Computers & Operations Research 38 (2011) 1219–1236

Contents lists available at ScienceDirect

Computers & Operations Research

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

A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems

Jérémie Dubois-Lacoste *, Manuel López-Ibáñez, Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles (ULB), 50 Av. F. Roosevelt (ULB), 1050 Brussels, Belgium

ARTICLE INFO

Available online 14 October 2010

Keywords:
Scheduling
Flowshop
Multi-objective
Hybrid

ABSTRACT

This paper presents a new, carefully designed algorithm for five bi-objective permutation flow shop scheduling problems that arise from the pairwise combinations of the objectives (i) makespan, (ii) the sum of the completion times of the jobs, and (iii) both, the weighted and non-weighted total tardiness of all jobs. The proposed algorithm combines two search methods, two-phase local search and Pareto local search, which are representative of two different, but complementary, paradigms for multi-objective optimization in terms of Pareto-optimality. The design of the hybrid algorithm is based on a careful experimental analysis of crucial algorithmic components of these two search methods. We compared our algorithm to the two best algorithms identified, among a set of 23 candidate algorithms, in a recent review of the bi-objective permutation flow-shop scheduling problem. We have reimplemented carefully these two algorithms in order to assess the quality of our algorithm. The experimental comparison in this paper shows that the proposed algorithm obtains results that often dominate the output of the two best algorithms from the literature. Therefore, our analysis shows without ambiguity that the proposed algorithm is a new state-of-the-art algorithm for the bi-objective permutation flow-shop problems studied in this paper.

1. Introduction

The permutation flow-shop scheduling problem (PFSP) is a well-known scheduling problem that models production environments found, for example, in manufacturing or chemical industries.

In flow-shop problems, each job consists of a set of operations, each of which is to be carried out on a particular machine, and the order of machines is the same for all jobs. For flow-shops, several objectives have been considered as optimization goals. Common objectives that are to be minimized include the completion time of the last job (makespan) [1], the sum of completion times of all jobs [2–4] and the total (weighted) tardiness [5]. These objectives are also considered to be important in practical applications. Flow-shop problems with these objectives are $\mathcal{NP}-$hard from a specific number of machines on [6,7]. For makespan this is the case for three and more machines, for the sum of flowtimes for two or more, and for the total (weighted) tardiness this is the case already for a single machine. Therefore, large instances are often tackled by means of approximate (*heuristic*) algorithms, and among these, stochastic local search (SLS) algorithms [8] have proven to be particularly effective.

Many optimization problems encountered in real life can be evaluated according to various, often conflicting objectives. Therefore, since a few decades multi-objective optimization attracts considerable efforts and has become a very active field of research.

A significant amount of recent research deals with the application of SLS algorithms to multi-objective problems and, in particular, to those problems that are tackled without *a priori* information about the decision maker's preferences [9,10]; in this latter case, the goal becomes to find an as good as possible approximation to the Pareto-optimal set.

In recent years, bi-objective PFSPs in terms of Pareto optimality have attracted a substantial research effort. In this paper, we tackle bi-objective PFSPs that arise from the pairwise combinations of the objectives makespan, sum of completion times of all jobs and total (weighted) tardiness. The recent comprehensive review and experimental comparison by Minella et al. [11] summarizes the current state-of-the-art for three bi-objective PFSPs, those arising from the pairwise combinations of the objectives makespan, sum of completion times, and total tardiness. They identified the algorithms MOSA [12] and MOGLS [13] as the currently best performing ones. In this paper, we present a new, hybrid state-of-the-art SLS algorithm for these three bi-objective PFSPs and also for two variants that replace the total tardiness objective with its weighted version.

Our algorithm combines two multi-objective frameworks: two-phase local search (TPLS) [14] and Pareto local search (PLS) [15]. TPLS uses a single-objective algorithm to solve several *scalarizations*, that is, aggregations of the multiple objective functions into a single-objective problem. PLS exhaustively

* Corresponding author. Tel.: +32 2 650 27 45; fax: +32 2 650 27 15.
E-mail addresses: jeremie.dubois-lacoste@ulb.ac.be (J. Dubois-Lacoste), manuel.lopez-ibanez@ulb.ac.be (M. López-Ibáñez), stuetzle@ulb.ac.be (T. Stützle).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.10.008

searches for non-dominated solutions in the neighborhood of an initial set. Lust and Teghem [16] have recently also combined search methods based on scalarizations and dominance for tackling the multi-objective TSP. Apart from differences in several details, their approach focuses on algorithms that have a natural stopping criteria, whereas our hybrid algorithm is designed with a user-defined time limit in mind.

For the design of our SLS algorithm we follow a bottom-up SLS algorithm engineering approach, and our first step is the development of high-performing algorithms for each single objective PFSP. As a starting point, we choose the iterated greedy (IG) algorithm of Ruiz and Stützle [17], which is a state-of-the-art SLS algorithm for the PFSP for makespan minimization. We extend this algorithm to tackle the other objectives (the sum of flowtime, total tardiness and weighted total tardiness). The result is a high-performing algorithm for each single-objective problem.

In the next step of our algorithm engineering, we further adapt these single-objective algorithms to tackle each of the five bi-objective variants of the PFSP by integrating the IG algorithms into the TPLS framework. We also implement PLS algorithms that use various neighborhood operators to explore the search space. We carry out a careful experimental study of the main algorithmic components of the TPLS and PLS algorithms and then exploit the gained insights to propose a hybrid SLS algorithm that combines both algorithms.

In order to perform a fair evaluation of the quality of the resulting hybrid algorithm, we reimplemented MOSA [12] and MOGLS [13], two state-of-the-art algorithms for bi-objective PFSPs [11]. Our experimental results show the excellent performance of our proposed hybrid algorithm: it often finds Pareto fronts that completely dominate those found by these two state-of-the-art algorithms.

The contributions of this paper are several high-performing single-objective algorithms for various PFSPs, improved variants of the TPLS and PLS frameworks, and a method to engineer SLS algorithms for bi-objective problems. We further combine these individual contributions to produce a new hybrid algorithm that advances the state-of-the-art for the five bi-objective PFSPs.

2. Preliminaries

In this section, we first introduce the flow-shop scheduling problem. Next, we give some basic notions on multi-objective optimization that are required in the rest of the paper. Finally, we present TPLS and PLS, the two algorithm frameworks hybridized in our final algorithm.

2.1. Permutation flow-shop scheduling problem

The flow-shop scheduling problem (FSP) is one of the most widely studied scheduling problems. In the FSP, a set of n jobs ($J_1,\ldots J_n$) is to be processed on m machines ($M_1,\ldots,M_m$). All jobs go through the machines in the same order, i.e., all jobs have to be processed first on machine M_1, then on machine M_2, and so on until machine M_m. This results in a set of $(n!)^m$ different candidate solutions. A common restriction in the FSP is to forbid job passing, i.e., the processing sequence of the jobs is the same on all machines. In this case, a candidate solution can be represented as a permutation of the jobs and, hence, there are $n!$ possible sequences. The resulting problem is called permutation flow-shop scheduling problem (PFSP).

In the PFSP, all processing times p_{ij} for a job J_i on a machine M_j are fixed, known in advance, and non-negative. For a given job permutation π, π_i denotes the job in the ith position. Let C_{ij} denote the completion time of job J_i on machine M_j, then the completion times of all jobs on all machines are given by the following formula:

$$C_{\pi_0 j} = 0, \quad j = 1,\ldots,m, \tag{1}$$

$$C_{\pi_i 0} = 0, \quad i = 1,\ldots,n, \tag{2}$$

$$C_{\pi_i j} = \max\{C_{\pi_{i-1} j}, C_{\pi_i j-1}\} + p_{ij}, \quad i = 1,\ldots,n, \; j = 1,\ldots,m. \tag{3}$$

For simplicity, in the remainder of the paper, C_i denotes the completion time of a job J_i on the last machine M_m. The *makespan* is the completion time of the last job in the permutation, that is, $C_{\max} = C_{\pi_n}$. In the following, we refer to the PFSP with makespan minimization as *PFSP-*$C_{\max}$; this problem is $\mathcal{NP}-$hard in the strong sense for $m \geq 3$ [7].

The other objectives studied in this paper are the minimization of the *sum of flowtimes* and the minimization of the *weighted tardiness*. Because all jobs are assumed to be available at time zero, the sum of flowtimes is given by $\sum_{i=1}^{n} C_i$. This objective is also known as sum of completion times or total completion time. We refer to the PFSP with sum of flowtimes minimization as *PFSP-SFT*. It is strongly $\mathcal{NP}-$hard for only two machines [7].

In many practical situations, a job has a due date d_i. The tardiness of a job J_i is then defined as $T_i = \max\{C_i - d_i, 0\}$ and the total weighted tardiness is given by $\sum_{i=1}^{n} w_i \cdot T_i$, where w_i is a weight assigned to job J_i. The problem of minimizing the total weighted tardiness is denoted by *PFSP-WT*. If all weights w_i are the same, we say that the objective is the *total tardiness*, and we name it *PFSP-TT*. The *PFSP-TT* and the *PFSP-WT* are strongly $\mathcal{NP}-$hard even for a single machine [6]. Most of the studies on the tardiness objective, including the review of Minella et al. [11], focus only on the non-weighted variant. However, in this work, we consider both, the *PFSP-TT* and the *PFSP-WT*.

We tackle the bi-objective PFSPs that result from five possible pairs of objectives (we do not consider the combination of the total and weighted tardiness). We refer to these five bi-objective problems as follows. *PFSP-*($C_{\max}$, *SFT*) denotes the minimization of the makespan and the sum of flowtimes, *PFSP-*($C_{\max}$, *TT*) and *PFSP-*($C_{\max}$, *WT*) denote the minimization of the makespan and the total tardiness and weighted tardiness, respectively; *PFSP-*(*SFT*, *TT*) and *PFSP-*(*SFT*, *WT*) denote the minimization of the sum of flowtimes and the total tardiness and weighted tardiness, respectively. A number of algorithms have been proposed to tackle each of these bi-objective problems separately. Rarely, the same paper has addressed more than one combination. Minella et al. [11] give a comprehensive overview of the literature on the three most commonly tackled problems and present the results of a sound and extensive experimental analysis of 23 algorithms. These algorithms are either PFSP-specific or they are more general but have been adapted by Minella et al. to tackle this problem. Their review identifies a multi-objective simulated annealing (MOSA) [12] as the best performing algorithm for all combinations of objectives. They also point out a multi-objective genetic local search (MOGLS) [13] as the best performing alternative. These two algorithms are therefore the current state of the art for the bi-objective PFSP variants tackled in this paper.

2.2. Multi-objective optimization

In multi-objective combinatorial optimization problems (MCOPs), candidate solutions are evaluated according to an *objective function vector* $\vec{f} = (f_1,\ldots,f_q)$ with q objectives. Many early studies used a simple *a priori* approach where some preferences are given among the set of objectives. These preferences can be given as a ponderation for each objective (*scalarized* approach), or as a total order of the objectives (*lexicographic* approach).

If no *a priori* assumptions upon the decision maker's preferences can be made, the goal typically becomes to find a set of feasible solutions that "minimize" $\vec{f}$ in the sense of Pareto optimality, from which the decision maker can choose a final solution *a posteriori*. In such a case, a partial order is given on the set of feasible solutions, defined as follows. If $\vec{u}$ and $\vec{v}$ are vectors in $\mathbb{R}^q$ (q being the number of objectives), we say that $\vec{u}$ *dominates* $\vec{v}$ ($\vec{u} \prec \vec{v}$) iff $\vec{u} \neq \vec{v}$ and $u_i \leq v_i$, $i = 1,\ldots,q$. We say that $\vec{u}$ *weakly dominates* $\vec{v}$ ($\vec{u} \preceq \vec{v}$) iff $u_i \leq v_i$, $i=1,\ldots,q$.

We say that $\vec{u}$ and $\vec{v}$ are mutually *non-dominated* iff $\vec{u} \nprec \vec{v}$ and $\vec{v} \nprec \vec{u}$. We also say that $\vec{u}$ and $\vec{v}$ are *non weakly-dominated* if $\vec{u} \npreceq \vec{v}$ and $\vec{v} \npreceq \vec{u}$. For simplicity, we extend the dominance criteria to solutions, that is, a solution s dominates another one s' iff $\vec{f}(s) \prec \vec{f}(s')$.

If no s' exists such that $\vec{f}(s') \prec \vec{f}(s)$, the solution s is called *Pareto optimal*. The goal in MCOPs typically is to determine the set of all Pareto-optimal solutions. Since this goal is in many cases computationally intractable [9], in practice the goal becomes to find the best possible approximation to the set of Pareto optimal solutions within a particular time limit. Any set of mutually non-dominated solutions provides such an approximation but some approximations are better than others. In fact, the notion of Pareto optimality can be extended to compare sets of mutually non-dominated solutions [18]. In particular, we can say that one set A dominates another set B ($A \prec B$), iff every $\vec{b} \in B$ is dominated by at least one $\vec{a} \in A$. When comparing two sets A and B of non-dominated solutions, we say that A *is better than* B in terms of Pareto optimality ($A \lhd B$) iff every $\vec{b} \in B$ is dominated by or equal to at least one $\vec{a} \in A$, and $A \neq B$. If we have $A \ntriangleleft B$, $B \ntriangleleft A$, and $A \neq B$, then A and B are *incomparable* ($A \| B$), and none of the two sets is preferred over the other according only to Pareto optimality.

2.3. Two-phase local search

TPLS [14] is a general algorithmic framework for multi-objective optimization that consists of two phases. The first phase uses an effective single-objective algorithm to find a high-quality solution for one objective. The second phase solves a sequence of *scalarizations*, that is, weighted sum aggregations of the multiple objectives into a single scalar function. We focus on bi-objective problems. Given a normalized weight vector $\vec{\lambda} = (\lambda, 1-\lambda)$, $\lambda \in [0,1] \subset \mathbb{R}$, the scalar value of a solution s with objective function vector $\vec{f}(s) = (f_1(s), f_2(s))$ is computed as

$$f_\lambda(s) = \lambda \cdot f_1(s) + (1-\lambda) \cdot f_2(s). \tag{4}$$

Henceforth, a weight vector $\vec{\lambda}$ will be simply denoted by its first component λ.

One central idea of TPLS is to use the best solution found by the previous scalarization as the initial solution for the next scalarization. This strategy exploits the connectedness of solutions, that is, solutions that are close to each other in the solution space are expected to be also close in the objective space. In our implementation of TPLS, we first generate a very good solution for each objective because we have good algorithms for each single-objective problem (Section 3). Then, we solve a number of scalarized problems using a sequence of weights. Algorithm 1 describes our implementation of the TPLS framework that generates first one solution for each objective. SLS_1 and SLS_2 denote, respectively, the single-objective algorithms to minimize the first and the second objectives. SLS_Λ denotes the single-objective algorithm to solve scalarized problems. *1to2* is a Boolean determining the direction of the scalarizations, either from the first objective to the second one or vice versa, and therefore it also determines which of the two initial solutions should be used as a seed for the first scalarization. When all scalarizations have been solved, procedure Filter removes dominated solutions from the archive. The weights to define scalarized problems are generated using a recently proposed weight setting strategy [19], which will be described in Section 4.3.

Algorithm 1. Two-phase local search.

```
 1: $\pi_1 := \text{SLS}_1()$
 2: $\pi_2 := \text{SLS}_2()$
 3: Add $\pi_1$, $\pi_2$ to Archive
 4: if 1to2 then $\pi' := \pi_1$ else $\pi' := \pi_2$
 5: for each weight $\lambda$ do
 6:     $\pi' := \text{SLS}_\Lambda(\pi', \lambda)$
 7:     Add $\pi'$ to Archive
 8: end for
 9: Filter(Archive)
10: Output: Archive
```

2.4. Pareto local search

PLS is an iterative improvement method for solving MCOPs. It can be seen as an extension of iterative improvement algorithms for single-objective problems to the multi-objective context [15]. In PLS, an acceptance criterion based on Pareto dominance replaces the usual single-objective acceptance criterion. Algorithm 2 illustrates the PLS framework. Given an initial archive of non-dominated solutions, which are initially marked as unvisited (line 2), PLS iteratively applies the following steps. First, a solution s is randomly chosen among the unvisited ones in the archive (line 5). Then, the neighborhood of s is fully explored and all neighbors that are not weakly dominated by s or by any solution in the archive are added to the archive (lines 7–12). Solutions in the archive dominated by the newly added solutions are removed (procedure Filter on line 13). Once the neighborhood of s has been fully explored, s is marked as visited (line 14). When all solutions in the archive have been visited, the algorithm stops in a Pareto local optimum [20].

Algorithm 2. Pareto local search.

```
 1: Input: An initial set of non-dominated solutions $\mathcal{A}$
 2: explored$(s) :=$ FALSE $\forall s \in \mathcal{A}$
 3: $\mathcal{A}_0 := \mathcal{A}$
 4: while $\mathcal{A}_0 \neq \emptyset$ do
 5:     $s :=$ extract solution randomly from $\mathcal{A}_0$
 6:     $\mathcal{A}_0 := \mathcal{A}_0 \setminus \{s\}$
 7:     for each $s' \in \mathcal{N}(s)$ do
 8:         if $s \npreceq s'$ then
 9:             explored$(s') :=$ FALSE
10:             Add$(\mathcal{A}, s')$
11:         end if
12:     end for
13:     Filter$(\mathcal{A})$
14:     explored$(s) :=$ TRUE
15:     $\mathcal{A}_0 := \{s \in \mathcal{A} \mid \text{explored}(s) = \text{FALSE}\}$
16: end while
17: Output: $\mathcal{A}$
```

We have also implemented a restricted version of PLS called *component-wise step* (CW-step). CW-step adds non-dominated solutions in the neighborhood of the initial solutions to the archive, but it does not explore the neighborhood of these newly added solutions further. CW-step may be interpreted as a specific variant of PLS with an early stopping criterion. Because of this early stopping criterion, it is clear that from a solution quality point of view the CW-step generates worse non-dominated sets than PLS. However, compared to running a full PLS, CW-step typically requires only a very small additional computation time and has been found to be useful in practice as a post-processing step of the solutions produced by TPLS [21,14,22].

3. Single-objective SLS algorithms

Since the aim of the TPLS framework is to extend the efficiency of single-objective algorithms to the multi-objective context, the performance of the underlying single-objective algorithms used by TPLS is crucial. In fact, they should be state-of-the-art algorithms for each single-objective problem, and as good as possible for the scalarized problems resulting from the weighted sum aggregations. In this section, we describe the algorithms used to solve each of the single-objective problems.

3.1. SLS algorithm for PFSP-C_{max}

For the *PFSP*-$C_{\max}$, we re-implemented the iterated greedy (IG) algorithm (*IG*-$C_{\max}$) by Ruiz and Stützle [17]. The IG algorithm has shown to be very competitive when compared to more complex SLS algorithms, and it requires only very few parameters to be set. Algorithm 3 gives an algorithmic outline of IG. The essential idea of IG is to iterate over the following steps. First, IG partially destructs a complete solution by removing some of its components (procedure Destruction). Next, a greedy constructive heuristic reconstructs the partial solution (procedure Reconstruction). A local search algorithm may further improve the newly constructed complete solution (procedure LocalSearch). Finally, an acceptance criterion determines whether the new solution replaces the current solution for the next iteration.

Algorithm 3. Iterated greedy (IG).

```
1: $\pi :=$ Generate Initial Solution
2: **while** termination criterion not satisfied **do**
3:     $\pi_R :=$ Destruction($\pi$)
4:     $\pi' :=$ Reconstruction($\pi_R$)
5:     $\pi' :=$ LocalSearch($\pi'$)     /* optional */
6:     $\pi :=$ AcceptanceCriterion($\pi$,$\pi'$)
7: **end while**
8: **Output**: $\pi$
```

IG-$C_{\max}$ uses the well-known NEH constructive heuristic [23] to construct the initial solution and to reconstruct a full solution from a partial one in the main IG loop. NEH sorts the jobs in descending order of their sum of processing times and inserts them following this order in the partial solution at the best position according to the objective value. When using *IG*-$C_{\max}$, this ordering is only used when NEH creates the initial solution. In the main loop of IG, the algorithm reconstructs a complete solution by reinserting previously removed jobs in random order. After reconstruction, the solution is improved by a first-improvement local search based on the *insert* neighborhood. This neighborhood is defined such that all π' are neighbors of π if they can be obtained from π by removing a job π_i and inserting it at a different position j. The local search scans the neighborhood job by job. For a job π_i, it determines the best position where it can be inserted. If this best move improves the objective value, it is immediately applied. These steps are then repeated with the next job until a local optimum is found. We use a speed-up proposed by Taillard [24] to find the best position to insert a job in $O(mn)$.

The acceptance criterion uses the Metropolis condition: A worse solution is accepted with a probability given by $\exp\{(f(\pi)-f(\pi'))/T\}$, where $f(\pi)$ and $f(\pi')$ are the objective values of the current and new solution, respectively. T is a constant computed as

$$T = T_c \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}, \qquad (5)$$

which is equivalent to the average of the processing times of the jobs over all machines divided by 10 and multiplied by a constant T_c, which is a user-defined parameter that has to be adjusted. The idea behind the formula is to adapt the acceptance probability to the instance size and to the variability of the objective function. Ruiz and Stützle [17] report some experiments to identify good parameter settings. According to their findings, the algorithm is quite robust to different parameter settings. They finally set the number of removed jobs d to 4 and T_c to 0.4, and we use the same parameter settings.

3.2. SLS algorithm for PFSP-SFT

Given the very good performance of IG for makespan minimization (*PFSP*-$C_{\max}$), we decided to adapt the IG algorithm to tackle the *PFSP-SFT*. Although the main outline of IG (Algorithm 3) remains the same, several modifications are necessary to reach a high performance for the *PFSP-SFT*. In particular, the speed-up proposed by Taillard for exploring the insertion neighborhood is only valid for makespan minimization. Without this technique, the complexity of exploring the full insertion neighborhood becomes $O(mn^3)$, and there is no clear *a priori* advantage of using this neighborhood operator over pairwise exchanges of jobs. Therefore, we implement and test three neighborhood operators based on the following moves: *insertion*, which is the same as for makespan minimization but without the speed-up; *exchange*, which exchanges the positions of any pair of jobs; and *swap*, which considers only swaps of the positions of adjacent jobs. Our implementation takes advantage of the following observation: when a job in a (partial or complete) solution is moved to an earlier or later position, this move does not affect the completion times of jobs that precede the affected positions in the schedule. Therefore it is not required to recompute the completion times of unaffected jobs, which effectively halves the time of the neighborhood search. Experimental tests [25,26], which are not reported here, showed that the neighborhood operator based on swap moves leads to the best results, and therefore we used this operator to tackle the *PFSP-SFT*. More precisely, our local search sequentially examines all possible swap moves, and if it finds an improvement, it performs the move (first improvement) and continues the evaluation of the remaining moves. Then, if the objective value has been improved, a new sequential evaluation can be performed from the current solution in order to reach a local optimum.

We also consider the possibility of stopping the iterative improvement algorithm before reaching a local optimum. For this purpose, we add a parameter N_{LS} that limits the number of neighborhood scans. Experimental tests suggest that the local search often finds a local optimum in less than five neighborhood scans. Therefore, we test possible settings of N_{LS} in $\{1,2,3,4,\infty\}$. If $N_{\text{LS}} = \infty$, the search stops at a local optimum, no matter how many neighborhood scans it takes.

For the initial solution, we use the same NEH algorithm as for *PFSP*-$C_{\max}$ since it was shown to provide good quality solutions for *PFSP-SFT* as well [27].

We modified the formula for computing the temperature in the acceptance criterion (Eq. (6)) because of the different range of objective values. We experimentally found [26] that good results are produced by using the formula

$$T = T_c \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{m \cdot 10}, \qquad (6)$$

which is the same as Eq. (5) but multiplied by n.

3.3. SLS algorithm for PFSP-TT and PFSP-WT

We also adapted IG to tackle the *PFSP-TT* and the *PFSP-WT*. There are many constructive heuristics for the tardiness criterion, none of them being really optimized for the weighted tardiness. Therefore, we compared several constructive heuristics to find the best one for these two objectives.

The well-known SLACK dispatching rule defines an order of jobs, and it is often used as a simple constructive heuristic [5] to provide acceptable solutions. We extended SLACK to take into account the jobs weights, and we call this variant the WSLACK heuristic. Our evaluation of WSLACK and other heuristics, reported in [25], has shown that using WSLACK to provide the initial order for the NEH algorithm produces the best results. A further modification of the NEH algorithm is necessary for the case when there are several positions for inserting a job that minimizes the objective value of a (possibly partial) solution. This situation specially arises when the solution is partial and each job can be processed before its due date, and, hence, several insertion positions result in a tardiness value of 0. Our implementation of the NEH algorithm inserts jobs in the earliest position from such a set of equally good positions.

Because of the due dates assigned to the jobs, objective values can differ very much between instances, and we could not find an effective setting based on the input data for the formula of the temperature (Eq. (5)) as for the other two objectives.

Therefore, for the *PFSP-TT* and the *PFSP-WT*, we modified the acceptance criterion to accept a new solution with a probability p, which depends on the relative difference between the objective value of the current and the new solution and a parameter T_c:

$$p=\exp(100\cdot((f(\pi)-f(\pi'))/f(\pi))/T_c). \quad (7)$$

3.4. SLS algorithms for the scalarized problems

Given the very good performance of our IG algorithms to minimize each objective [25,26], we also use IG to solve each scalarized problem. To define the acceptance probability in the procedure AcceptanceCriterion (Algorithm 3, line 6), we use the same formula as for the tardiness objective (Eq. (7)). Another important adaptation of IG for solving the scalarized problems is the normalization of the objective values.

When solving the scalarization of a bi-objective problem, the range of the two objectives may be rather different and, without normalization, the objective with the highest values would be almost the only one to be minimized because of its strong influence on the weighted sum value. For this reason, we compute the weighted sum using *relative values* rather than absolute values. The normalization maps each objective to the range [1,100] by using the worst and the best known values of each objective, with the worst value corresponding to 100 and the best one to 1. Because the best and worst values for each objective change during computation time, that is, the normalization is *dynamic*, we recalculate the weighted sum value of the best known solution before comparing it with the current solution if any objective bounds have changed.

Our normalization procedure also takes into account that the range of the objective function values of partial solutions during the reconstruction phase is smaller than for complete solutions. To overcome this issue, the normalization mechanism keeps bounds for each possible number of jobs in a partial solution, and, thus, uses the adequate normalization for each partial or complete solution. Since the destruction phase removes at most d jobs from a complete solution, the normalization procedure needs to keep d sets of objective bounds corresponding to the d possible number of jobs in a partial or complete solution. Henceforth, we implicitly assume that the appropriate normalization is performed when calculating the weighted sum.

3.5. Parameter tuning

We fine-tuned the parameters d, T_c and N_{LS} of each IG variant for *PFSP-SFT*, *PFSP-TT* and *PFSP-WT*, and for the five scalarized problems. The range of possible values for d was [2,12], for T_c it was (0,10] (if $T_c=0$, a worse solution cannot be accepted), and for N_{LS} the set of possible values was $\{1,2,3,4,\infty\}$. We did not fine-tune the parameters for the makespan minimization problem (*PFSP-C_{max}*), because Ruiz and Stützle [17] have already proposed good parameter settings (see Section 3.1).

Table 1
IG parameter settings.

Problem	d	T_c	N_{LS}
PFSP-C_{max}	4	0.4	∞
PFSP-SFT	5	0.5	3
PFSP-TT	6	0.9	3
PFSP-WT	5	1.2	2
PFSP-(C_{max}, SFT)	5	6	1
PFSP-(C_{max}, TT)	4	5	1
PFSP-(C_{max}, WT)	4	4	1
PFSP-(SFT, TT)	6	5	1
PFSP-(SFT, WT)	6	3	1

The settings for *PFSP-C_{max}* are taken from Ruiz and Stützle [17], in particular the neighborhood they use is based on best insertion moves and is stopped when reaching a local optimum. The other settings were found by means of automatic tuning (Section 3.5).

We used iterated F-Race [28] for the automatic tuning. For this purpose, we generated 100 new instances for each number of jobs in {20, 50, 100, 200}, following the procedure described by Minella et al. [11]. These instances have 20 machines, since they are the hardest considered in this paper. All code is implemented in C++ and compiled with gcc version 3.4.6 using the -O3 flag. Experiments presented in this paper are run on an Intel Xeon E5410 CPU 2.33 GHz with 6 MB cache, under Cluster Rocks Linux. Each process uses one single core due to the sequential implementation of the algorithm. For each problem and each instance size, we performed five independent runs of the automatic tuner and allocate a limit of 10 000 experiments for each run. Each experiment involving the execution of one algorithm configuration on one instance uses a time limit of $(0.1\cdot n\cdot m)/30$ seconds, that is, the time used by Minella et al. [11] divided by 30. (In fact, for the time limits we heuristically adopted those used by Minella et al., since they have run the experiments on a very similar hardware as ours.) The choice of the time limits is also motivated by the assumption that the TPLS phase of the final algorithm will be allocated half of the total time and that we use 15 scalarizations overall.

To assess the importance of the parameter tuning on the solution quality, we compared two versions of our algorithm. In the first, we consider parameter settings that are specific to each number of jobs, which is done by executing multiple tuning runs one for each number of jobs. In the second, we use the same parameter settings for all instance sizes of a problem, which is done by executing the tuning runs using a mix of instances with different sizes. The size-independent parameter settings produce only slightly worse results [29] and they are arguably more robust when applied to instances of an intermediate size, which is not considered in the tuning. Table 1 describes the parameter settings of IG we used to produce all results given later in this paper. We provide as supplementary material [29] results obtained using size-specific parameters.

4. Multi-objective algorithms

In this section, we turn our attention to the bi-objective problems in terms of Pareto optimality. First, we briefly introduce a tool used in the remainder of the paper to analyze and compare multi-objective algorithms. Next, we empirically analyze the main components of the two multi-objective frameworks used in this paper, PLS and TPLS. The results of this analysis lead us to propose a hybrid multi-objective algorithm that combines TPLS and PLS.

This hybrid TP+PLS algorithm will be compared in Section 5 with the state-of-the-art algorithms for the bi-objective PFSPs.

4.1. *Quality assessment of multi-objective algorithms*

The problem of evaluating the quality of multi-objective algorithms is still a current research topic [18]. When trying to decide which of two algorithms is better, the simplest case occurs when the outcome of an algorithm A is better than that of an algorithm B, given the relation defined in Section 2.2. More difficult is the case when the outcomes of two or more algorithms are incomparable. A further difficulty stems from the stochasticity of multi-objective SLS algorithms, which requires to analyze the results of several runs to assess the expected behavior of the algorithm. To overcome these issues, quality indicators have been devised. These are scalar values calculated for each non-dominated set (unary indicators) or pairs of sets (binary indicators).

The hypervolume [30,18] is a unary measure used to evaluate the quality of any non-dominated set. It is commonly employed to compare several multi-objective algorithms by evaluating their outputs when none of them is better than the other in the Pareto sense, as defined in Section 2.2.

Good quality indicators as the hypervolume reflect desirable characteristics of the non-dominated sets, while being consistent with the Pareto optimality principle. Their simplicity, however, does not allow to extract much information about the behavior of the algorithm. A different approach that is well suited for bi-objective problems is the use of exploratory graphical tools based on the *empirical attainment function* (EAF).

The EAF of an algorithm gives an estimate of the probability of an arbitrary point in the objective space being attained by (dominated by or equal to) a solution obtained by a single run of the algorithm [31]. An *attainment surface* delimits the region of the objective space attained by an algorithm with a certain minimum frequency. In particular, the worst attainment surface delimits the region of the objective space that is always attained by an algorithm, whereas the best attainment surface delimits the region attained with the minimum non-zero frequency. Similarly, 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 allows to assess the likely location of the output of an algorithm. In addition to this, differences between the EAFs of two algorithms identify regions of the objective space where one algorithm performs better than another. Given a pair of algorithms, a plot of the differences between their EAFs shows the differences in favor of each algorithm side-by-side and encodes the magnitude of the difference in gray levels: the darker, the stronger the difference at that point of the objective space.

An example of such a plot is Fig. 2, where each side shows the EAF differences in favor of one algorithm over the other. The continuous lines are the same in each side of the plot and they correspond to the overall best and overall worst attainment surfaces, that is, they delimit, respectively, the region attained at least once and the region attained always by any of the two algorithms. These lines provide information about the best and worst overall output, and any difference between the algorithms is contained between these two lines. On the other hand, the dashed lines are different in each side and they correspond to the median attainment surface of each algorithm. López-Ibáñez et al. [32] provide a more detailed explanation of these graphical tools.

4.2. *Analysis of PLS components*

In the following paragraphs, we study the two main components of PLS, namely the initial set of solutions given as input to PLS (the seed of PLS); and the neighborhood operator used for generating new solutions. We also discuss a simple way to improve the *anytime* property [33] of PLS.

Seeding: We analyze the computation time required by PLS and the final quality of its output when seeding PLS with solutions of different quality. We test seeding PLS with (i) one randomly generated solution, (ii) two solutions, one for each single objective, obtained by the appropriate version of the NEH heuristic for each objective (Section 3), and (iii) two solutions obtained by IG (Section 3) for each objective after 10 000 iterations. The neighborhood used for PLS is a combination of exchange and insertion (for details, see the next paragraph on the neighborhood operator).

Fig. 1 gives representative examples of non-dominated sets obtained by PLS for each kind of seed along with the initial seeding solutions of NEH and IG. Generally, seeding PLS with very good initial solutions, as obtained by IG runs, produces better non-dominated sets in terms of a wider range of the Pareto front and better quality. This result is strongest for *PFSP*-($C_{\max}$, *SFT*). As shown on the supplementary material page [29], the differences between the EAFs obtained for each kind of seed across 10 runs confirm this result for the instance of Fig. 1 and also other instances. The computation time required by PLS in dependence of the initial seed is given in Table 2.

The results show that seeding PLS with solutions of higher quality does not strongly affect the computation time required by PLS. From these results, we also expect that seeding PLS with solutions obtained by TPLS will further enhance the quality of the results without an excessive computation time overhead.

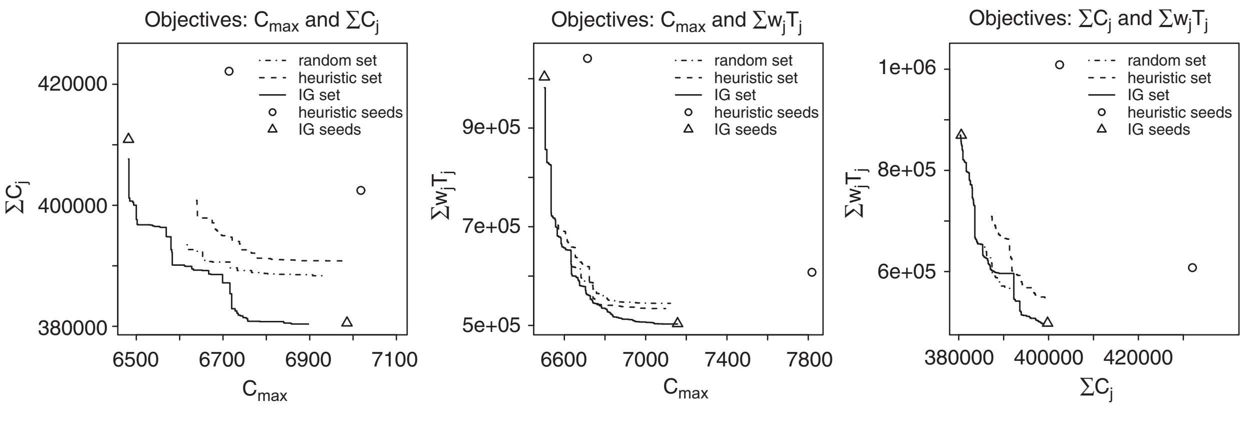

Fig. 1. In each plot is given one non-dominated set obtained by PLS using different quality of seeds for instance `100 × 20_3`. The randomly generated solutions are outside the shown range.

Table 2
Confidence intervals (at level = 0.95) of the computation time (seconds) of PLS for different kinds of seeding solutions (random, heuristic, or IG seeds).

Problems	$n \times m$	Random	Heuristic	IG
PFSP-($C_{\max}$, *SFT*)	50×20	[7.794, 9.896]	[5.596, 6.871]	[4.291, 4.837]
	100×20	[161.4, 193.4]	[134.6, 149.9]	[149.0, 175.3]
PFSP-($C_{\max}$, *WT*)	50×20	[29.17, 34.05]	[31.94, 35.77]	[22.5, 25.53]
	100×20	[577.5, 706.5]	[690.4, 844.1]	[590.4, 662.5]
PFSP-(*SFT*, *WT*)	50×20	[24.27, 29.16]	[27.5, 28.85]	[22.36, 25.03]
	100×20	[673.5, 811.4]	[776.5, 839.0]	[831.7, 958.8]

For details see the text.

Table 3
Confidence intervals (at level = 0.95) of the computation time (seconds) of PLS using different neighborhood operators.

Problems	$n \times m$	Insertion	Exchange	Ex. + Ins.
PFSP-($C_{\max}$, *SFT*)	50×20	[1.379, 1.766]	[1.973, 2.448]	[4.378, 5.303]
	100×20	[65.93, 75.89]	[71.11, 84.01]	[147.6, 167.7]
PFSP-($C_{\max}$, *WT*)	50×20	[9.451, 10.77]	[11.87, 14.02]	[21.67, 24.4]
	100×20	[236.6, 267.1]	[292.9, 336.4]	[577.3, 645.9]
PFSP-(*SFT*, *WT*)	50×20	[8.682, 10.35]	[13.04, 15.45]	[22.04, 25.39]
	100×20	[204.4, 273.7]	[458, 527.9]	[799.1, 945.5]

For details see the text.

Neighborhood operator: Starting from two solutions obtained by IG (Section 3) for each objective after 10 000 iterations, we test variants of PLS based on three different neighborhoods: (i) insertion, (ii) exchange, and (iii) the combination of exchange and insertion. The latter one simply checks for all moves in the exchange and insertion neighborhoods. We measure the computation time of PLS with each neighborhood operator for different combinations of objectives in Table 3. The computation time of the combined exchange and insertion neighborhood is slightly more than the sum of the computation times for the exchange and the insertion neighborhoods separately. For comparing the quality of the results, we examine the EAF differences of 10 independent runs. Fig. 2 gives two representative examples. Typically, the exchange and insertion neighborhoods produce better results in different regions of the Pareto front (top plot), and obviously both of them are consistently outperformed by the combined exchange and insertion neighborhood (bottom plot). Given the complementarity of exchange and insertion to perform well in different regions, we decided to use the combined neighborhood in our hybrid approach.

Continuous improvement (*anytime property*): The original PLS stops when no unexplored non-dominated solutions remain in the archive. When using a limit for the computation time, for instance to compare with other algorithms, PLS may naturally finish before the available time is consumed, and, in this case, the remaining time would be wasted. We therefore modify PLS to continue exploring the search space up to the time limit by extending the neighborhood to those solutions that can be reached by applying to each non-dominated solution the exchange or the insert neighborhood operators twice.

Searching in this extended neighborhood, however, only improves slightly the quality of the results for the smallest instances since on the largest instances the computation time available to PLS was not enough to even finish a single run using the basic exchange and insert neighborhood operators.

4.3. Analysis of TPLS components

In this section, we examine several components of the TPLS framework. Probably the most important is the weight setting strategy, which defines the sequence of weights used by consecutive scalarizations. We present an *adaptive anytime* weight setting strategy that we recently proposed [19] and which was shown to be superior to the classical, deterministic strategies [14]. Next, we examine whether TPLS performs better than a *restart* strategy that generates a new initial solution for each scalarization, independently of previously found solutions. Finally, we discuss appropriate settings for the number of scalarizations.

Weight setting strategy: The original TPLS [14] uses a regular sequence of equally distributed weights defined from either the first objective to the second (*1to2*) or vice versa (*2to1*). The *double* TPLS (D-TPLS) strategy [14] performs the first half of the scalarizations sequentially from one objective to the other, and another half in the inverse direction.

We recently proposed TPLS variants [19] that try to improve the *anytime* property so that for each possible stopping time they reach an as good as possible performance. Among several proposed anytime variants of TPLS, the *Adaptive Focus* variant, henceforth simply called *adaptive* TPLS (A-TPLS), was found to perform overall best. Therefore, we concluded that A-TPLS should be chosen as the weight setting strategy for the PFSP, and we use it as the weight setting strategy of our hybrid algorithm.

A-TPLS is inspired by the *dichotomic* scheme [34,16]. The dichotomic scheme does not define the sequence of weights in advance, but determines them in dependence of the solutions already found. Formally, given a pair of solutions (s_1, s_2), the new weight λ is perpendicular to the segment defined in the objective space by s_1 and s_2, that is,

$$\lambda = \frac{f_2(s_1) - f_2(s_2)}{f_2(s_1) - f_2(s_2) + f_1(s_2) - f_1(s_1)}. \quad (8)$$

A fundamental difference between A-TPLS and the dichotomic scheme is that the latter has a natural stopping criterion, and it progresses recursively depth-first [34,16]. By contrast, A-TPLS explicitly aims to satisfy the anytime property by focusing on the largest gap in the Pareto frontier approximation. Another important difference is that none of the dichotomic schemes proposed in the literature [34,16] uses as seeds the solutions found by previous scalarizations, whereas chaining the scalarizations is the main idea of the TPLS framework.

Algorithm 4. Adaptive "Anytime" TPLS strategy.

```
1:  $s_1 := \mathrm{SLS}_1()$
2:  $s_2 := \mathrm{SLS}_2()$
3:  Add $s_1$, $s_2$ to Archive
4:  $S := \{(s_1, s_2)\}$
5:  **while** not stopping criteria met **do**
6:      $(s_{\sup}, s_{\inf}) := \mathrm{argmax}_{(s,s') \in S}\{\overline{\vec{f}(s)\vec{f}(s')}\}$
7:      Calculate $\lambda$ perpendicular to $\overline{\vec{f}(s_{\sup})\vec{f}(s_{\inf})}$ following Eq. (8)
8:      Calculate $\lambda_1$ and $\lambda_2$ following Eq. (9)
9:      $s_{\sup}' := \mathrm{SLS}_{\Lambda}(s_{\sup}, \lambda_1)$
10:     $s_{\inf}' := \mathrm{SLS}_{\Lambda}(s_{\inf}, \lambda_2)$
11:     Add $s_{\sup}'$ and $s_{\inf}'$ to Archive
12:     Update$(S, s_{\sup}')$
13:     Update$(S, s_{\inf}')$
14: **end while**
15: Filter(Archive)
16: **Output**: Archive
```

Algorithm 4 outlines the main schema of A-TPLS. Initially, two solutions are obtained by optimizing each single objective by means of $\mathrm{SLS}_1()$ and $\mathrm{SLS}_2()$. This pair of solutions is the initial element of set S. At each iteration, the algorithm selects the pair of solutions $(s_{\sup}, s_{\inf}) \in S$ with the largest Euclidean distance between its two normalized objective vectors, and calculates a new weight λ following

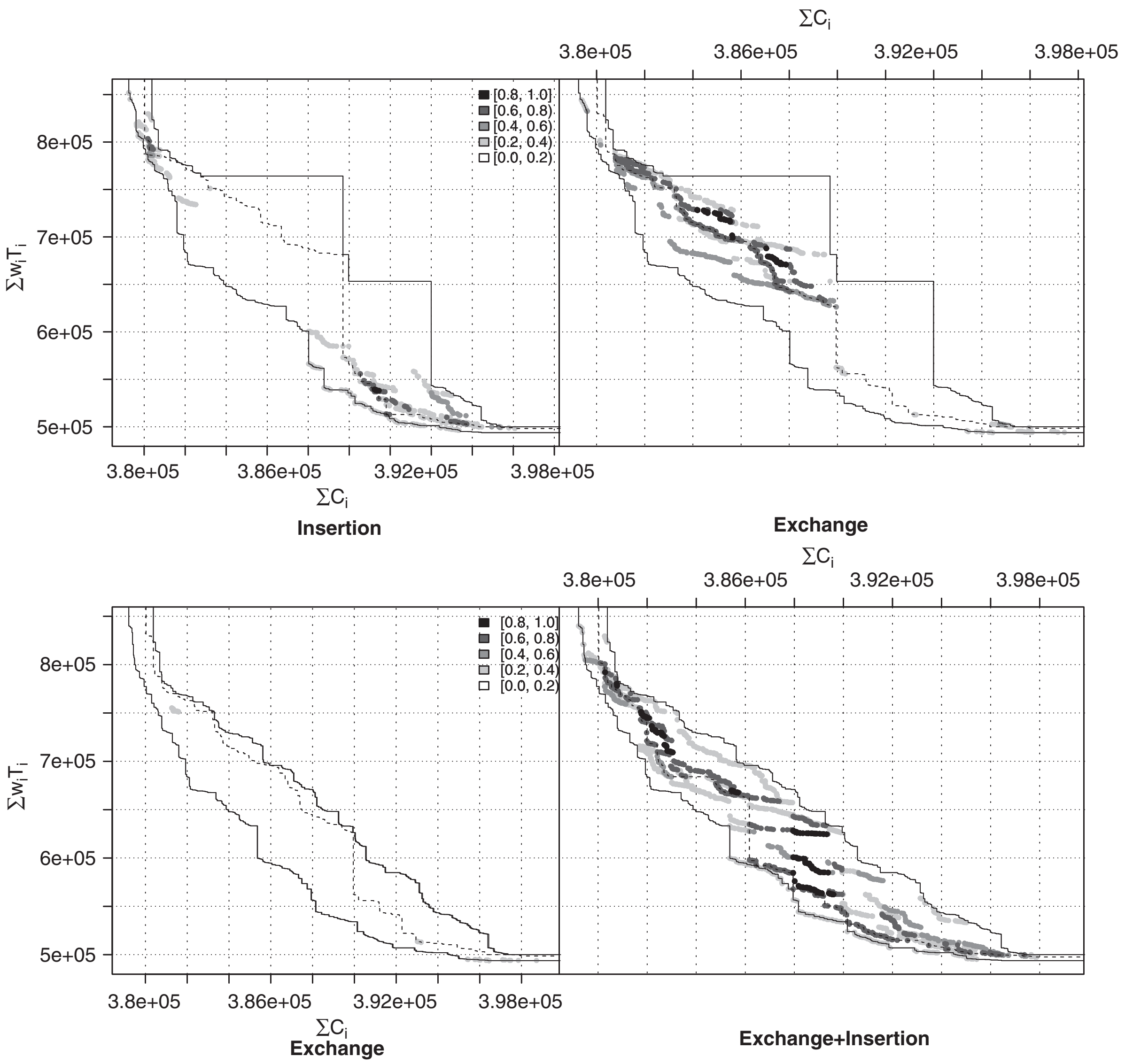

Fig. 2. EAF differences for (top) insertion vs. exchange and (bottom) exchange vs. exchange and insertion, for *PFSP-(SFT, WT)*. Dashed lines are the median attainment surfaces of each algorithm. Black lines correspond to the overall best and overall worst attainment surfaces of both algorithms.

Eq. (8). Then, two new weights λ_1 and λ_2 are calculated from λ:

$$\lambda_1 = \lambda - \theta \cdot \lambda \quad \text{and} \quad \lambda_2 = \lambda + \theta(1-\lambda), \tag{9}$$

where θ is a parameter that modifies λ towards the center of the segment (see Fig. 3).[1] The underlying single-objective SLS algorithm, SLS_Λ, solves then two scalarizations: one from solution s_{sup} using weight λ_1, and another from solution s_{inf} using weight λ_2.

At the end of each iteration, the set of seeds S is updated (procedure Update) as follows: If s' is a newly found solution, it is added to S if it is non-dominated. Solutions in S that possibly become dominated are replaced by s', and any pair of solutions dominated by s' is removed. If a solution s' is accepted for inclusion in S, then the segment $(s_{left}, s_{right}) \in S$ with $f_1(s_{left}) < f_1(s') < f_1(s_{right})$ if it exists, is removed and two new segments (s_{left}, s') and (s', s_{right}) are added to S.

Since each iteration produces two new solutions, a maximum of three new segments is added to S in every iteration. Fig. 4 shows an example of the update of S after the first iteration of the A-TPLS algorithm.

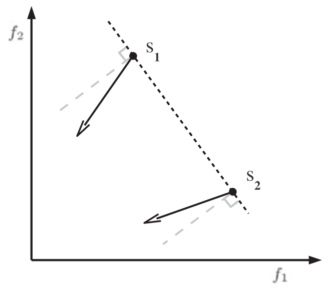

Fig. 3. The scalarization directions from each of the two seeds when $\theta > 0$.

TPLS versus restart: A central idea of TPLS-like strategies is to use the solution found by a previous run of the underlying single-objective algorithm as a seed to initialize the single-objective algorithm in a successive scalarization. A simpler strategy is to use a random or heuristic solution to initialize the underlying single-objective algorithm, effectively making each new scalarization an independent *restart* of the single-objective algorithm.

[1] A value of $\theta > 0$ helps to ensure that scalarizations that start from the same initial solution use different weights. This is particularly important if two adjacent segments in S are almost parallel, since with $\theta = 0$ two scalarizations would be solved using the same seed (the solution shared by the two segments) and very similar weights. In our final algorithm, we use $\theta = 0.25$.

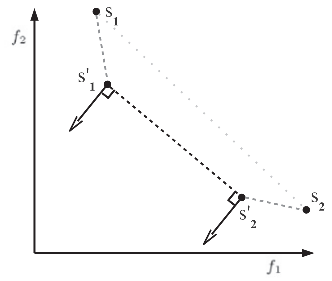

Fig. 4. The first iteration of the A-TPLS algorithm (with $\theta=0$), s'_1 and s'_2 are the newly added solutions. The segment (s'_1,s'_2) is the longest, and, hence, will be explored next.

So far in this paper, we have assumed that TPLS is superior to independent restarts for the bi-objective PFSPs tackled in this paper. To confirm this hypothesis, we performed experiments comparing both strategies.

We implemented a *Restart* strategy derived from A-TPLS. In this *Restart* strategy, the initial solution of each scalarization is generated by variants of the NEH heuristic for the scalarized problems.[2] The *Restart* strategy solves only one scalarization for each pair of solutions (since it does not involve the two different seeds), however, in our experiments it still executes the same number of scalarizations as A-TPLS. We tested the algorithms on five randomly generated instances of size $20 \times 20, 50 \times 20, 100 \times 20$. Each algorithm performs 30 scalarizations, and we limit the overall computation time to $0.05 \cdot n \cdot m$ seconds, equally distributed among all scalarizations. We repeated each experiment 25 times with different seeds for the random number generator.

For the small instances (n=20), there are no clear differences between the two strategies, the differences observed being not consistent. However, for instances of 50 jobs (n=50), we observe a clear improvement of the TPLS strategy over *Restart*. This difference is even stronger for instances of 100 jobs. Fig. 5 illustrates these differences in two particular instances for one combination of objectives, but we obtain similar results for the other bi-objective problems [29]. Therefore, we conclude that the TPLS strategy is a better strategy to tackle the bi-objective PFSPs.

Number of scalarizations: Given a fixed computation time limit, there is a trade-off in TPLS between the number of scalarizations (N_{scalar}) and the computation time allocated to solve each scalarization. Intuitively, the number of non-dominated solutions found, and, hence, how diverse is the resulting approximation to the Pareto front, depends strongly on the number of scalarizations. On the other hand, allocating more time to solve each scalarization may lead to higher quality solutions. We carried out an experimental analysis in order to find a good balance between these two parameters.

We set the total computation time to $(0.05 \cdot n \cdot m)$ seconds, using $(0.005 \cdot n \cdot m)$ seconds for each of the two initial solutions, and dividing the remaining time equally among the scalarizations. As the overall computation time remains the same, increasing the number of scalarizations will decrease the time available to solve each of these, and vice versa. We tested the A-TPLS algorithm with the number of scalarizations $N_{\text{scalar}} \in \{10,20,40,80\}$. We used the hypervolume indicator [18,30] to compare the quality obtained for the four values for N_{scalar}; the objective values are normalized to the range [1,2] such that 2 corresponds to the worst value of the corresponding objective plus one. Then we computed the hypervolume of these normalized non-dominated sets, using (2,2) as the reference point. We used five instances of size 50 × 20_1 and 100 × 20_1 and 25 independent runs of A-TPLS per instance. We performed an analysis of variance (ANOVA) in order to determine if there are significant differences. The difference in the results quality appeared to be rather small, showing that A-TPLS is rather robust to the change of this parameter. However, significant differences are never in disfavor of 10 and 20 scalarizations, and therefore, we focus on a rather small number of scalarizations in our final algorithm.

[2] These variants insert jobs in the best positions according to the scalarized objective value for a given weight.

4.4. TPLS + CW-step, TPLS + PLS

As a final step of our algorithm engineering process, we compare the performance trade-offs incurred by post-processing TPLS results by either PLS or the CW-step (both using the combination of the insertion and exchange neighborhood). For all instances, we generated 10 initial sets of solutions by running TPLS for 30 scalarizations each of 1000 iterations of IG and, in order to reduce variance, we apply CW-step and PLS once to each of these sets.

Table 4 gives the computation time that is incurred by PLS and the CW-step after TPLS has finished. The CW-step incurs only a very minor overhead with respect to TPLS, while PLS requires considerably longer times, especially on instances with 100 jobs. However, the times required for PLS to finish are much lower than when seeding it with only two very good solutions (compare with Table 2). With respect to solution quality, Fig. 6 compares TPLS versus TPLS+CW-step (top), and TPLS+CW step versus TP+PLS (bottom). As expected, the CW-step is able to slightly improve the results of TPLS, while PLS produces much better results. In summary, if the computation time is very limited, the CW-step provides significantly better results at almost no computational cost; if enough time is available, a full execution of PLS gives a further substantial improvement. These conclusions lead us to propose a hybrid TP+PLS algorithm, where a time-bounded PLS is applied to the solutions obtained by TPLS.

4.5. Hybrid TP+PLS algorithm

We designed a final hybrid algorithm that uses TPLS to provide a set of good initial solutions for PLS. This hybrid algorithm uses the IG algorithm for each single objective to obtain two high-quality initial solutions. Then it uses A-TPLS to perform a series of scalarizations and to produce a set of high-quality, non-dominated solutions. This set is then further improved by a time-bounded PLS that uses a combined insertion plus exchange neighborhood operator. The result is a hybrid TP+PLS algorithm for each of the five bi-objective PFSPs. The parameters of TP+PLS are the time given to the initial IG algorithms for each single objective, the number of scalarizations of A-TPLS, the time given to each scalarization, and the time limit of the final PLS run. In the next section, we will examine adequate settings for these parameters and compare the performance of our TP+PLS algorithm with state-of-the-art algorithms.

5. Performance evaluation of TP+PLS

5.1. Experimental setup

For the experimental analysis of TP+PLS, we use the same benchmark instances as Minella et al. [11]. This benchmark set consists of 10 instances of size $\{20,50,100\} \times \{5,10,20\}$ and $\{200\} \times \{10,20\}$, originally proposed by Taillard [35] and augmented with due dates by Minella et al. Recall that these instances are different from the ones we used for the tuning of IG and the design of the TP+PLS algorithm. In other words, we have a

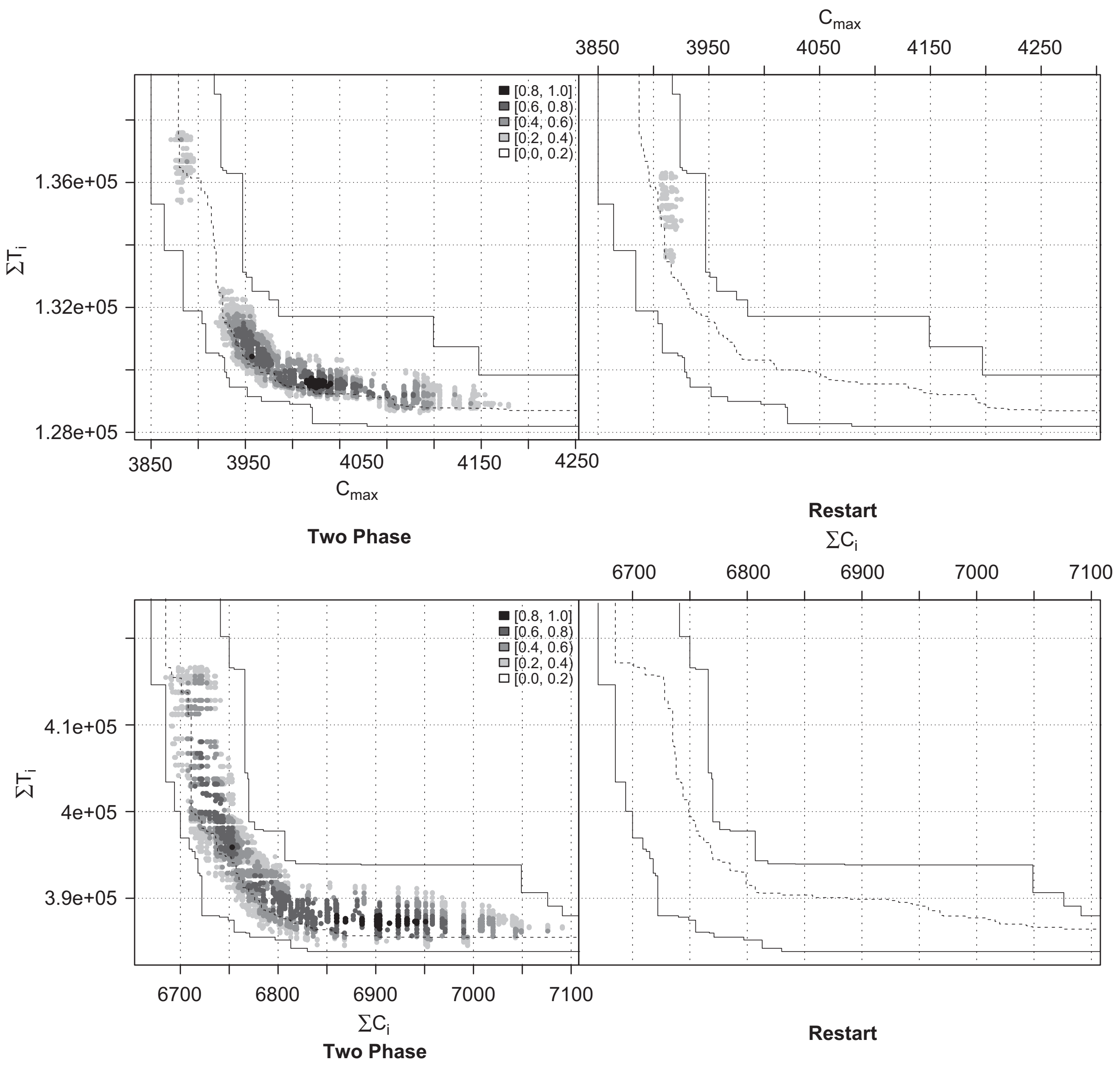

Fig. 5. EAF differences between TPLS on the left and *Restart* on the right. Results are shown for objectives $C_{\max}$ and $\sum C_i$, and instances `50 × 20_1` (top) and `100 × 20_1` (bottom).

Table 4
Confidence intervals (at level = 0.95) of the computation time (seconds) for CW-step and PLS seeding with the output of TPLS.

Problems	$n \times m$	CW-step	PLS
PFSP-($C_{\max}$, *SFT*)	50 × 20	[0.1934, 0.2186]	[1.956, 2.555]
	100 × 20	[1.374, 1.56]	[53.66, 73.59]
PFSP-($C_{\max}$, *WT*)	50 × 20	[0.3542, 0.3808]	[6.484, 7.959]
	100 × 20	[2.289, 2.57]	[209.8, 275.3]
PFSP-(*SFT*, *WT*)	50 × 20	[0.334, 0.3563]	[7.909, 9.446]
	100 × 20	[2.349, 2.569]	[323.7, 385.8]

For details see the text.

clear separation between training instances used to define the algorithms and test instances that are used for comparisons to state-of-the-art algorithms.

Table 5 shows, for a given CPU time, how many iterations can be performed by our implementation, on the hardware environment we used (see Section 4.2), to indicate the relationship between computation time and a measure of operation counts.

Each experiment is run until a time limit of $0.1 \cdot n \cdot m$ seconds, in order to allow a time proportional to the instance size, as suggested by Minella et al. [11]. Each experiment is repeated 25 times with different random seeds. The main parameters of our TP+PLS algorithm are the number of scalarizations (N_{scalar}), and the time required by each scalarization. We perform longer runs of IG for the two single objectives ($\text{IG}_{\{1,2\}}$) than of the IG that solves the scalarizations (IG_{Λ}), with the time assigned to $\text{IG}_{\{1,2\}}$ being 1.5 times the time assigned to IG_{Λ}. Once all scalarizations are finished, the remaining time is spent on PLS.

Table 6 gives the value of these parameters for each instance size. We set these values based on the following considerations. First, we focus on the time settings for instances with `20` machines, and obtain the time settings for instances with `5` and `10` machines by dividing it by 4 and 2, respectively. We assign 200 s to PLS for instances of `200 × 20`, 100 s for instances of `100 × 20`, and 10 s for instances of {`20`, `50`} × `20`. We use 12 scalarizations for all instance sizes. Nonetheless, TP+PLS appears to be very robust with respect to variations of these settings.

Note that the tuning of the IG algorithms in Section 3.5 was done using slightly different computation time limits for each of

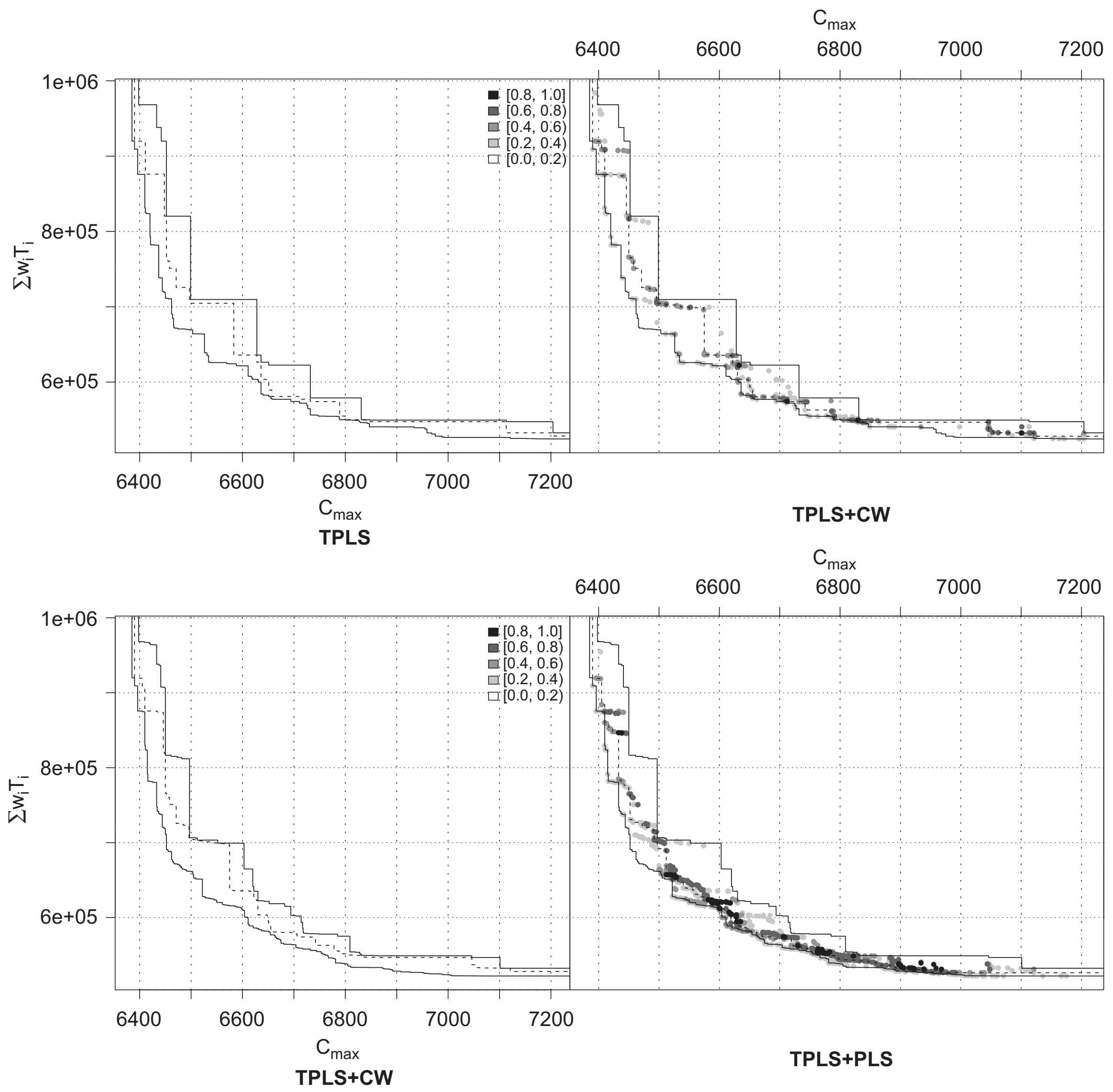

Fig. 6. EAF differences between (top) simple TPLS vs. TPLS + CW-step, and (bottom) TPLS + CW-step vs. TPLS + PLS. Objectives are $C_{\max}$ and $\sum w_i T_i$.

Table 5
Average number of iterations performed in 10 s by IG algorithms, for each instance size.

Size	$C_{\max}$	*SFT*	*TT*	($C_{\max}$, *SFT*)	($C_{\max}$, *TT*)	(*SFT*, *TT*)
20 × 5	68000	78000	60000	70500	65000	66000
20 × 10	30200	48000	42300	42400	41500	45000
20 × 20	14800	29000	26000	25000	24600	28000
50 × 5	15800	18600	14300	16000	14600	15200
50 × 10	4500	11100	9600	9400	9000	9900
50 × 20	2100	5900	5400	5000	4800	5500
100 × 5	4800	6200	4750	5100	4400	4700
100 × 10	1330	3350	2900	2700	2500	2900
100 × 20	500	1650	1550	1350	1300	1550
200 × 10	460	900	800	730	680	790
200 × 20	120	450	420	360	350	420

The numbers for weighted tardiness are similar to those for the total tardiness.

Table 6
Settings for the components of TP+PLS.

Instance size	Time for $\text{IG}_{\{1,2\}}$	Time for IG_{Λ}	N_{scalar}	Overall time
20 × 5	0.75	0.5	12	10
20 × 10	1.5	1.0	12	20
20 × 20	3.0	2.0	12	40
50 × 5	2.25	1.5	12	25
50 × 10	4.5	3.0	12	50
50 × 20	9.0	6.0	12	100
100 × 5	2.5	1.66	12	50
100 × 10	5.0	3.33	12	100
100 × 20	10.0	6.66	12	200
200 × 10	10.0	6.66	12	200
200 × 20	20.0	13.33	12	400

$\text{IG}_{\{1,2\}}$ denotes the IG algorithms that optimize each single objective. IG_{Λ} denotes the IG algorithm that solves scalarizations. N_{scalar} denotes the number of scalarizations (it does not include $\text{IG}_{\{1,2\}}$). PLS is run until the overall computation time is reached. Computation times are given in seconds. N_{scalar} does not include the runs of IG for the two initial solutions.

the IG runs. In fact, we did not repeat the tuning of IG for these slightly different computation time limits, since we anyway hope the final IG algorithm to be relatively robust with respect to the parameter settings so that a re-tuning would not result in very strong gains in solution quality. The very high performance of the hybrid algorithm, as shown in the following, confirms this assumption.

5.2. Comparison with reference sets

We first compare the results of our TP+PLS with the reference sets provided by Minella et al. [11]. These reference sets correspond to the non-dominated points from all outcomes of 10 independent runs of 23 heuristics and metaheuristics, including algorithms for specific PFSP variants or adaptations of algorithms originally proposed for other problems. Each of those runs was stopped after the same time limit as our TP+PLS. These reference sets were obtained on an Intel Dual Core E6600 CPU running at 2.4 GHz, which is similar in speed to the CPU we use. As illustrative examples of the comparison between TP+PLS and the reference sets, Fig. 7 shows the best, median and worst attainment surfaces of TP+PLS together with the points of the reference set corresponding to that instance.

The plots show that the median attainment surface of TP+PLS typically matches and is often better than the reference set. That is, in at least half of the runs, TP+PLS obtains better solutions than those from the reference set. Moreover, the worst attainment surface of TP+PLS sometimes dominates the reference set. In such cases, the worst solutions obtained by TP+PLS in 10 runs dominate all the solutions of the reference set. This result is consistent across all instances and all combinations of objectives, and it indicates the high quality of the non-dominated sets obtained by our TP+PLS algorithm.

5.3. Comparison with MOSA and MOGLS

Given the good quality of TP+PLS suggested by the comparison with reference sets, we next compare the results of TP+PLS with multi-objective simulated annealing [12] and multi-objective genetic local search [13], two algorithms that have recently been shown to be state-of-the-art for various bi-objective PFSPs [11]. To make this comparison more fair and account for possible differences

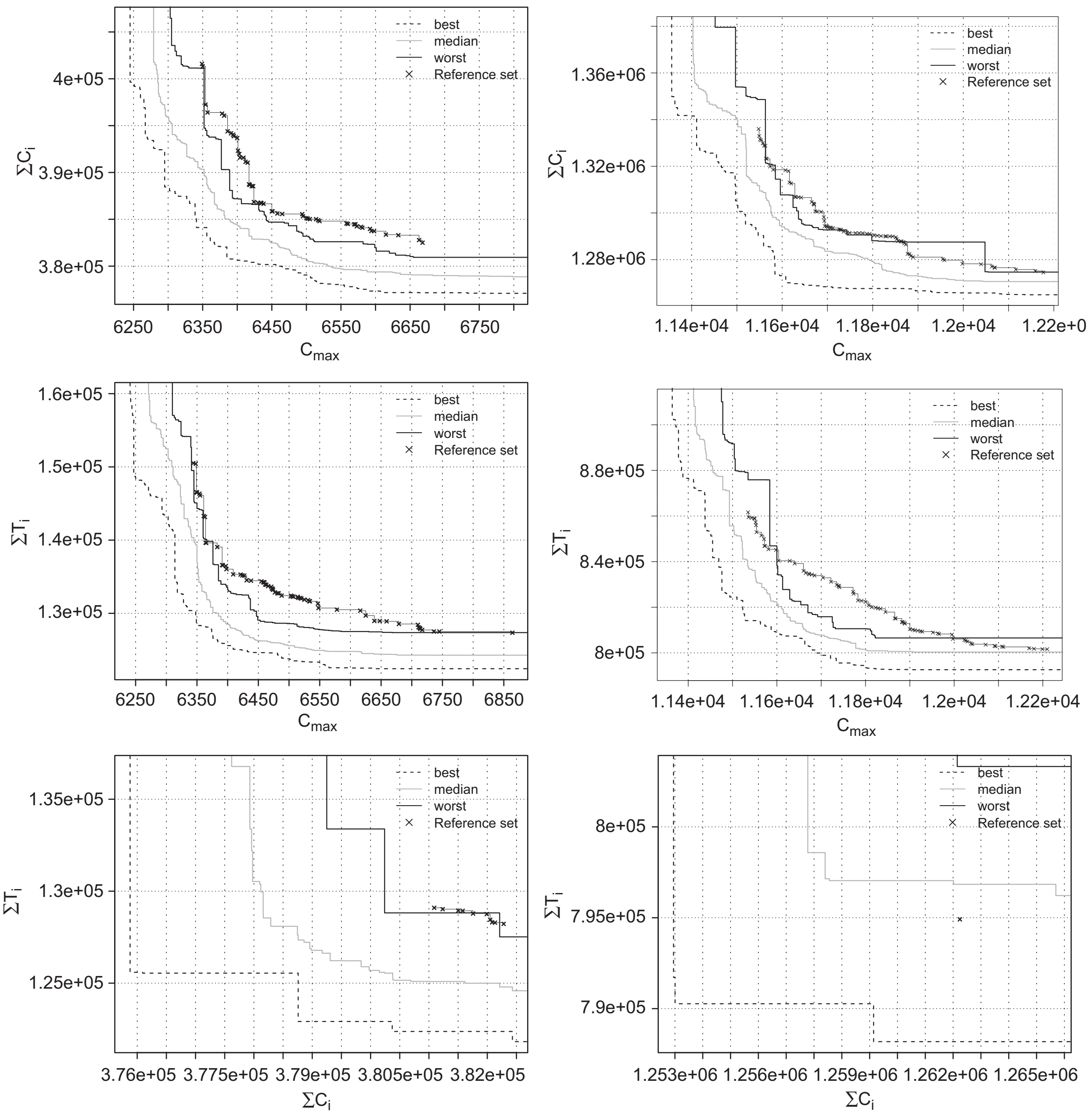

Fig. 7. Attainment surfaces of TP+PLS against the reference set for instances DD_Ta082 (100 × 20) on the left and DD_Ta102 (200 × 20) on the right, for top: *PFSP*-(C_{max}, *SFT*), middle: *PFSP*-(C_{max}, *TT*) and bottom: *PFSP*-(*SFT*, *TT*).

in implementations and computing environment, we have reimplemented these two algorithms. We describe first the implementation of these two algorithms, and we present later the results of our experimental analysis.

Multi-objective simulated annealing (*MOSA*): Varadharajan and Rajendran [12] designed MOSA for the bi-objective *PFSP*-($C_{\max}$, *SFT*) (minimization of the makespan and sum of flowtimes). Recently, Minella et al. [11] identified MOSA as the best algorithm among 23 algorithms for the three bi-objective PFSPs arising from the combinations of the objectives makespan, sum of flowtimes and total tardiness. Varadharajan and Rajendran [12] proposed two combinations of parameters for MOSA, one for shorter and another for longer runs. We use here the parameters for longer runs, with a higher value for the epoch length and a lower temperature threshold.

The core of MOSA is a standard classical simulated annealing algorithm, henceforth denoted by single-SA. It compares each new solution with the current solution according to only one of the objectives in order to accept the new solution or not. The choice of the objective is done probabilistically for each comparison. The probability to choose one objective over the other is kept constant until the temperature reaches a certain value. Then single-SA restarts by setting the temperature again to its initial value, but the probability to choose an objective over the other one is slightly changed.

From a high-level point of view, MOSA consists of two main phases. The first phase starts from a solution provided by the NEH heuristic to minimize the makespan, which is subsequently improved by three improvement schemes (named JIBIS, OSSBIS and JIBSS) that evaluate a sequence of *insertion* or *exchange* moves by considering the job following either their index or their position, and apply the improving moves. Then, single-SA is run four times with different probabilities to consider the makespan instead of the sum of flowtimes when a new solution has to be considered. These probabilities for the four runs are (1, 0.83, 0.66, 0.5). Each run of single-SA starts from the previous solution found and stops when the temperature threshold is reached. The second phase of MOSA starts from a solution provided by Rajendran's heuristic [36], which is a constructive heuristic to minimize the sum of flowtimes, and this solution is further improved by the three improvement schemes mentioned above. As in the first phase, single-SA is run four times, with the probability of choosing the sum of flowtimes objective over the makespan one being (1, 0.83, 0.66, 0.5). The acceptance criterion is similar to the one defined in Eq. (7).

MOSA was originally proposed to tackle the bi-objective *PFSP*-($C_{\max}$, *SFT*) only. To provide an initial solution for the tardiness objectives (weighted or not), we use the same heuristic (NEH+WSLACK) as our algorithm (Section 3.1). Moreover, since the stopping criterion of MOSA is a temperature threshold, we further modify the algorithm to stop after a certain computation time limit. For this purpose, we have considered two alternatives. The first alternative uses a modified cooling rate of the temperature that *approximately* reaches the temperature threshold when the computation time reaches the limit. The second alternative keeps the original cooling rate, and sets again the temperature to its initial value when it reaches the threshold (but keeping the current solution). This latter possibility allows to run the algorithm for a precise computation time, which is exactly divided among the eight runs of single-SA. We carried out some preliminary experiments to compare the quality of the outputs provided by each variant of MOSA. The quality of the non-dominated sets was roughly equivalent, and we decided to use the second variant for our comparison. The other parameter settings of MOSA are taken directly from the original publication.

Multi-objective genetic local search (*MOGLS*): MOGLS, proposed by Arroyo and Armentano [13], was the second-best algorithm for the bi-objective PFSPs studied in the review of Minella et al. [11]. MOGLS uses elitism, the *OX* crossover to recombine solutions, and the insertion operator for mutation. A partial enumeration heuristic that constructs a set of non-dominated solutions [37] provides the initial population. If this heuristic generates less non-dominated solutions than the expected number of initial solutions (i.e. the population size), then a diversification scheme for permutation problems [38] generates the remaining solutions. The original MOGLS—and, as far as we know, the implementation of Minella et al. [11]—uses the version of non-dominated sorting proposed by Deb et al. [39] in order to assign fitness to candidate solutions. However, to be as fair as possible, in our implementation of MOGLS, we use the faster version proposed by Jensen [40]. After a given number of generations, a multi-objective local search is performed on a subset of the current population, for a fixed number of iterations. This subset is selected among the non-dominated solutions of the current population using a clustering procedure based on the centroids technique [41]. A list records the non-dominated solutions already explored by the multi-objective local search, to avoid exploring them again. The local search uses a restricted insertion neighborhood, where each job is inserted in the best position among the positions closer than a given distance from the job's initial position, and this distance decreases at each iteration. The original and our implementation of this restricted insertion operator use the same speed-up as the insertion operator used in IG. The remaining parameters of MOGLS are set to the same values as in the original publication.

Comparison of TP+PLS with MOSA and MOGLS: We test our implementation of MOSA and MOGLS by extracting all non-dominated solutions they obtained across 25 independent runs each, and comparing these non-dominated sets with the reference sets provided by Minella et al. [11]. As we mentioned earlier, these reference sets were obtained from the results of 23 algorithms including the implementation of MOSA and MOGLS by Minella et al.

The non-dominated sets extracted from the results of our implementations of MOSA and MOGLS often dominate the reference sets (we provide these plots as supplementary material [29]). Since the differences in implementation language and computation environment with respect to Minella et al. [11] are small, we believe that the comparison indicates that our implementation of MOSA and MOGLS is at least as efficient as the original ones.

MOSA and MOGLS are run under the same experimental conditions (language, compiler, computers) and for the same computation time and the same number of runs (25) as TP+PLS.

We give in Table 7 the percentage of runs (computed for each instance over the 625 pairwise comparisons of the 25 runs, and averaged over the 10 instances of each size) that the output set of our TP+PLS algorithm is better in the Pareto sense (in the sense of "$\lhd$", see Section 2.2) than the output set obtained by a run of MOSA, and, conversely, the average percentage of runs that the output set of MOSA is better than TP+PLS. The same comparison is done in Table 8 between TP+PLS and MOGLS. Detailed tables with percentage values for each instance are available as supplementary material [29]. Percentages in favor of our algorithm are very strong, whereas the percentages in favor of MOSA and MOGLS are very low. A value of 0 means that MOSA (or MOGLS) is not able to produce in any run a non-dominated set better than the worst one produced by TP+PLS in any of the 25 runs of the 10 instances of a given size. The percentages in Table 7 show that for small instances of 20 jobs, MOSA and our TP+PLS algorithm are difficult to compare. The low percentages are explained by the fact that both algorithms often find the same non-dominated set, which is probably the optimal Pareto front. For these small instances, differences are not consistent across instances and combinations of objectives, and it cannot be said that any algorithm is clearly better than the other. Nevertheless, for all the remaining instances, Tables 7 and 8 show

Table 7
For each bi-objective problem, the left column shows the percentage of runs (computed over 25 runs per instance and averaged over 10 instances of the same size) in which an output set obtained by TP+PLS is better in the Pareto sense than an output set obtained by MOSA.

$n \times m$	PFSP-($C_{\max}$, SFT)		PFSP-($C_{\max}$,TT)		PFSP-($C_{\max}$, WT)		PFSP-(SFT, TT)		PFSP-(SFT, WT)	
	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA
20 × 5	4.66	5.83	6.1	1.34	14.95	0.18	10.19	26.31	0.02	20.15
20 × 10	1.87	9.2	0.07	0.26	0.02	0.06	0.19	0.63	0.03	0.07
20 × 20	0.13	1.23	1.27	1.57	1.99	2.32	3.63	5.55	4.2	10.09
50 × 5	89.49	0	84.33	0	79.22	0	98.13	0.08	33.67	0
50 × 10	72.92	0	63.17	0	63.24	0	94.07	0	20.53	0
50 × 20	75.94	0	61.11	0	63.01	0	5.79	0	14.72	0
100 × 5	84.97	0	70.5	0	67.12	0	93.66	2.54	9.72	0
100 × 10	76.94	0.05	69.86	0	37.49	0	95.38	0.58	16.84	0
100 × 20	73.17	0	63.29	0	23.81	0	97.35	0	15.31	0
200 × 10	18.04	0.16	24.5	0	4.15	0	91.77	3.72	0.02	0
200 × 20	15.16	0	37.83	0	0.25	0	78.23	6.28	1.04	0.02

The right column shows the converse values for the comparison of an output set of MOSA being better than an output set of TP+PLS.

Table 8
For each bi-objective problem, the left column shows the percentage of runs (computed over 25 runs per instance and averaged over 10 instances of the same size) in which an output set obtained by TP+PLS is better in the Pareto sense than an output set obtained by MOGLS.

$n \times m$	PFSP-($C_{\max}$, SFT)		PFSP-($C_{\max}$, TT)		PFSP-($C_{\max}$, WT)		PFSP-(SFT, TT)		PFSP-(SFT, WT)	
	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS
20 × 5	18.35	0	26.39	0	28.36	0	57.52	0.14	26.64	0
20 × 10	18.79	0	11.52	0	5.46	0	20.7	0	17.63	0
20 × 20	13.82	0	19.58	0.13	25.47	0.06	20.44	0	18.83	0
50 × 5	39.45	0	58.11	0	75.38	0	99.29	0	95.1	0
50 × 10	60.28	0	70.46	0	81.08	0	96.76	0	98.21	0
50 × 20	74.77	0	74.44	0	70.3	0	97.85	0	97.75	0
100 × 5	24.97	1.12	87.79	0	76.11	0	91	4.5	42.3	0
100 × 10	62.43	0.27	93.02	0	79.17	0	96.21	0.04	97.4	0
100 × 20	83.88	0	83.42	0	68.14	0	99.55	0	98.57	0
200 × 10	9.55	0	81.6	0	60.03	0	94.73	1.88	28.91	0
200 × 20	33.37	0	83.3	0	35.45	0	96.72	0	83.19	0

The right column shows the converse values for the comparison of an output set of MOGLS being better than an output set of TP+PLS.

the excellent results of our TP+PLS algorithm, with very high percentages in its favor, whereas the percentages in favor of MOSA and MOGLS are negligible.

Beyond the fact that TP+PLS often dominates MOSA and MOGLS (Tables 7 and 8), one may wonder how important is the difference between the sets. To answer this question, we also examine the EAF differences between the algorithms (Section 4.1). Plots in Figs. 8–10 show some examples of these differences for three different instances. These plots reveal strong differences and a large gap along the whole Pareto frontier between the region typically attained by MOSA and MOGLS and the region typically attained by TP+PLS. Hence, we can conclude that the difference between the non-dominated sets is not only very often in favor of our algorithm, but that these differences are also very strong.

All the additional plots and detailed tables (including the ones with the version of our hybrid algorithm using size-specific parameters), together with new reference sets obtained from our results are available as supplementary material [29].

Given such clear results, the usage of unary or binary performance indicators, which assess the quality of non-dominated sets that are not comparable in the Pareto sense, is superfluous. Our conclusion from this assessment is that TP+PLS is the new state of art for the bi-objective permutation flow-shop scheduling problem, for all combinations of objectives we studied.

6. Conclusion

In this paper we have detailed the steps followed in the engineering process of a multi-objective SLS algorithm for five bi-objective permutation flow-shop problems.

The high-level design of the algorithm as consisting of a combination between the TPLS [14] and PLS [15] algorithm frameworks was motivated by recent results on the experimental analysis of multi-objective SLS algorithms [22] and by very recent successful implementations of such ideas [16]. Here, we have followed a bottom-up SLS algorithm engineering process that first engineered effective SLS algorithms for each of the single-objective problems that underly the bi-objective ones, and for the weighted sum scalarization of pairs of objectives. In fact, high performing SLS algorithms for these single-objective problems are of crucial importance for the final performance of the TPLS algorithm. In a second step, we examined the main components of the PLS and TPLS algorithm frameworks, which constitute the components of the final hybrid algorithm. In this process, new algorithmic features have been proposed and exploited for each of the algorithm frameworks. In the case of TPLS, we have used a strategy that adaptively sets the weights for the scalarizations such that the algorithm adapts to the shape of the Pareto front and that the algorithm improves the anytime behavior, that is, that it tries to reach an as good as possible solution quality of the output

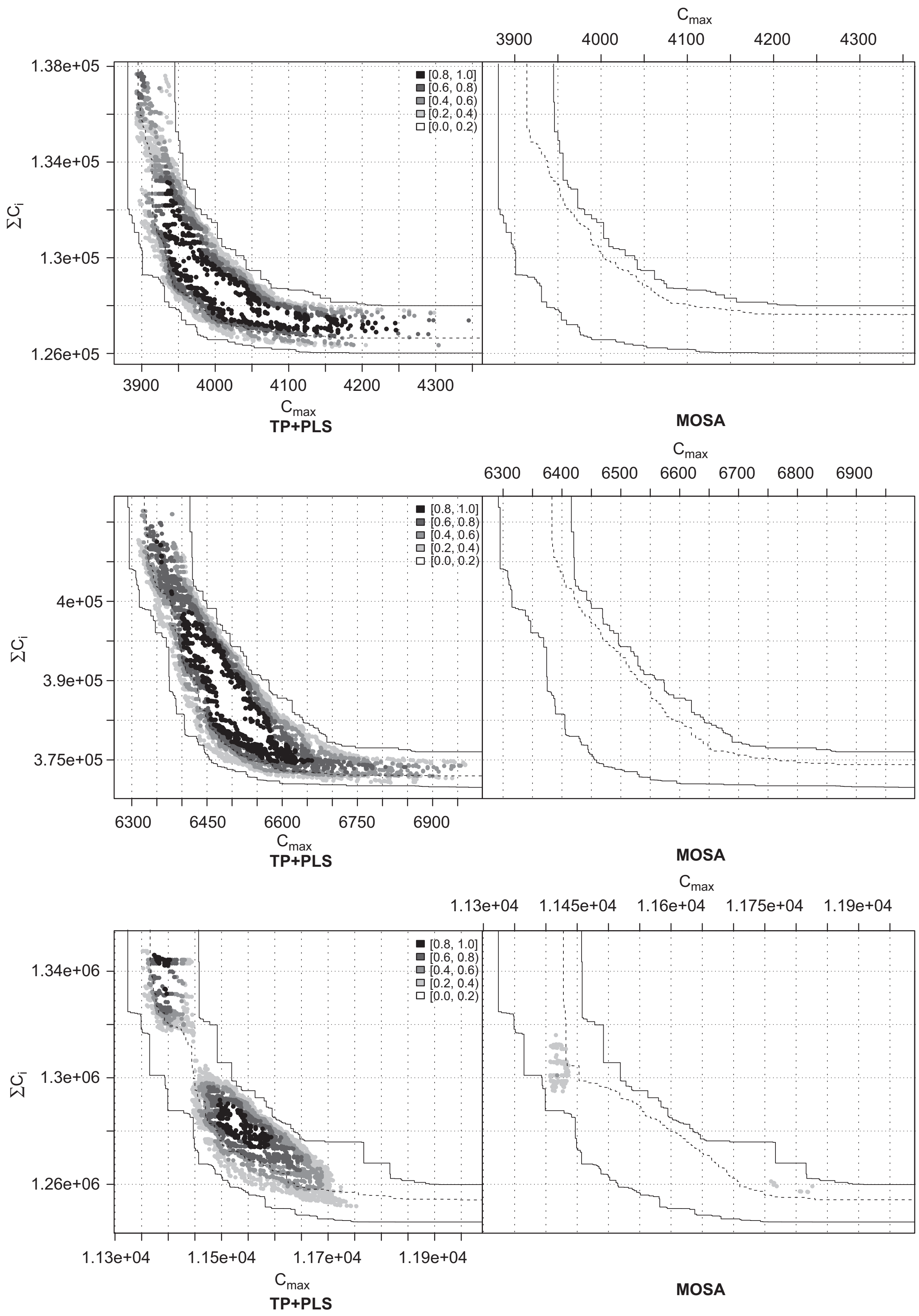

Fig. 8. EAF difference for *PFSP*-($C_{\max}$, *SFT*) on instances (from top to bottom) DD_Ta051 (50 × 20), DD_Ta081 (100 × 20), DD_Ta101 (200 × 20).

independent of the final computation time. This strategy was also found to be superior to a simple restart strategy and it was shown to be rather robust with respect to the number of scalarizations. In the study of the PLS algorithm components, we could show that PLS strongly profits from seeding it with good initial solutions, a result that also provided further motivation for the combination of the TPLS

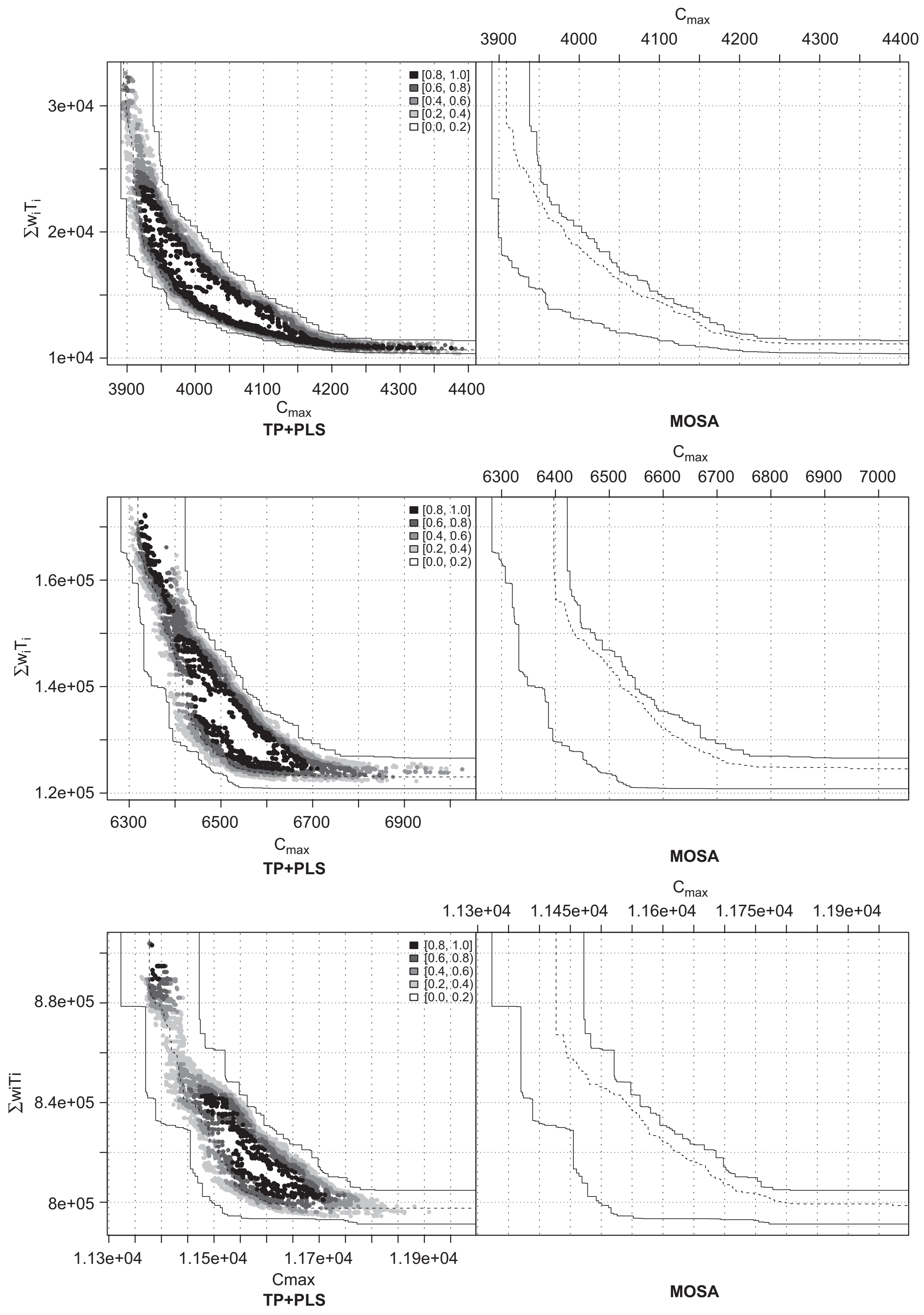

Fig. 9. EAF difference for *PFSP*-($C_{\max}$, *TT*) on instances (from top to bottom) DD_Ta051 (50 × 20), DD_Ta081 (100 × 20), DD_Ta101 (200 × 20).

algorithms with PLS. Concerning the neighborhood to be used in PLS, we found that a combination of the exchange and the insert neighborhoods was beneficial. In a final step, we examined the usefulness of combining TPLS with either the component-wise step [14,22] or a time-bounded run of PLS, the latter resulting in clearly superior solution quality.

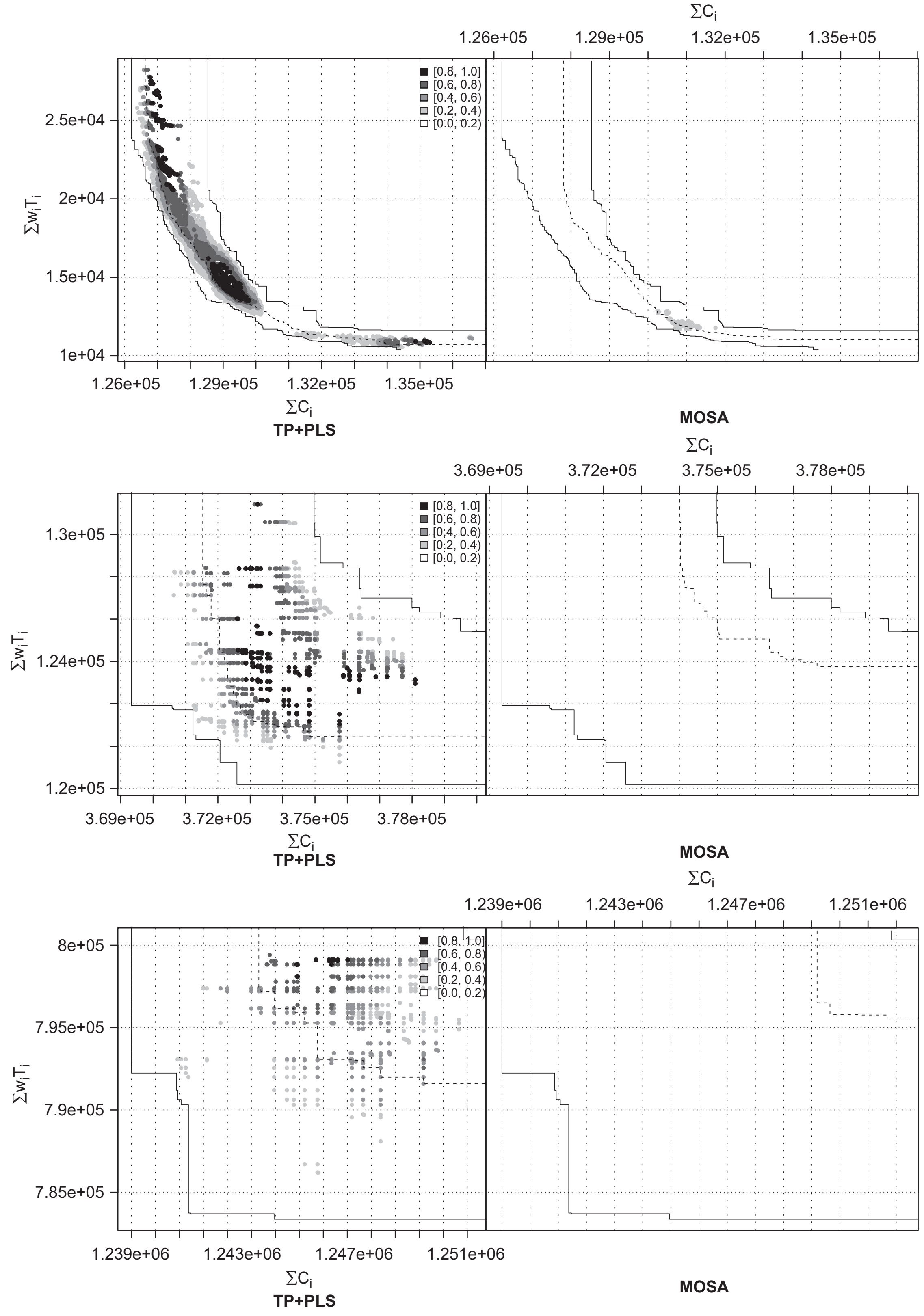

Fig. 10. EAF difference for *PFSP-*(*SFT*, *TT*) on instances (from top to bottom) DD_Ta051 (50 × 20), DD_Ta081 (100 × 20), DD_Ta101 (200 × 20).

The final TP+PLS algorithm consists of a first phase, where high-quality solutions are generated using TPLS; these provide the seed for a time-bounded version of PLS. This algorithm not only obtains better results than 23 other algorithms reported in the literature, but also a careful experimental comparison of our proposal with the two best existing algorithms for the bi-objective PFSPs shows conclusively that our hybrid TP+PLS algorithm strongly outperforms the current state-of-the-art algorithms.

Our results, in-depth experimental analyses [22] and other similar success stories recently reported in the literature [16] indicate the large potential of hybrid algorithms combining the TPLS and PLS frameworks. We believe that the engineering methodology followed here is applicable to other bi-objective problems. Moreover, we plan to extend the TPLS framework and our adaptive TPLS variant to problems with three or more objectives in order to apply our hybrid SLS algorithm to such problems.

Acknowledgments

This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium, and by the MIBISOC network, an Initial Training Network funded by the European Commission, Grant PITN-GA-2009-238819. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*".

References

[1] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2):479–94.
[2] Liao CJ, Tseng CT, Luarn P. A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research 2007;34(10):3099–111.
[3] Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 2004;155(2):426–38.
[4] Tseng LY, Lin YT. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2009;198(1):84–92.
[5] Vallada E, Ruiz R, Minella G. Minimising total tardiness in the m-machine flowshop problem: a review and evaluation of heuristics and metaheuristics. Computers & Operations Research 2008;35(4):1350–73.
[6] Du J, Leung JYT. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 1990;15(3):483–95.
[7] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1976;1:117–29.
[8] Hoos HH, Stützle T. Stochastic local search: foundations and applications. San Francisco, CA: Morgan Kaufmann Publishers; 2005.
[9] Ehrgott M, Gandibleux X. Approximative solution methods for combinatorial multicriteria optimization. TOP 2004;12(1):1–88.
[10] Paquete L, Stützle T. Stochastic local search algorithms for multiobjective combinatorial optimization: a review. In: Gonzalez TF, editor. Handbook of approximation algorithms and metaheuristics. Chapman & Hall, CRC; 2007. p. 29-1–15.
[11] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing 2008;20(3):451–71.
[12] Varadharajan TK, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research 2005;167(3):772–95.
[13] Arroyo JE, Armentano VA. Genetic local search for multi-objective flowshop scheduling problems. European Journal of Operational Research 2005;167(3):717–38.
[14] Paquete L, Stützle T. A two-phase local search for the biobjective traveling salesman problem. In: Fonseca CM, editor. Proceedings of EMO 2003. Lecture notes in computer science, vol. 2632. Heidelberg, Germany: Springer; 2003. p. 479–93.
[15] Paquete L, Chiarandini M, Stützle T. Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. In: Gandibleux X, editor. Metaheuristics for multiobjective optimisation. Lecture notes in economics and mathematical systems, vol. 535. Springer; 2004. p. 177–200.
[16] Lust T, Teghem J. Two-phase Pareto local search for the biobjective traveling salesman problem. Journal of Heuristics. 2010;16(3):475–510.
[17] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.
[18] Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 2003;7(2):117–32.
[19] Dubois-Lacoste J, López-Ibáñez M, Stützle T. Adaptive "anytime" two-phase local search. In: Learning and intelligent optimization, 4th international conference, LION 4. Lecture notes in computer science, vol. 6073. Heidelberg, Germany: Springer; 2010. p. 52–67.
[20] Paquete L, Schiavinotto T, Stützle T. On local optima in multiobjective combinatorial optimization problems. Annals of Operations Research 2007;156:83–98.
[21] Paquete L. Stochastic local search algorithms for multiobjective combinatorial optimization: methods and analysis. PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, Germany; 2005.
[22] Paquete L, Stützle T. Design and analysis of stochastic local search for the multiobjective traveling salesman problem. Computers & Operations Research 2009;36(9):2619–31.
[23] Nawaz M, Enscore Jr. E, Ham I. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. OMEGA 1983;11(1):91–5.
[24] Taillard ÉD. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 1990;47(1):65–74.
[25] Dubois-Lacoste J. A study of Pareto and two-phase local search algorithms for biobjective permutation flowshop scheduling. Master's thesis, IRIDIA, Université Libre de Bruxelles; Brussels, Belgium; 2009.
[26] Dubois-Lacoste J, López-Ibáñez M, Stützle T. Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In: Hybrid metaheuristics—6th international workshop, HM 2009. Lecture notes in computer science, vol. 5818. Heidelberg, Germany: Springer; 2009. p. 100–14.
[27] Woo H, Yim D. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers & Operations Research 1998;25(3):175–82.
[28] Balaprakash P, Birattari M, Stützle T. Improvement strategies for the F-race algorithm: sampling design and iterative refinement. In: Bartz-Beielstein T, Blesa MJ, Blum C, Naujoks B, Roli A, Rudolph G, editors. HM 2007. Lecture notes in computer science, vol. 4771. Heidelberg, Germany: Springer; 2007. p. 108–22.
[29] Dubois-Lacoste J, López-Ibáñez M, Stützle T. Supplementary material: a hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. ⟨http://iridia.ulb.ac.be/supp/IridiaSupp2010-001⟩, 2010.
[30] Fonseca CM, Paquete L, López-Ibáñez M. An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE congress on evolutionary computation. IEEE Press; 2006. p. 1157–63.
[31] Grunert da Fonseca V, Fonseca CM, Hall AO. Inferential performance assessment of stochastic optimisers and the attainment function. In: Zitzler E, Deb K, Thiele L, Coello CA, Corne D, editors. Proceedings of EMO 2001. Lecture notes in computer science, vol. 1993. Heidelberg, Germany: Springer; 2001. p. 213–25.
[32] López-Ibáñez M, Paquete L, Stützle T. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M, editors. Experimental methods for the analysis of optimization algorithms. Springer; 2010. p. 209–33.
[33] Zilberstein S. Using anytime algorithms in intelligent systems. AI Magazine 1996;17(3):73–83.
[34] Aneja YP, Nair KPK. Bicriteria transportation problem. Management Science 1979;25(1):73–8.
[35] Taillard ÉD. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[36] Rajendran C. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. International Journal of Production Economics 1993;29(1):65–73.
[37] Arroyo JE, Armentano VA. A partial enumeration heuristic for multi-objective flowshop scheduling problems. Journal of the Operational Research Society 2004;55(9):1000–7.
[38] Glover F. A template for scatter search and path relinking. In: Artificial evolution. Lecture notes in computer science. Heidelberg, Germany: Springer; 1998. p. 1–51.
[39] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 2002;6(2):181–97.
[40] Jensen MT. Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms. IEEE Transactions on Evolutionary Computation 2003;7(5):503–15.
[41] Morse JN. Reducing the size of the nondominated set: pruning by clustering. Computers & Operations Research 1980;7(1–2):55–66.