A study of Pareto and Two-Phase Local Search Algorithms for Biobjective Permutation Flowshop Scheduling

Jérémie Dubois-Lacoste

IRIDIA
Université Libre de Bruxelles
Avenue Franklin Roosevelt 50
CP 194/6
B1050 Bruxelles – Belgium
jeremie.dl@gmail.com

Abstract. In many practical problems, several conflicting criteria exist for evaluating solutions. In this thesis we study stochastic local search algorithms for biobjective permutation flow shop scheduling problems. In particular, we tackle three such problems that arise from considering all pairwise combinations of the objectives (i) makespan, (ii) the sum of the completion times of the jobs, and (iii) the weighted (and by extension, the total) tardiness of all jobs. The algorithms proposed herein are combinations of two local search methods: the two-phase local search and Pareto local search. The design of the algorithms is based on a careful experimental analysis of crucial algorithmic components of the two search methods. The final results show that the newly developed algorithms reach very high performance: The solutions obtained frequently improve upon the best nondominated solutions previously known, while requiring shorter computation times.

1 Introduction

Many problems that arise in applications such as manufacturing, telecommunications, business administration, or bioinformatics are of combinatorial nature. Combinatorial problems involve finding some type of arrangement, grouping, partitioning, or subset of a typically finite set of discrete objects. In *combinatorial optimization problems*, an objective function value is associated to each possible candidate solution and the goal is to search for a candidate solution that minimizes (or maximizes) the objective function, that is for a globally optimal solution [1].

Combinatorial problems are often very hard to solve. This fact is identified with the property of many such problems being $\mathcal{NP}$*-hard* [2]. That is, according to the current state of knowledge, in the worst case the computation time necessary for solving such problems increases exponentially with instance size. Since these problems anyway need to be tackled in practice, many algorithmic approaches towards solving them have been proposed. These can be classified as either being *exact* or *approximate* algorithms. An exact algorithm is guaranteed to find the optimal solution as well as a proof of the optimality of such a solution, in a finite amount of time. Despite quite some recent successes, exact algorithms are in many cases not competitive or flexible enough in practice. Therefore, often, the only feasible goal is to reach a reasonably good solution without any insurance of optimality. Algorithms with such a goal are often called approximate or *heuristic* algorithms. Among these, stochastic local search (SLS) algorithms are probably the most effective class of algorithms [3, 4].

Many practical problems involve the assessment of candidate solutions according to multiple objectives (or criteria). In such a multi-objective approach, the notion of optimality is different from the single-objective approach. In the single-objective case one can always say that a solution is better, as good, or worse than another one. However, in the multi-objective case the different candidate solutions are not necessarily comparable with each other. The optimality depends on the preferences of the decision maker, who may give different importance to each objective. If nothing is known about the decision maker's preferences, it is common to tackle problems in terms of Pareto optimality [5], to obtain a set of Pareto optimal solutions. A significant amount of research has recently been dedicated to applying SLS algorithms to multi-objective problems, in particular for those where optimisation is understood in Pareto terms [6, 7].

In this thesis we tackle multi-objective flow-shop scheduling problems. The flow-shop environment models problems where each job consists of a set of operations that are to be carried on machines, and the machines route, i.e. the order of the machines, is the same for each job. Flow-shops are a common production environment, for example in the chemical industry. We consider the flow-shop with the following criteria: the minimization of the makespan, i.e. the completion time of the last job, which has been the most intensively studied criterion for

this problem [8]; the minimization of the sum of completion times (also called flowtime) of all jobs, which has attracted recently a lot of efforts [9–11]; and the minimization of the weighted tardiness, a less common criterion which is however useful in practical applications. For an overview of this problem we refer to [12].
We develop effective SLS algorithms for the single-objective problems, assuming that using them as components of a higher-level general purpose algorithm framework can lead to a very effective algorithm to tackle their multi-objective counterparts. We use the Two-Phase Local Search [13] and the Pareto Local Search [14] frameworks. The algorithm for the single-objective problems is an iterated greedy algorithm [15]. This algorithm has been shown to be state-of-the-art for the makespan criterion. In a first step, we have re-implemented this algorithm in C++ for making its extension to the other objectives and also for the multi-objective problems easier. In the second step, we extend this initial algorithm to the sum of flowtime and weighted tardiness objectives and fine-tune the algorithms. In the final, third step we extend these algorithms to tackle the bi-objective versions of the flowshop problem that result from each of the pairwise combinations of the three objectives.

The main contributions of this thesis can be summarized as follows:

1. We develop new, high performing SLS algorithms for the single-objective permutation flow-shop scheduling problems (PFSPs) with the sum of flowtime and the weighted tardiness criteria. Many new best known solutions have been found with these algorithms for commonly used benchmark instances.
2. We explore the application of TPLS and PLS to a new problem and we study their behavior on three bi-objective PFSPs.
3. The multi-objective algorithm shows very good performance on the problems tested. In fact, they often find better Pareto fronts than those of a reference set that extracted the best non-dominated solutions of a set of 23 other algorithms.

This thesis is structured as follows. In Section 2 we introduce basic notions of multi-objective optimization, the problems we tackle and the algorithm frameworks with which we work. In Section 3 we describe the single-objective algorithms that are underlying components for the multi-objective part. Section 4 presents the components and results of the various experimental studies on multi-objective components. Finally we conclude in Section 5.
As an appendix we give an extended version of this thesis with additional details, plots and results [16].

2 Preliminaries

2.1 Multi-objective optimization

In MCOPs, (candidate) solutions are ranked according to an *objective function vector* $\boldsymbol{f} = (f_1, \ldots, f_d)$ with d objectives. If no *a priori* assumptions upon the decision maker's preferences can be made, the goal typically becomes to determine a set of feasible solutions that "minimize" $\boldsymbol{f}$ in the sense of Pareto optimality. If $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors in $\mathbb{R}^d$, we say that $\boldsymbol{u}$ *dominates* $\boldsymbol{v}$ ($\boldsymbol{u} \prec \boldsymbol{v}$) iff $\boldsymbol{u} \neq \boldsymbol{v}$ and $u_i \leq v_i,\ i = 1, \ldots, d$; if $\boldsymbol{u} \leq \boldsymbol{v}$, then $\boldsymbol{u}$ *weakly dominates* $\boldsymbol{v}$. We also say that $\boldsymbol{u}$ and $\boldsymbol{v}$ are *nondominated* iff $\boldsymbol{u} \nprec \boldsymbol{v}$ and $\boldsymbol{v} \nprec \boldsymbol{u}$ and are (pairwise) *non weakly dominated* if $\boldsymbol{u} \nleq \boldsymbol{v}$ and $\boldsymbol{v} \nleq \boldsymbol{u}$. For simplicity, we also say that a solution s dominates another one s' iff $\boldsymbol{f}(s) \prec \boldsymbol{f}(s')$. If no other s' exists such that $\boldsymbol{f}(s') \prec \boldsymbol{f}(s)$, the solution s is called a *Pareto optimum.* The goal in MCOPs then typically is to determine the set of all Pareto optimal solutions, which is often called the *Pareto front.* Since this task is in many cases computationally intractable, in practice the goal becomes to find an approximation to the set of Pareto optimal solutions in a given amount of time that is as good as possible. In fact, any set of mutually nondominated solutions provides such an approximation. The notion of Pareto optimality can be extended to compare sets of mutually nondominated solutions [17, 18]. In particular, we can say that one set A dominates another set B ($A \prec B$), iff every $\boldsymbol{b} \in B$ is dominated by at least one $\boldsymbol{a} \in A$.

2.2 Bi-objective permutation flowshop scheduling

In the flowshop scheduling problem (FSP) a set of n jobs $(J_1, \ldots, J_n)$ is given 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. 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, candidate solutions correspond to permutations of the jobs and the resulting problem, on which we focus here, is the permutation flowshop scheduling problem (PFSP). All processing times p_{ij} for a job J_i on a machine M_j are fixed, known in advance and nonnegative. In the following, we denote by C_i the completion time of a job J_i on machine M_m. For a given job permutation π, the makespan is the completion time of the last job in the permutation, i.e., $C_{\max} = C_{\pi(n)}$. For $m \geq 3$ this problem is $\mathcal{NP}$-hard in the strong sense [19]. In the following, we refer to this problem as *PFSP-*$C_{\max}$.

The other objectives we study are the minimization of the sum of flowtimes and the minimization of the weighted tardiness. The sum of flowtimes is defined as $\sum_{i=1}^{n} C_i$. The PFSP with this objective is strongly $\mathcal{NP}$-hard even with only two machines [19]. In the following, we refer to this problem as *PFSP-SFT*. For the weighted tardiness objective, each job has a due date d_i by which it is to be finished and a weight w_i indicating its priority. The tardiness is defined as $T_i = \max\{C_i - d_i, 0\}$ and the total weighted tardiness is given by $\sum_{i=1}^{n} w_i \cdot T_i$. This problem, denoted *PFSP-WT*, is strongly $\mathcal{NP}$-hard even for one machine.

For a review of previous work dealing with this problem the reader should refer to the extended version of this thesis [16]. A wide review is given for each criterion and for the multi-objective PFSP.
In this thesis, we tackle the three bi-objective problems that result from any of the three possible pairs of objectives. A number of algorithms have been proposed to tackle each of these specific biobjective problems, but rarely more than one possible combination of the objectives has been addressed in a single paper. The available algorithmic approaches range from simple constructive algorithms to applications of SLS methods such as evolutionary algorithms, tabu search, or simulated annealing. We refer to the very comprehensive review article of Minella et al. [12]. This review article gives a comprehensive overview of the literature on the three problems we tackle here and presents the results of an extensive experimental analysis of 23 algorithms, either specific or adapted for tackling the three biobjective PFSPs. This study shows that MOSA [20] seems to be the best performing algorithm overall. The authors also provide reference sets for all benchmark instances tackled, where each reference set contains the nondominated solutions obtained from all the 10 runs of each of the 23 algorithms (that is, of 230 trials), each run was stopped after the same maximum CPU time.

2.3 Two-phase local search and Pareto local search

In this thesis, we study SLS algorithms that represent two main classes of multi-objective SLS algorithms [7]: algorithms that follow a component-wise acceptance criterion (CWAC), and those that follow a scalarized acceptance criterion (SAC). As two paradigmatic examples of each of these classes, we use two-phase local search (TPLS) [13] and Pareto local search (PLS) [14].

Two-Phase Local Search. The first phase of the TPLS consists of using an effective single-objective algorithm to find a good solution for one objective. This solution is the initial solution for the second phase, where a sequence of scalarizations are solved by an SLS algorithm. Each scalarization transforms the multi-objective problem into a single-objective one using a weighted sum aggregation. For a given weight vector $\lambda = (\lambda_1, \lambda_2)$, the value w of a solution s with objective function vector $\boldsymbol{f}(s) = (y_1, y_2)$ is computed as:

$$w = (\lambda_1 \cdot y_1) + (\lambda_2 \cdot y_2) \quad \text{s.t. } \lambda_1, \lambda_2 \in [0, 1] \subset \mathbb{R} \text{ and } \lambda_1 + \lambda_2 = 1.$$

In TPLS, each run of the LS algorithm for solving a scalarization uses as an initial solution the best one found by the previous run of the LS algorithm. The motivation for using such a method is to exploit the effectiveness of the underlying single-objective algorithm. The pseudo-code of TPLS is given in Algorithm 1. We denote by SLS_1 the SLS algorithm to minimize the first single objective. SLS_2 is the SLS algorithm to minimize the weighted sums.

Pareto Local Search. PLS is an iterative improvement method for solving MCOPs [21, 22], which is obtained by replacing the usual acceptance criterion of iterative improvement algorithms for single-objective problems by an acceptance

Algorithm 1 Two-Phase Local Search

```
Input: A random or heuristic solution $\pi$
$\pi' := SLS_1(\pi)$;
for all weight vectors $\lambda$ do
    $\pi := \pi'$;
    $\pi' := SLS_2(\pi, \lambda)$;
    Add $\pi'$ to Archive;
end for
Filter Archive;
```

Algorithm 2 Pareto Local Search

```
Input: An initial set of solutions $\mathcal{A}$
for each $x \in \mathcal{A}$ do
    explored($s$) := FALSE;
end for
while $\exists x \in \mathcal{A}$ with explored($s$) = FALSE do
    choose randomly $x$ in $\mathcal{A}$ s.t. explored($s$) = FALSE;
    for each $x'$ in $\mathcal{N}(s)$ do
        if $x' \preceq x$ then
            AddAndFilter($\mathcal{A}, x'$);
            explored($s'$) := FALSE;
        end if
    end for
    explored($s$) := TRUE;
end while
```

criterion that uses the dominance relation. Given an initial archive of unvisited nondominated solutions, PLS iteratively applies the following steps. First, it randomly chooses an unvisited solution s from the candidate set. 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. Solutions in the archive dominated by the newly added solutions are eliminated. Once the neighborhood of s is fully explored, s is marked as visited. The algorithm stops when all solutions in the archive have been visited. The PLS algorithm is illustrated in Algorithm 2.

We also implemented the *component-wise step* (CW step) procedure. This CW step procedure has been proposed as a post-processing step of the solutions produced by TPLS. It adds nondominated solutions in the neighborhood of the solutions returned by TPLS to the archive, but it does not explore the neighborhood of these newly added solutions further. Hence, CW step may be interpreted as a specific variant of PLS with an early stopping criterion. Because of this early stopping criterion, the CW step results in worse nondominated sets than PLS. However, its main advantage compared to running a full PLS is that it typically results only in a very small additional computation time.

3 Single-objective SLS algorithms

For TPLS, the performance of the single-objective algorithm is of high importance and they actually should be state-of-the-art algorithms for the underlying single-objective problems and as good as possible for the scalarized problems resulting from the weighted sum aggregations. Motivated by these considerations, we adopted for *PFSP*-$C_{\max}$ the iterated greedy (IG) algorithm (*IG*-$C_{\max}$) by Ruiz and Stützle [15], which is a current state-of-the-art algorithm for this problem. An algorithmic outline is given in Algorithm 3. The essential idea of IG is to iterate over a construction heuristic by first destructing partially a complete solution; next, from the resulting partial solution π_R a full problem solution is reconstructed and possibly further improved by a local search algorithm. This solution is then accepted in dependence of an acceptance criterion.

Algorithm 3 Iterated Greedy

```
$\pi$ := NEH;
while termination criterion not satisfied do
    $\pi_R$ := $\mathit{Destruction}(\pi)$;
    $\pi'$ := $\mathit{Construction}(\pi_R)$;
    $\pi'$ := $\mathit{LocalSearch}(\pi')$        % optional;
    $\pi$ := $\mathit{AcceptanceCriterion}(\pi, \pi')$;
end while
```

For our algorithm development, we reimplemented the IG algorithm [15] in C++. In a nutshell, *IG*-$C_{\max}$ uses the NEH heuristic [23] for constructing the initial solution and for reconstructing full solutions in the main IG loop. A small number of d randomly chosen jobs is removed in the destruction phase, and an effective first-improvement type algorithm based on the insert-neighborhood is used in the local search. In the insert neighborhood, two solutions are neighbored if they can be obtained by removing a job from one position and inserting it in a different one. The acceptance criterion uses the Metropolis condition: A worse solution is accepted with a probability given by $\exp\left(f(\pi') - f(\pi)\right)/T$, where f is the objective function and the temperature parameter T is maintained constant throughout the run of the algorithm. Parameter values are given in Table 2.

Given the known very good performance of *IG*-$C_{\max}$, we considered using it also for the other two objectives. A main change concerns the local search implementations, since the speed-ups of Taillard [24] are not anymore applicable, which leads to a factor n increase of the local search time complexity. As a side result, it is unclear whether the same neighborhood as for the makespan criterion should be chosen. We have therefore considered also two other neighborhoods: (i) the swap neighborhood, where two solutions are neighbored if they can be obtained by swapping the position of two adjacent jobs; and (ii) the exchange neighborhood, where two solutions are neighbored if they can be obtained by exchanging the position of two jobs. We considered only restricted versions of

insert and exchange neighborhoods, where the possible insertion and exchange moves of only one job are considered.
Other changes concern the formula for the definition of the temperature parameter for the acceptance criterion. This is rather straightforward for the *PFSP-SFT*, which can be done by adapting slightly the way the temperature is defined. For *PFSP-WT* no input data-driven setting as for the other two objectives could be obtained due to large variation of the objective values. Therefore, the temperature parameter is defined relating it to a given target percentage deviation from the current solution. Another important adaptation for the *PFSP-WT* has been to find a good constructive heuristic for the initial solution of the IG. Indeed the original NEH heuristic is not adapted to this criterion. The reason is simple: up to a certain length of partial solution during the construction phase, the sum of the tardiness stays zero. In this case, only the last few jobs will be really inserted to the best position. In the NEH heuristic these jobs are the ones with the smallest sum of processing time, without consideration of their due dates or priorities. The NEH heuristic is not anymore helpfull in its original version. Therefore we have decided to study several heuristics with the aim of selecting the best one.
As the weighted tardiness criterion is not very studied, we have adapted some heuristics originally proposed for the total tardiness criterion, and tested other possibilites. This results in a comparison of 16 differents heuristics that have been tested. Actually, among these heuristics eight are relatively simply compared to NEH, in particular four of them simply sort the jobs following a given formula, that is so called dispatching rules. The eight more complex heuristics are an adaptation of the NEH heuristic seeded by an ordering given by the sort. Obviously this results in a different computation time between the dispatching rules and the NEH adaptations. However, we focus here on the quality of the constructed solution, considering that even the adaptations of the original NEH are fast enough. We give here the formulas corresponding to each heuristic. The arrow denotes if the jobs are sorted following an increasing ($\uparrow$) or a decreasing ($\downarrow$) order. We denote as $C_j(i)$ the completion time of the job j when inserting at the end of partial solution at iteration i, i.e. in position i. The following dispatching rules have been tested to compute the "interest value" of a given job j. During iteration i, the job with the lowest ($\uparrow$) or the largest ($\downarrow$) "interest value" is chosen among the remaining jobs, and inserted in position i.

LWDD (Latest Weigthed Due Dates):	$w_j.d_j \downarrow$
EWDD (Earliest WDD:	$w_j.d_j \uparrow$
LWDDP (LWDD with Proc. times):	$w_j.d_j/\sum_{i=1}^{m} p_{ij} \downarrow$
EWDDP (EWDD with Proc. times):	$w_j.d_j/\sum_{i=1}^{m} p_{ij} \uparrow$
MWDD (Modified WDD):	$w_j.max\{d_j, C_j(s)\} \uparrow$
MWDDP (MWDD with Proc. times):	$w_j.max\{d_j, C_j(s)/\}\sum_{i=1}^{m} p_{ij} \uparrow$
WSLACK (Weighted SLACK heuristic):	$w_j \cdot (d_j - C_j(s)) \uparrow$
WSLACKP (WSLACK with Proc. times):	$w_j \cdot (d_j - C_j(s))/\sum_{i=1}^{m} p_{ij} \uparrow$

The eight others heuristics denoted nehLWDD, nehEWDD, etc., are an adaptation of the original NEH heuristic taking the results of the above heuristic as initial order, instead of taking the jobs sorted by the decreasing sum of their processing times.
For each instance (we generated our own benchmark set), and for each heuristic, we computed the relative difference over the minimum found by one of the 16 heuristics. Then, for each heuristics we compute the mean of its relative difference over all instances. This allow to order these heuristics, as shown in Table 1. Thanks to this experiment, we have decided to use the heuristic nehWSLACK as the constructive heuristic for the *PFSP-WT*. To the best of our knowledges, this heuristic has never been used before. Note that this sorting is only relevant when NEH constructs the initial solution; in the main loop of IG the jobs are considered in random order.

We tuned the IG algorithms for *PFSP-SFT* and *PFSP-WT* using iterated F-Race [25]. The final configurations retained from this tuning phase are given in Table 2. The lines *IG*-($C_{\max}$, *SFT*) and *IG*-($\cdot$, *WT*) concern the scalarized problems where the weights are different from one and zero for the indicated objectives. The parameter for these algorithms have been chosen because they seem good for each criterion.
A closer examination of the performance of the resulting single-objective algorithms (not reported here) showed that for the sum of flowtime the final IG algorithm is competitive to current state-of-the-art algorithms as of 2009; for the total tardiness objective the performance is also very good and very close to state-of-the-art; in fact we could improve with the IG algorithms a large fraction (in each case more than 50%) of the best known solutions of available benchmark sets.

Table 1. Indicated are the name of the heuristics and the average relative difference to the minimum value found so far. nehWSLACK performs by far the best.

1	nehWSLACK	1.6
2	nehWSLACKP	3.4
3	nehEWDD	14.5
4	nehEWDDP	15.0
5	nehLWDDP	15.9
6	nehLWDD	16.2
7	nehMWDD	23.3
8	nehMWDDP	23.9
9	WSLACK	29.2
10	WSALCKP	30.3
11	LWDDP	43.5
12	LWDD	46.4
13	EWDD	61.5
14	EWDDP	62.8
15	MWDD	63.7
16	MWDDP	64.7

Table 2. Adaptation of IG to tackle each objective, and the scalarized problems of each combination of criteria. A (↓) denotes a decreasing order, and a (↑) denotes an increasing order. Tp is a parameter of the formula for Temperature. Settings for *IG-*$C_{\max}$ follow [15]. For *IG-SFT*, the formula of Temperature is the same as *IG-*$C_{\max}$ but multiplied by n. For *IG-WT*, the initial order for NEH is given by the well-known SLACK heuristic [26] augmented with priorities w_i. Parameter d is the number of jobs removed in the destruction phase. Insert-T refers to a full insert iterative improvement using the speed-ups of Taillard [24]; Swap to a full iterative improvement execution using the swap neighborhood and Ins. to the insertion search for one job. For details see the text.

Algorithm	Init. order for NEH	Temperature	Tp	d	LS
IG-$C_{\max}$	$\sum_{j=1}^{m} p_{ij}$ (↓)	$Tp \cdot \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10}$	0.4	4	Insert-T
IG-SFT	$\sum_{j=1}^{m} p_{ij}$ (↓)	$Tp \cdot \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{m \cdot 10}$	0.5	5	Swap
IG-WT	$w_i \cdot (d_i - C_i(s))$ (↑)	$\frac{100}{Tp \cdot f(s)}$	0.7	4	Swap+Ins.
IG-$(C_{\max}, SFT)$	$\sum_{j=1}^{m} p_{ij}$ (↓)	$\frac{100}{Tp \cdot f(s)}$	0.5	5	Swap
IG-$(\cdot, WT)$	$w_i \cdot (d_i - C_i(s))$ (↑)	$\frac{100}{Tp \cdot f(s)}$	0.5	5	Swap

4 Multi-objective SLS algorithms

In what follows, we first study main algorithm components of the PLS and TPLS algorithms and then present a comparison of a final hybrid SLS algorithm to reference sets of the best solutions found so far for a number of benchmark instances.

Benchmark set. We used the benchmark from Minella et al. [12]. This benchmark consists of the benchmark set of Taillard [27], which has been augmented with due dates and priorities. In order to avoid over-tuning, we performed the algorithm component analysis on 20 additional instances of size `50x20` and `100x20`, which have been generated following the procedure used by Minella et al. [12].

Correlations. We want to measure the correlation between the objectives. We do so by generating 10000 random solutions for 10 large instances (200x20). We found the following results for the linear correlation between each pair of objectives.

Correlation between C_{max} and $\sum C_j$:	0.655
Correlation between C_{max} and $\sum w_j \cdot T_j$:	0.404
Correlation between $\sum C_j$ and $\sum w_j \cdot T_j$:	0.625

We applied the Spearman's rank order test at $\alpha = 0.05$ to check wether the correlations are statistically signifcantly different from zero. In all cases, the null hypothesis was rejected.

Performance assessment. Results are analyzed by graphically examining the attainment surfaces of a single algorithm and differences between the empirical attainment functions (EAF) of pairs of algorithms. The EAF of an algorithm provides the probability, estimated from several runs, 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 [28]. An attainment surface delimits the region of the objective space attained by an algorithm with a certain minimum probability. In particular, the worst attainment surface delimits the region of the objective space always attained by an algorithm, whereas the best attainment surface delimits the region attained with the minimum non-zero probability. Similarly, the median attainment surface delimits the region of the objective space attained by half of the runs of the algorithm. Examining the attainment surfaces allows to assess the likely location of the output of an algorithm. On the other hand, examining the differences between the EAFs of two algorithms allows to identify regions of the objective space where one algorithm performs better than another. Given a pair of algorithms, the differences in favor of each algorithm are plotted side-by-side and the magnitude of the difference is encoded in gray levels.

We give in Figure 1 two plots which show the EAF for two different algorithms. In Figure 2 we give an example of EAF difference for these algorithms. López-Ibáñez et al. [29] provide a more detailed explanation of these graphical techniques and tools for their application.

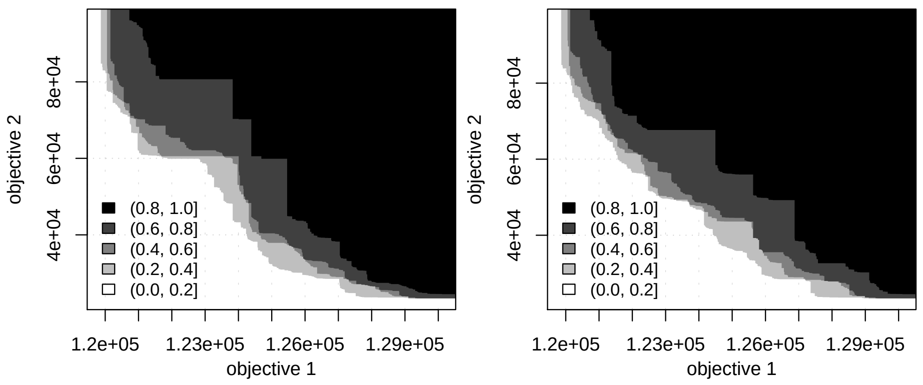

Fig. 1. An example of the EAF for two different algorithms.

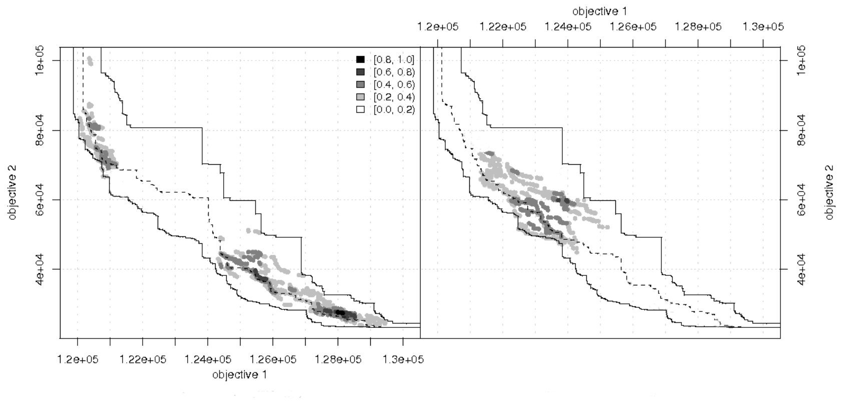

Fig. 2. The difference of the two previous EAFs. On the left, the dark region represents where the first algorithm performs better that the second one. On the right it is the opposite. The darker is the color, the higher is the difference. One can also see the worst, median and best region attained by each algorithm. We can see that the algorithm on the left performs sligthly better on the sides of the Pareto front approximation, relatively to the other one. Conversely, the algorithm on the right is slightly better on the middle.

4.1 Analysis of PLS components

Seeding. As a first experiment, we analyzed the computation time required and the final quality of the nondominated sets obtained by PLS when PLS is seeded with solutions of different quality. We test seeding PLS with: (i) one randomly generated solution, (ii) two good solutions (one for each single objective) obtained by the NEH heuristics (see Table 2), and (iii) two solutions obtained by IG for each objective after 10 000 iterations. Figure 3 gives representative examples of nondominated sets obtained by PLS for each kind of seed and indicates the initial seeding solutions of NEH and IG (the random initial solution is not shown because it is far out of the plotted objective ranges). The best nondominated sets, in terms of a wider range of the Pareto front and higher quality solutions, are obtained when using the IG seeds. Generally, seeding PLS by good and very good solutions gives clear advantages in terms of the quality of the nondominated sets obtained; the advantage is strongest for the bi-objective problem that considers makespan and total flowtime and slightly less for the other two. We further examined the computation time required by PLS in dependence of the initial seed in Table 3. Our conclusion is that seeding PLS with very good initial solutions does not have a strong effect in computation time. Given the strong improvement on solution quality, seeding PLS with a set of nondominated solutions obtained by TPLS is pertinent.

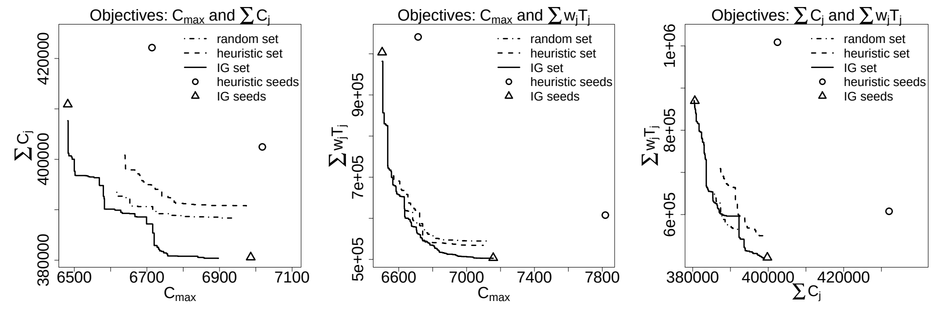

Fig. 3. Nondominated sets obtained by PLS using different quality of seeds. The solution randomly generated is outside the range shown.

Table 3. Average computation time (and standard deviation) of PLS for different types of seeds.

		random		heuristic		IG	
Objectives	Instance Size	avg.	sd.	avg.	sd.	avg.	sd.
$(C_{\max}, \sum C_i)$	`50x20`	8.85	2.05	6.23	2.48	4.56	0.38
	`100x20`	177.40	27.60	142.23	29.79	162.14	26.09
$(C_{\max}, \sum w_i T_i)$	`50x20`	31.61	6.84	33.85	7.46	24.02	3.84
	`100x20`	641.96	215.55	767.23	299.33	626.48	114.08
$(\sum C_i, \sum w_i T_i)$	`50x20`	26.72	3.02	28.17	2.62	23.70	3.33
	`100x20`	742.42	157.10	807.75	121.70	895.23	176.29

Neighborhood operator. We experiment with PLS variants using three different neighborhoods: (i) the insertion, (ii) the exchange, and (iii) the exchange plus insertion neighborhoods. The latter simply checks for all moves in the exchange and insertion neighborhood of each solution. We measured the computation time of PLS with each underlying operator for each combination of objectives (Table 4). The computation time of the combined exchange and insertion neighborhood is slightly more than the sum of the computation times for the exchange and insertion neighborhoods. For comparing the quality of the results, we examine the EAF differences of 10 independent runs. Figure 4 gives two representative examples. Typically, the exchange and insertion neighborhoods lead to better performance in different regions of the Pareto front (top plot), and both of them are consistently outperformed by the combined exchange and insertion neighborhood (bottom plot).

Table 4. Average computation time (and standard deviation) of PLS for different neighborhood operators.

		exchange		insertion		ex. + ins.	
Objectives	Instance Size	avg.	sd.	avg.	sd.	avg.	sd.
$(C_{\max}, \sum C_i)$	`50x20`	2.21	0.35	1.57	0.44	4.84	1.06
	`100x20`	77.56	19.44	70.91	12.8	157.64	30.26
$(C_{\max}, \sum w_i T_i)$	`50x20`	12.94	3.11	10.11	1.75	23.03	4.09
	`100x20`	314.63	69.08	251.84	49.33	611.6	115.02
$(\sum C_i, \sum w_i T_i)$	`50x20`	14.24	3.79	9.51	1.8	23.72	3.87
	`100x20`	492.91	102.59	239.04	101.47	872.32	262.21

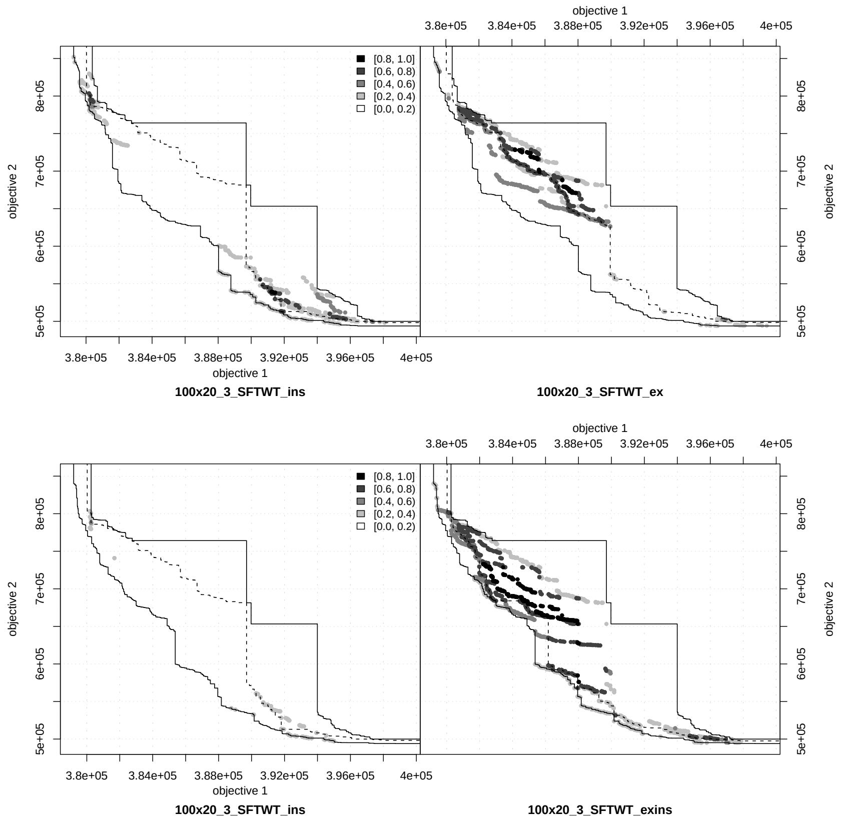

Fig. 4. EAF differences for (*top*) *insertion* vs. *exchange* and (*bottom*) exchange vs. exchange and insertion. The combination of objectives is $\sum C_i$ and $\sum w_i T_i$.

4.2 Analysis of TPLS components

Search strategy. The search strategy determines how the information is shared between each scalarization.
The most simple strategy is to not share any information between each underlying resolution. When a new scalarization is solve, we simply start from a random solution or with solution from a constructive heuristic (we tested this latter option). We denote this strategy RESTART, the pseudo-code is given in Algorithm 4. An other strategy is to transfer some information from a previous run of the SLS algorithm to the next one, i.e. to start the next run from the previously found solution. We denote this strategy 2PHASE as it is the original one from [13]. The pseudo-code has been given in Section 2. Figure 5 illustrates these two strategies.

Algorithm 4 RESTART strategy

```
for all weight vectors $\lambda$ do
    $\pi$ := random or heuristic solution;
    $\pi'$ := $SLS(\pi, \lambda)$;
    Add $\pi'$ in Archive;
end for
```

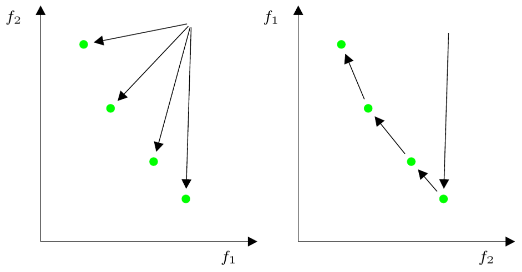

Fig. 5. Search strategy. The RESTART strategy is illustrated on the left, the 2PHASE strategy on the right.

The two stategies perform roughly similar. However, each time a difference appears, it is in advantage of 2PHASE. The search space structure for the PFSP seems to be such that use a good solution as a seed increase the chance for each scalarization to yield a better solution. We can expect that for the PFSP, two good solutions close to each other in the objective space are also close to each other in the search space. As 2PHASE seems to perform most of the time at least as good as RESTART, and sometimes slightly better, the strategy should be used for the first phase. For all further experiments, we use the 2PHASE strategy.

Number of scalarizations and number of iterations. In TPLS, each scalarization is computed using a different weight vector. In this paper, we use a regular sequence of weight vectors from $\boldsymbol{\lambda} = (1, 0)$ to $\boldsymbol{\lambda} = (0, 1)$. If N_{scalar} is the number of scalarizations, the successive scalarizations are defined by weight vectors $\boldsymbol{\lambda}_i = (1 - (i/N_{\text{scalar}}), i/N_{\text{scalar}})$, $i = 0, \ldots, N_{\text{scalar}}$.

For a fixed computation time, there is a tradeoff between the number of scalarizations to be used and the number of iterations to be given for each of the invocations of the single-objective SLS algorithm. In fact, the number of scalarizations (N_{scalar}) determines how many scalarized problems will be solved (intuitively, the more the better approximations to the Pareto front may be obtained), while the number of iterations (N_{iter}) of IG determines decisively how good the final IG solution will be. Here, we examine the trade-off between the settings of these two parameters by testing all 9 combinations of the following settings: $N_{\text{scalar}} = \{10, 31, 100\}$ and $N_{\text{iter}} = \{100, 1\,000, 10\,000\}$.

We first studied the impact of increasing either N_{scalar} or N_{iter} for a fixed setting of the other parameter. Although clearly improvements are obtained by increasing each of the two parameters, there are significant differences. While for the number of scalarizations some type of limiting behavior without strong improvements was observed when going from 31 to 100 scalarizations (while improvements from 10 to 31 were considerable), increasing the number of iterations of IG alone seems always to incur significant improvements.

Next, we compare settings that require roughly the same computation time. Figure 6 compares a configuration of TPLS using $N_{\text{scalar}} = 100$ and $N_{\text{iter}} = 1000$ against other configuration using $N_{\text{scalar}} = 10$ and $N_{\text{iter}} = 10000$. Results are shown for two of the three combinations of objective functions. (The results are representative for other instances.) As illustrated by the plots, there is no clear winner in this case. A larger number of iterations typically produces better solutions in the extremes of the Pareto front. On the other hand, a larger number of scalarizations allows to find trade-off solutions that are not found with a smaller number of scalarizations. Given these results, among settings that require roughly the same computation time, there is no single combination of settings that produces the best results overall among all objectives and instances.

Double TPLS. We denote as Double TPLS (DTPLS) the following strategy. First, the scalarizations go sequentially from one objective to the other one, as in the usual TPLS. Then, another sequence of scalarizations is performed starting from the second objective back to the first one. To introduce more variability, the weight vectors used in the first TPLS pass are alternated with the weight vectors used for the second TPLS pass. We compared this DTPLS strategy with the simple TPLS using 30 scalarizations of 1000 iterations. Although we found on several instances strong differences, these differences were not consistent in favor of advantages of DTPLS over TPLS or vice versa. This gives some evidence that the two strategies do not behave the same, but we left it for further research to investigate which instances features may explain the observed differences.

Weight setting strategy. In order to define the sequence of weight vectors that define each scalarization, several strategies can be used. Here we will focus on the REGULAR strategy, and the ANYTIME strategy. We call REGULAR the strategy which uses a regular distribution of the weight vectors, from $\lambda = (1, 0)$ to $\lambda = (0, 1)$. In this case, the different vectors are sequential, going from one objective to the other one, and the stopping criterion for the first phase is, as seen above, the number of scalarization. We call ANYTIME an other strategy that we have implemented. This strategy can be useful in real practical applications, since the algorithm can be stopped at a given time unknown at the beginning. In this case, the sequence of weight vectors is not sequencially distributed from one objective to the other one. The first one (after the two initial solutions have been found), is the vector $(0.5, 0.5)$. Then the two next vectors will be $(0.25, 0.75)$ and $(0.75, 0.25)$, and so on. This strategy can be seen as an iterative weight setting which at each iteration, define a new weight vector be-

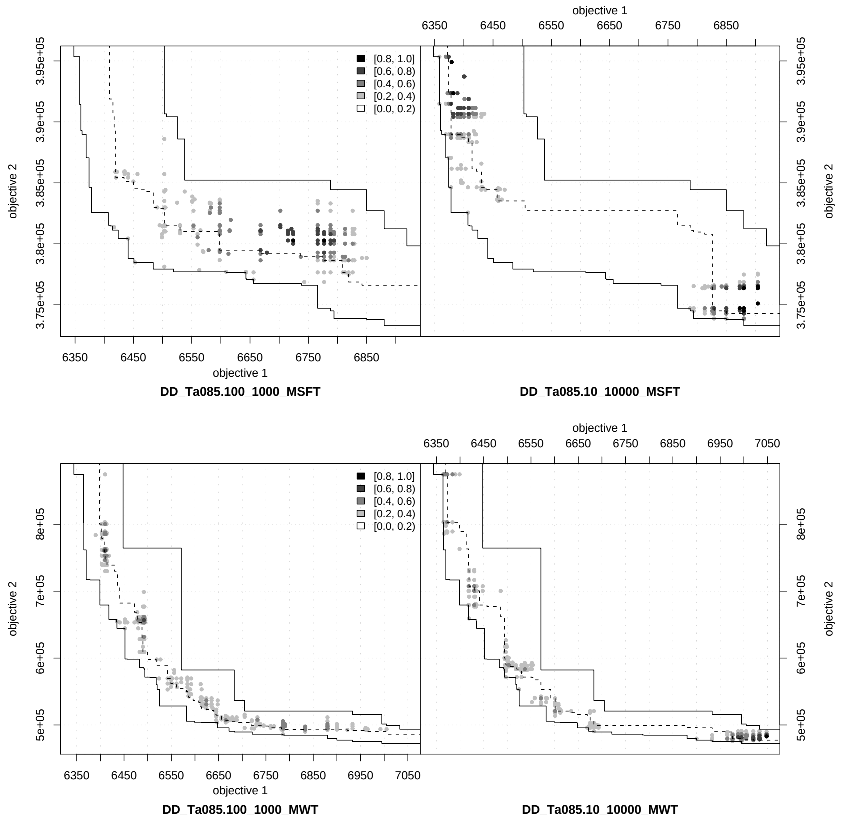

Fig. 6. EAF differences between $N_{\mathrm{scalar}} = 100$ and $N_{\mathrm{iter}} = 1000$, versus $N_{\mathrm{scalar}} = 10$ and $N_{\mathrm{iter}} = 10000$ for two combinations of objectives: $C_{\max}$ and $\sum C_i$ (top) and $C_{\max}$ and $\sum w_i T_i$ (bottom).

tween each pairwise of adjacent vectors previously found. We present in Figure 7 an example of the beginning of such a strategy.
Moreover, we have implemented an interesting feature. In order to choose an initial solution for the next scalarization, we cannot use anymore the solution found by the previous solution. Indeed, the weight vectors can be quite far from the previous one. Therefore we propose a new way to define these initial solutions. All previously found solutions are keep in memory, along with the weight vectors from which they were derived. This allow to know, when defining a new scalarization, the solutions found by the two nearest weight vectors at each side of the new one. Then the aggregate function value is computed for these two solutions, and the best one with respect to the current weight vector is kept as initial solution. Technically, the solutions in memory are sorted following their corresonding weight vectors to allow an efficient search among the data structure.

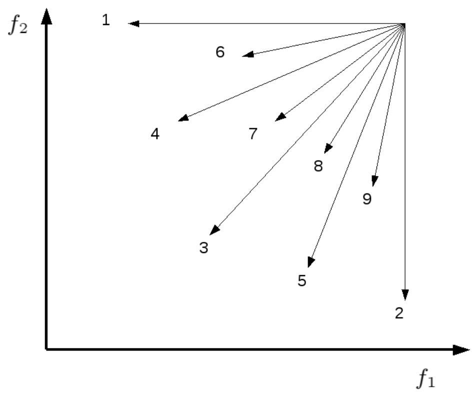

Fig. 7. Anytime strategy: The number at the end of each arrow gives the order in which the aggregate problems are solve.

In Figure 8, we give some results for the evaluation of the weight setting strategies. We used 10 runs for each instance, and 31 scalarizations of 1000 iterations. We chose 31 in order to make a fair comparison, because with this number the ANYTIME strategy can finish entirely a "level of depth". These two strategies seem very similar in term of performance. We can conclude that our implementation of the ANYTIME strategy, with the particular way for choose the initial solution, perform as good as the regular one (if it is stopped at the end of a "level of depth"). Using this strategy can be very useful in practice, for example in all practical dynamic applications, when the computation time devoted to the resolution of a problem is not necessarily fixed in advance.

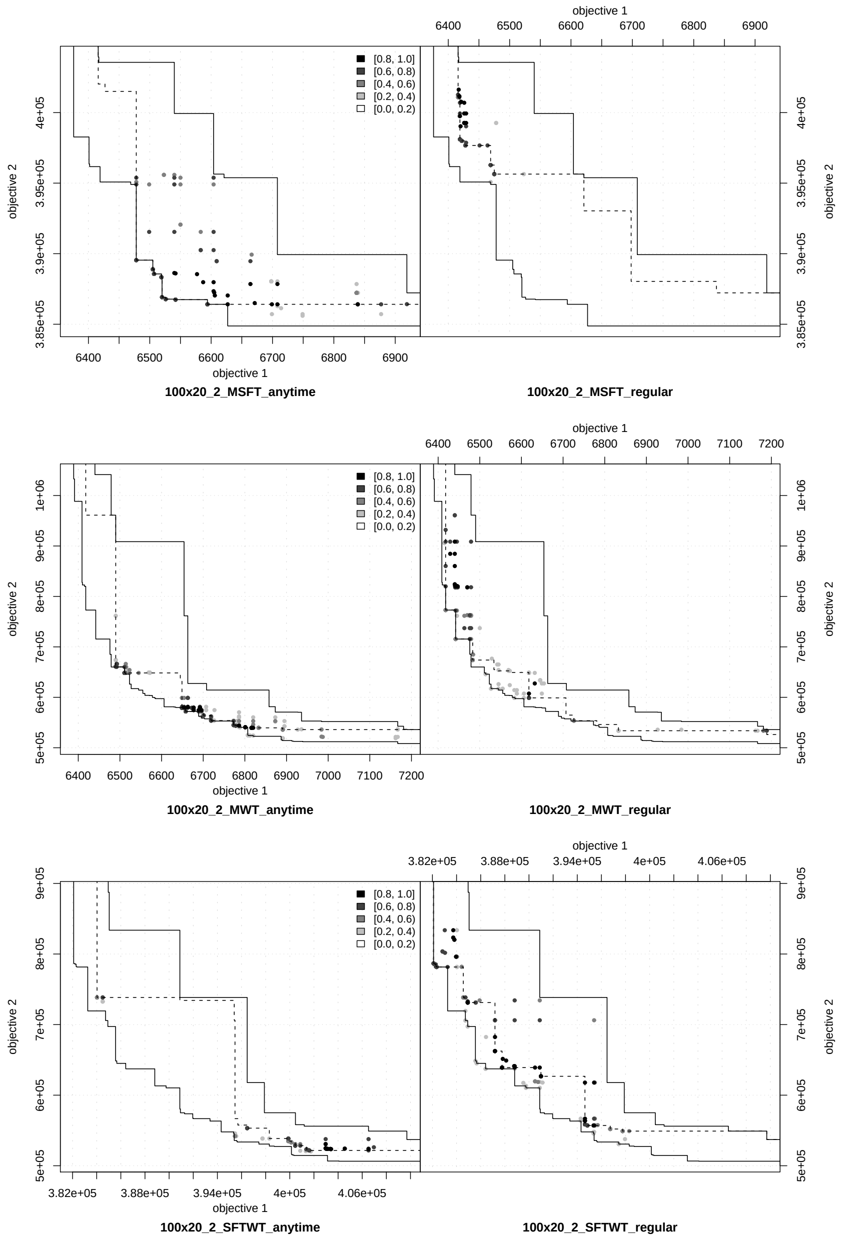

Fig. 8. Evaluation of the two different weight setting strategies on one instance for the three combinations of criteria. Performances are similar.

4.3 TPLS + PLS vs. TPLS + CW step.

As a final step, we compare the performance tradeoffs incurred by either running PLS or the CW step to the archive of solutions returned by TPLS. For all instances, we generated 10 initial sets of solutions by running a simple TPLS with 30 scalarizations of 1000 iterations each. Then, we independently apply to these initial sets the CW step and PLS, both using the exchange and the insertion neighborhoods. In other words, each method starts from the same set of initial solutions in order to reduce variance.

Table 5 gives the additional computation time that is incurred by PLS and the CW step after TPLS has finished for each of the three combinations of objectives. The results clearly show that the CW step incurs only a minor overhead with respect to TPLS, while PLS requires considerably larger times, especially on larger instances. Moreover, the times required to terminate PLS are much lower than when seeding it with only two very good solutions (compare with Table 3). With respect to solution quality, Figure 9 compares the three approaches: TPLS versus TPLS+CW step (top), and TPLS+CW step versus TPLS+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 computation time is very limited, the CW step provides significantly better results at almost no computational cost; however, if enough time is available, PLS improves much further than the sole application of the CW step.

Table 5. Average computation time and standard deviation for CW step and PLS.

	Instance	CW-step		PLS	
Objectives	Size	avg.	sd.	avg.	sd.
$(C_{\max}, \sum C_i)$	50x20	0.20	0.02	2.40	0.75
	100x20	1.56	0.40	75.04	32.53
$(C_{\max}, \sum w_i T_i)$	50x20	0.37	0.03	7.43	2.11
	100x20	2.51	0.42	202.71	50.51
$(\sum C_i, \sum w_i T_i)$	50x20	0.34	0.04	8.99	1.71
	100x20	2.75	0.34	373.15	87.44

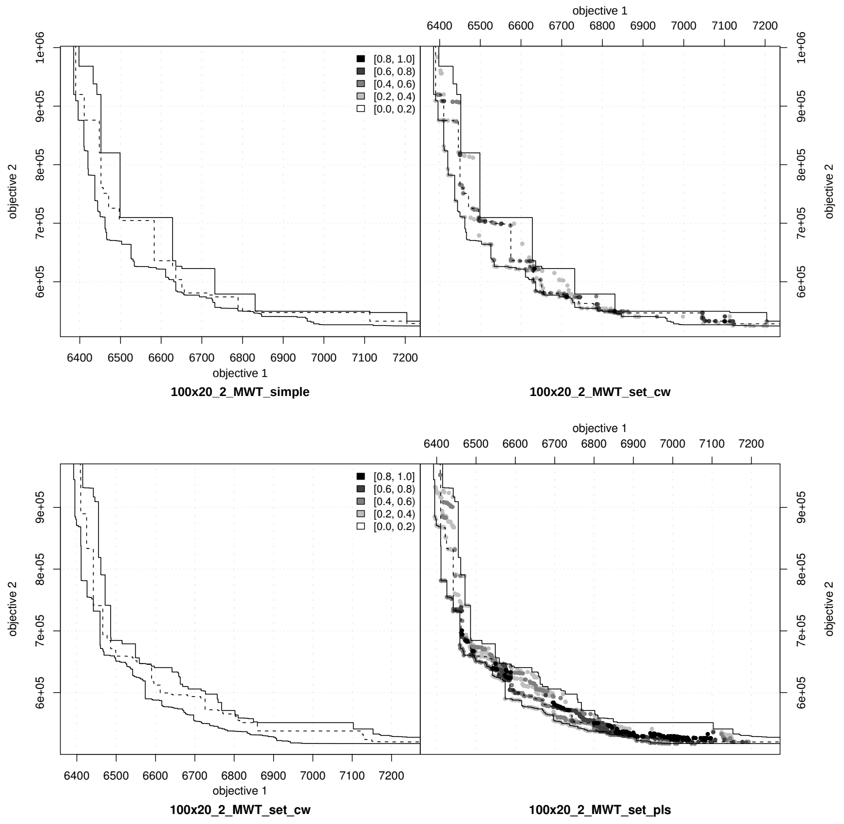

Fig. 9. EAF differences between (top) simple TPLS vs. TPLS + CW-step, and (bottom) TPLS + CW-step vs. TPLS + PLS. Objectives are $C_{\max}$ and $\sum w_i T_i$.

4.4 Comparison to existing algorithms

In order to compare our algorithm with existing work, we used the benchmark of Minella et al. [12], which is Taillard's benchmark set augmented with due dates. In their review, the authors compare 23 heuristics and metaheuristics using bi-objective combinations of makespan ($C_{\max}$), sum of flowtime ($\sum C_i$), and total tardiness ($\sum T_i$). They also provide the best-known nondominated solutions across all 23 algorithms and all 10 runs for each of the algorithms. We use these sets as reference sets to compare with our algorithm. As the reference sets are given for the total tardiness criterion, we slightly modified our algorithm by setting all the priorities to one ($w_i = 1$).

In particular, we compare our results with the reference sets given for a computation time of 200 seconds and corresponding to instances from `ta081` to `ta090` (size `100x20`). These reference sets were obtained on an Intel Dual Core E6600 CPU running at 2.4 Ghz. By comparison, our algorithms were run on a

Intel Xeon E5410 CPU running at 2.33 Ghz with 6MB of cache size, under Cluster Rocks Linux. Both machines result in approximately similar speed according to bogomips information; however, to be conservative, we decided to round down the quality of our results by using only 150 CPU seconds. For our algorithms, we used the following parameter settings. The two extreme solutions are generated by running IG for 10 seconds each. Then TPLS starts from the solution obtained for the first objective and runs 14 scalarizations of 5 seconds each. Finally, we apply PLS with the exchange plus insertion neighborhood and stop it after 60 CPU seconds. We repeat each run 10 times with different random seeds.

For each instance, we compare the best, median and worst attainment surfaces obtained by our algorithm with the corresponding reference set. Figure 10 shows results for the three objective combinations. In most cases, the median attainment surface of our algorithm is very close (and often dominates) the reference set obtained by 10 runs of 23 algorithms, each run using 200 CPU seconds (that is, the overall computation time is more than 12 hours of CPU time per instance). Moreover, the current state-of-the-art algorithms for these problems are among these algorithms. Therefore, we conclude that our algorithm is clearly competitive and probably superior to the current state-of-the-art for these problems. All the results and plots are available online at:
`http://iridia.ulb.ac.be/~jdubois`

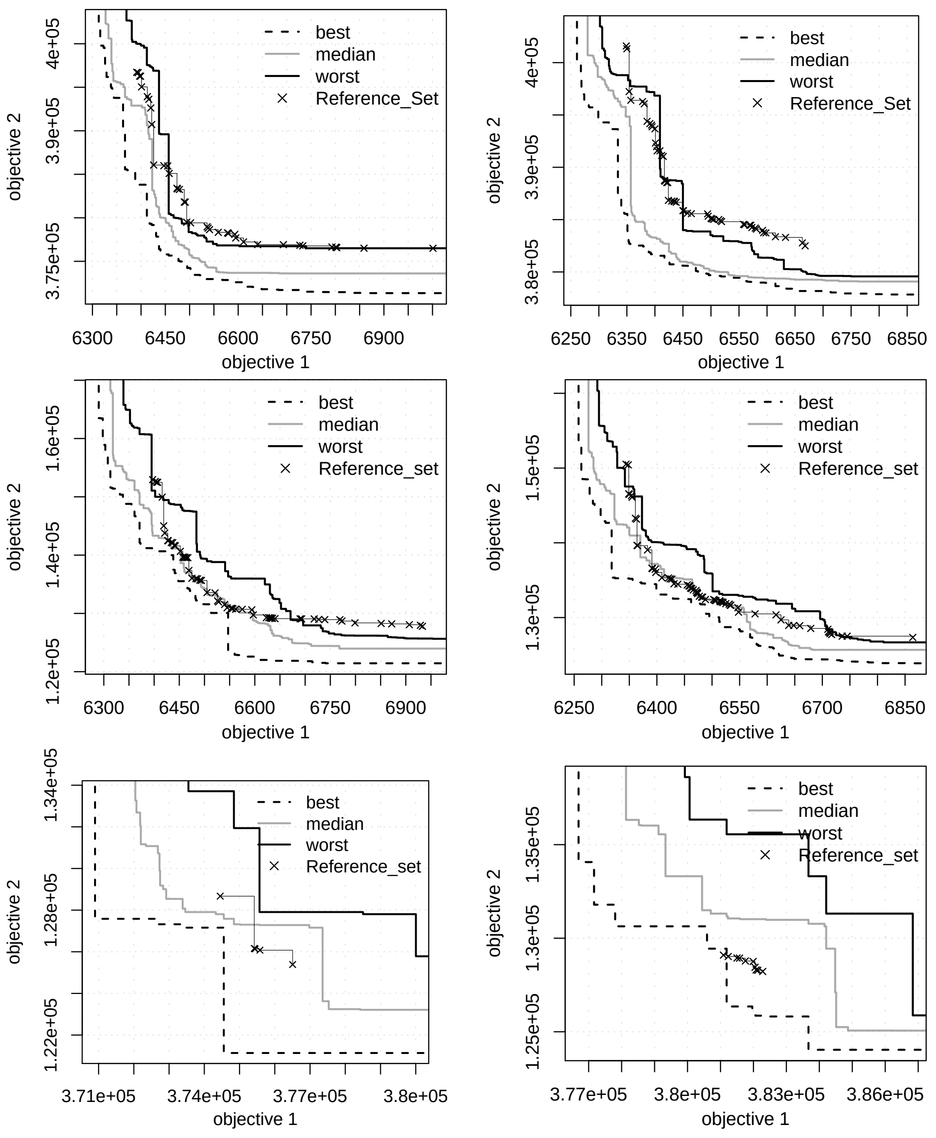

Fig. 10. Comparison of our algorithm against reference set for objectives $C_{\max}$ and $\sum C_i$ (top), $C_{\max}$ and $\sum T_i$ (middle), and on $\sum C_i$ and $\sum T_i$ on instances `DD_Ta081` (left) and `DD_Ta082` (right).

5 Conclusion

In this master thesis, we tackled the permutation flow-shop problem (PFSP), a well-known combinatorial optimization problem in scheduling. This problem has been studied with a multi-objective approach. The objectives we have considered are: (i) the minimization of the makespan, (ii) the minimization of the sum of the flowtime and (iii) the minimization of the weighted tardiness. To tackle these problems effectively, we have based our work on the following hypothesis:

An effective way of tackling multi-objective combinatorial optimization problems is to extend very effective algorithms for the underlying single-objective problems by using them as components for a higher-level multi-objective approach. Such approaches may result in state-of-the-art performance.

Following this hypothesis, we have first implemented a very effective single-objective algorithm for each criterion. For the makespan criterion, the Iterated Greedy (IG) of Ruiz and Stützle [15] is known to be the state-of-the-art approach and we based also our algorithms for the other objectives on this same method. We first re-implemented the IG for the makespan criterion, and checked that our implementation yields the same results as the original one. Then, for the single-objective step of this thesis, our main contributions are:

1. We adapted the IG algorithm to tackle the minimization of the sum of flowtime. We compared our algorithm with the most recent ones from the literature, and showed that it is very competitive. In a production mode of the algorithm we improved the best known solutions.
2. We adapted also the IG method to tackle the minimization of the weighted tardiness. The comparison with results from the literature showed that for this criterion too, our single-objective algorithm is very effective. We improved the best known solutions for many known benchmark instances for the total tardiness.

Once we had very effective algorithms to minimize each objective, we then approached the multi-objective problems derived from each pairwise combination of objectives. We used Two-Phase Local Search (TPLS) as a framework to tackle multi-objective problems relying on our single-objective algorithms as underlying components. This method tackles multi-objective problems by solving a sequence of scalarizations. In order to possibly yield better results, we implemented also the Pareto Local Search (PLS). For the multi-objective part of this thesis, our contributions are:

1. We implemented the TPLS framework as a set of components. A lot of experiments have been carried out to study the behavior of these components. We studied two different search strategies, by allowing information sharing between different scalarizations, and showed that for our problems the best strategy is to start a scalarization from the resulting solution of the previous one. We studied different weight setting strategies, showing that a procedure

that can be stopped at anytime can yield similar results to standard TPLS strategy. Such a procedure can be useful in practical applications if no fixed computation times are allocated. We experimented also the *double* TPLS and compare it to te standard TPLS, showing that the quality of the results can be considered in general as roughly similar.

2. We implemented PLS, and studied its behavior. We showed that the results can be significantly improved by seeding PLS with good solutions. We made a comparison of different underlying neighborhood operators, showing that the exchange operator performs better than the insertion operator for a similar amount of time, and that the best results are obtained with a combination of the two operators.
3. We applied a component-wise step (CW step) to the result set obtained by TPLS, and we compared the results with the application of PLS starting from the same set. PLS performed much better, but it required also a much higher additional computation time than CW step and it incurs a high variability of computation times.
4. From the insights obtained through the experimental analysis of the impact of the components of TPLS and PLS, we derived a final SLS algorithm that we applied to a large set of bi-objective PFSPs, comparing them to reference sets that have been extracted from the results of 10 runs of 23 algorithms (including all state-of-the-art algorithms for the respective bi-objective problems) on the same benchmark instances. We showed that this final SLS algorithm typically outperforms these reference sets and, hence, we are safe to assume that it is a new state-of-the-art algorithm for all the bi-objective PFSPs tackled in this thesis.

In additional work, we have implemented several components that are not presented in this thesis, in particular, other weight setting strategies or path-relinking operators. Studies on the behavior of these components are ongoing work. Further work could be also done by studying finely the interactions between components, and by studying the time allocation between PLS and TPLS.

In our work we have shown that our underlying working hypothesis, which is explicitely stated above, leads to very effective SLS algorithms for the bi-objective PFSPs we have studied here. In this sense, we add further evidence to previous work on similar types of algorithms [13, 30], which have recently received significant attention in multi-objective combinatorial optimization. In fact, judging by the overall results of our work here, we are convinced that the increased exploitation of the best single-objective algorithms as underlying components, will be instrumental to further boost the performance of SLS algorithms for multi-objective combinatorial optimization.

References

1. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization – Algorithms and Complexity. Prentice Hall, Englewood Cliffs, NJ, USA (1982)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness. Freeman, San Francisco, CA, USA (1979)
3. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA, USA (2005)
4. Yannakakis, M.: Computational complexity. In Aarts, E.H.L., Lenstra, J.K., eds.: Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester, UK (1997) 19–55
5. Ehrgott, M.: Multicriteria Optimization. Volume 491 of Lecture Notes in Economics and Mathematical Systems. Springer Verlag, Heidelberg, Germany (2000)
6. Ehrgott, M., Gandibleux, X.: Approximative solution methods for combinatorial multicriteria optimization. TOP **12**(1) (2004) 1–90
7. Paquete, L., Stützle, T.: Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In Gonzalez, T.F., ed.: Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC (2007) 29–1—29–15
8. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research **165** (2005) 479–494
9. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research **155**(2) (2004) 426 – 438
10. Liao, C.J., Tseng, C.T., Luarn, P.: A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research **34**(10) (2007) 3099 – 3111
11. Tsenga, L.Y., Lin, Y.T.: A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research **198**(1) (2009) 84 – 92
12. Minella, G., Ruiz, R., Ciavotta, M.: A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing **20**(3) (2008) 451–471
13. Paquete, L., Stützle, T.: A two-phase local search for the biobjective traveling salesman problem. In Fonseca, C.M., et al., eds.: Proceedings of the Evolutionary Multi-criterion Optimization (EMO 2003). Volume 2632 of LNCS., Springer Verlag (2003) 479–493
14. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In Gandibleux, X., et al., eds.: Metaheuristics for Multiobjective Optimisation. Volume 535 of LNEMS. Springer Verlag (2004) 177–200
15. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research **177**(3) (2007) 2033–2049
16. 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 (ULB), Brussels, Belgium (2009)
17. Hansen, M.P., Jaszkiewicz, A.: Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark (1998)

18. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation **7**(2) (2003) 117–132
19. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1** (1976) 117–129
20. Varadharajan, T.K., 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 **167**(3) (2005) 772 – 795
21. Paquete, L., Chiarandini, M., Stützle, T.: A study of local optima in the multiobjective traveling salesman problem. Technical Report AIDA-02-07, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt (2002)
22. Paquete, L., Stützle, T.: Clusters of non-dominated solutions in multiobjective combinatorial optimization. In Barichard, V., et al., eds.: Post-Conference Proceedings of MOPGP 2006. LNEMS. Springer (2008) In press.
23. Nawaz, M., Jr., E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA **11**(1) (1983) 91–95
24. Taillard, É.D.: Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research **47**(1) (1990) 65–74
25. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., et al., eds.: 4th International Workshop on Hybrid Metaheuristics, Proceedings, HM 2007. Volume 4771 of LNCS., Springer Verlag (2007) 108–122
26. 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 **35**(4) (2008) 1350 – 1373
27. Taillard, É.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2) (1993) 278–285
28. Grunert da Fonseca, V., Fonseca, C.M., Hall, A.: Inferential performance assessment of stochastic optimizers and the attainment function. In Zitzler, E., Deb, K., Thiele, L., Coello, C.C., Corne, D., eds.: Evolutionary Multi-criterion Optimization (EMO 2001). Volume 1993 of Lecture Notes in Computer Science., Springer Verlag, Berlin, Germany (2001) 213–225
29. López-Ibáñez, M., Paquete, L., Stützle, T.: Exploratory analysis of stochastic local search algorithms in biobjective optimization. Technical Report TR/IRIDIA/2009-015, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (May 2009)
30. Lust, T., Teghem, J.: Two-phase pareto local search for the biobjective traveling salesman problem. Journal of Heuristics (2009) To appear.