**	0.021	**0.817**	0.488
100×10	**1.000**	0.000	**0.948**	0.022	**0.873**	0.309	**1.000**	0.000	**0.953**	0.021	**0.722**	0.598
100×20	**1.000**	0.001	**0.978**	0.028	**0.937**	0.313	**1.000**	0.000	**0.969**	0.029	**0.773**	0.539
200×10	**1.000**	0.000	**0.991**	0.001	**0.999**	0.001	**1.000**	0.000	**1.000**	0.000	**1.000**	0.016
200×20	**1.000**	0.000	**0.997**	0.001	**1.000**	0.002	**1.000**	0.000	**0.999**	0.002	**0.997**	0.035

$n = \{50, 100, 200\}$, the final approximated *PFs* obtained by MOLS/D dominate an average of 100% of those achieved by MOEA/D. Moreover, the final approximated *PFs* obtained by MOLS/D are better than those obtained by RIPG for the group of instances with $n = \{50, 100, 200\}$. Besides, the difference between MOLS/D and RIPG is more significant for *SSDT50* than *SSD125*.

In Figure 5, we illustrate the Diff-EAF between MOLS/D and RIPG for three representative instances. Observing the attainment surfaces in both sides and the difference in levels of gray color, we can remark that MOLS/D outperforms RIPG mainly in the region of the objective space that focuses on the makespan criterion. Furthermore, the difference decome greater as the problem scale increases.

Moreover, the instantiated algorithms achieve a similar computational time and the time increases as the problem-scale increases. MOLS/D achieves an average computational time of 8.5 seconds (respectively 1126 seconds) for the smallest (respectively largest) problem-scales.

We have established the best-known results for the benchmark considered. We provide the raw results in the form of several approximated *PFs* online[2].

[2]Available at https://github.com/murilozangari/sdst_results.

V. CONCLUSION AND FUTURE WORK

In this paper, we have tackled the sequence-dependent setup times permutation flowshop scheduling minimizing makespan and total weighted tardiness. We have approached the problem with a simple, yet efficient, decomposition-based iterated local search algorithm, referenced to as MOLS/D. Our algorithm follows the main guidelines of the MOEA/D framework to deal with the multi-objective optimization scenario. MOLS/D is also embedded with a heuristic method to initialize the population of solutions. During the search process, for each single-objective subproblem, a single random local search operator is used to exploit the neighborhood. When a subproblem gets stuck in a local optimum, a shaking mechanism is used to perturb its solution. Experiments have been conducted on a set of 220 benchmark instances.

As a main contribution, we show that MOLS/D achieves better results when compared to the state-of-the art approaches from the specific literature. According to the performance indicators and statistical test, MOLS/D is able to outperform a MOEA/D variant that employs genetic operators, and the best-known approximated *PFs*. Thus, we have established new state-of-the-art results for the benchmark considered by making our approximated *PFs* available for further investigation by other researchers.

As future work, we consider the application of MOLS/D to

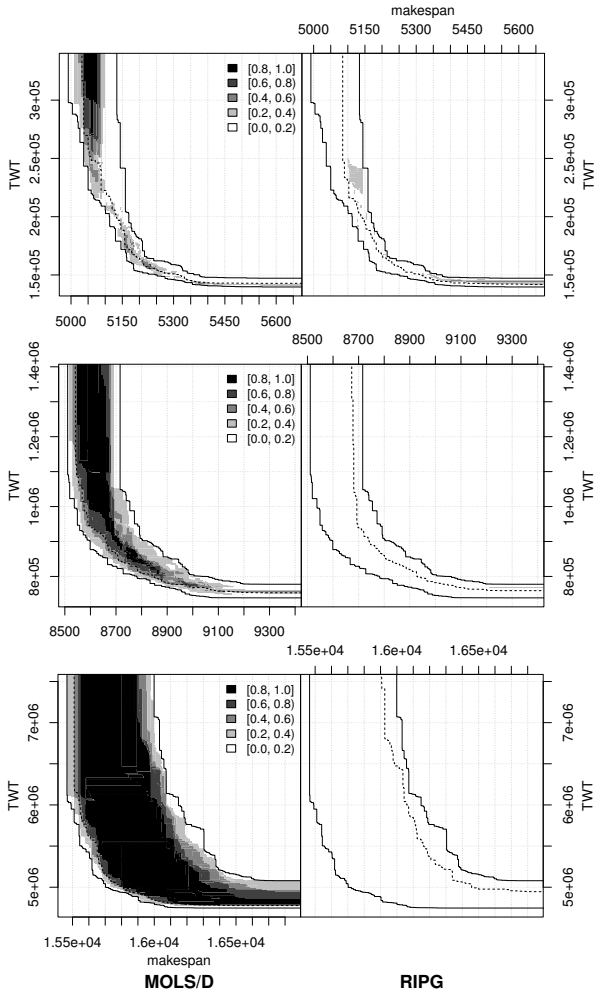

Fig. 5. Diff-EAF between MOLS/D (left) and RIPG (right) for SSD50_051 (50×20) (top), SSD50_071 (100×20) (middle), and SSD50_101 (200×20) (bottom).

solve SDST flowshop with three objectives, and other practical flowshop scheduling problems, such as the dynamic job shop scheduling problem.

VI. Acknowledgments

This work has been supported by the PNPD/CAPES (Brazilian Program of Post-Doctoral), and partially supported by the Research Groups 2013-2018 (IT-609-13) programs (Basque Government) and TIN2016-78365-R (Spanish Ministry of Economy, Industry, and Competitiveness).

References

[1] M. M. Yenisey and B. Yagmahan, "Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends," *Omega*, vol. 45, pp. 119–135, 2014.

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

[3] E. Vallada, R. Ruiz, and G. Minella, "Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics," *Computers & Operations Research*, vol. 35, no. 4, pp. 1350–1373, 2008.

[4] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[5] E. Vallada and R. Ruiz, "Cooperative metaheuristics for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 193, no. 2, pp. 365–376, 2009.

[6] M. Ciavotta, G. Minella, and R. Ruiz, "Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study," *European Journal of Operational Research*, vol. 227, no. 2, pp. 301–313, 2013.

[7] T. Varadharajan and C. Rajendran, "A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs," *European Journal of Operational Research*, vol. 167, no. 3, pp. 772–795, 2005.

[8] P. C. Chang, S. H. Chen, Q. Zhang, and J. L. Lin, "MOEA/D for flowshop scheduling problems," in *Proceedings of the Congress on Evolutionary Computation (CEC)*. IEEE, 2008, pp. 1433–1438.

[9] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 286–300, 2014.

[10] R. Ruiz, C. Maroto, and J. Alcaraz, "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics," *European Journal of Operational Research*, vol. 165, no. 1, pp. 34–54, 2005.

[11] J. Xu, C.-C. Wu, Y. Yin, and W.-C. Lin, "An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times," *Applied Soft Computing*, vol. 52, pp. 39–47, 2017.

[12] X. Li and M. Li, "Multiobjective local search algorithm-based decomposition for multiobjective permutation flow shop scheduling problem," *IEEE Transactions on Engineering Management*, vol. 62, no. 4, pp. 544–557, 2015.

[13] A. Trivedi, D. Srinivasan, K. Sanyal, and A. Ghosh, "A survey of multiobjective evolutionary algorithms based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 3, pp. 440–462, 2017.

[14] A. Alhindi and Q. Zhang, "MOEA/D with tabu search for multiobjective permutation flow shop scheduling problems," in *Proceedings of the Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 1155–1164.

[15] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[16] M. Zangari, A. Mendiburu, R. Santana, and A. Pozo, "Multiobjective decomposition-based Mallows models estimation of distribution algorithm. A case of study for permutation flowshop scheduling problem," *Information Sciences*, vol. 397, pp. 137–154, 2017.

[17] L. Hernando, F. Daolio, N. Veerapen, and G. Ochoa, "Local optima networks of the permutation flowshop scheduling problem: Makespan vs. total flow time," in *Proceedings of the Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 1964–1971.

[18] V. Pareto, *Cours d'économie politique*. Librairie Droz, 1964, vol. 1.

[19] F. Glover and G. Kochenberger, "Iterated local search," in *Handbook of Metaheuristics*. Kluwer, 2002.

[20] W. Liu, Y. Jin, and M. Price, "A new improved NEH heuristic for permutation flowshop scheduling problems," *International Journal of Production Economics*, vol. 193, pp. 21–30, 2017.

[21] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[22] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[23] O. Schütze, X. Esquivel, A. Lara, and C. A. C. Coello, "Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 504–522, 2012.

[24] V. Grunert da Fonseca, C. Fonseca, and A. Hall, "Inferential performance assessment of stochastic optimisers and the attainment function," in *Evolutionary multi-criterion optimization*. Springer, 2001, pp. 213–225.

[25] M. López-Ibánez, L. Paquete, and T. Stützle, "Exploratory analysis of stochastic local search algorithms in biobjective optimization," in *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 209–222.