Université Libre de Bruxelles
Facultée des sciences appliquées
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

UNIVERSITÉ LIBRE DE BRUXELLES ULB

UNIVERSITÉ D'EUROPE

Effective Stochastic Local Search Algorithms For Bi-Objective Permutation Flowshop Scheduling

Jérémie Dubois-Lacoste

Supervisors:

D^r Thomas Stützle

D^r Mauro Birattari

IRIDIA
Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Rapport d'avancement de recherches
présenté pour la Formation Doctorale
en sciences de l'Ingénieur.

– Année académique 2009/2010 –

Abstract

In this report, we study stochastic local search (SLS) algorithms for bi-objective optimization problems. In particular, we study multi-objective algorithms that belong to two different classes of approaches, those that use a scalarized acceptance criterion (SAC) and those that use a component-wise acceptance criterion (CWAC). As a representative example of a SAC algorithm, we study Two-Phase Local Search (TPLS). We propose an improved version of TPLS that has very good *anytime* properties: it achieves an as good approximation to the Pareto front as possible independent of the computation time, and thus, can be stopped at any moment while providing very good results, improving in this way over original TPLS versions.
We believe that hybridization of algorithms belonging to the SAC and CWAC classes can result in more performing algorithms. We design such a hybrid algorithm that combines our improved version of TPLS and Pareto Local Search (PLS), a representative example of a CWAC algorithm. We use as test problems bi-objective Permutation Flowshop Scheduling Problems (PFSP), a widely studied class of scheduling problems. We carefully study each component of TPLS and PLS and their effect on the final solution quality. By comparing the hybrid algorithm with the previous state-of-the-art, we show without ambiguity that our algorithm greatly improves upon the best ones previously known for bi-objective PFSPs.

Résumé

Dans ce rapport, nous étudions des algorithmes de recherche locale stochastique pour des problèmes d'optimisation bi-objectifs. En particulier, nous étudions des algorithmes multi-objectifs qui appartiennent à deux approches différentes, ceux qui utilisent un critère d'acceptation basé sur des scalarizations (SAC), et ceux qui utilisent un critère d'acceptation basé sur la dominance (CWAC). Comme exemple des algorithmes SAC, nous étudions Two-Phase Local Search (TPLS). Nous proposons une version améliorée de TPLS qui fournit une approximation de la frontière Pareto aussi bonne que possible, et ce indépendamment du temps, et ainsi surpasse les versions originales de TPLS.
Nous pensons que l'hybridisation d'algorithmes appartenant aux classes SAC et CWAC peut produire de meilleurs algorithmes. Nous concevons un tel algorithme hybride qui combine notre version améliorée de TPLS et Pareto Local Search (PLS), un exemple d'algorithme CWAC. Nous utilisons comme problèmes des Permutation Flowshop Scheduling Problems (PFSP) bi-objectifs, classe de problèmes d'ordonnancement largement étudié. Nous étudions en détail les composants de TPLS et PLS, leurs effets sur la qualité finale. En comparant notre algorithme avec l'état de l'art, nous montrons qu'il surpasse nettement les meilleurs connus pour le PFSPs bi-objectif.

Contents

Chapter 1

Introduction

Optimisation problems arise in nearly all fields of our society. Combinatorial problems, in particular, are ubiquitous in applications such as bioinformatics, manufacturing, telecommunications, business administration. They involve finding some type of arrangement, grouping, partitioning, or subset of a typically finite set of discrete objects. 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 [26].

Combinatorial problems are often very hard to solve. This fact is identified with the property of many such problems being $\mathcal{NP}$-hard [16]. 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 optimality, 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.

Many optimisation problems encountered in real life can be evaluated according to various, often conflicting objectives. Therefore, since a few decades multi-objective optimisation attracts considerable efforts and has become a very active field of research. In a multi-objective approach, the notion of optimality is different from the single-objective approach, where a solution can be always compared to another one, being better, equal or worse. However, in the multi-objective case the different candidate solutions

are not necessarily comparable with each other. Optimality depends on the preferences of the decision maker, who may give different importance to each objective. If nothing is known about such preferences, it is useful to tackle problems in terms of Pareto optimality [13], to obtain a whole set of Pareto-optimal solutions, none of them being dominated by another one. A significant amount of recent research deals with the application of SLS algorithms to multi-objective problems. In particular, without a priori information about decision maker's preferences, the goal using SLS algorithms becomes to find a good approximation of the Pareto-optimal front [14, 30].

The available approaches for tackling multi-objective problems with SLS algorithms can roughly be classified as following two main search paradigms [30]: algorithms that follow a component-wise acceptance criterion (CWAC search model), and those that follow a scalarized acceptance criterion (SAC search model), that is, they solve aggregations of the multi-objective problem into a single-objective one.

In this work, we study algorithms that belong to these two different paradigms. These algorithms are two-phase local search (TPLS) [29] that belongs to the SAC search model algorithms, and Pareto Local Search (PLS) [28] that belongs to the CWAC search model algorithms. TPLS uses a single-objective algorithm to solve several *scalarizations*, that is, aggregations of the multiple objective functions into a single-objective problem. On the other hand, PLS exhaustively searches for non-dominated solutions in the neighbourhood of an initial set.

This report is structured as follows. First, in Chapter 2 we introduce some basics on multi-objective optimisation, we present the permutation flow-shop scheduling problem (PFSP) which we used to support our studies, and we describe the TPLS and PLS frameworks. In Chapter 3 we focus on the TPLS framework and we propose some enhancements in order to avoid some drawbacks of the original one. Indeed, to provide good results, the original version requires to specify the computation time in advance, and, hence, as poor anytime behaviour [39]. We show that the improved version of TPLS we propose can reach the same solution quality as the original one but having additionally a much improved anytime behaviour. In Chapter 4 we study each components of TPLS and PLS and their influence on the solution quality. We design a hybrid algorithm (TP+PLS) for the bi-objective PFSP by combining the two frameworks, and we evaluate its efficiency by comparing it with state-of-the-art algorithms that we reimplemented. Results show that our algorithm clearly improves upon the previous state-of-the-art for all combinations of objectives we studied. Finally, in Chapter 5, we point out some directions for future work and we conclude.

Chapter 2

Basics & Existing Work

In this chapter, we introduce some basics that are required in the rest of the report. We first give in Section 2.1 formal definitions on multi-objective optimisation. We then describe in section 2.2 the flow-shop problem, on which we apply the algorithms studied in this report. We present the two frameworks studied, TPLS and PLS, in Sections 2.3 and 2.4, respectively. Finally, we see in Section 2.5 that comparing the output of multi-objective algorithms is a difficult actively studied problem, in particular we present the tools that we use in the rest of this report.

2.1 Multi-objective optimisation

Since a few decades, multi-objective optimisation is attracting more and more effort from researchers and has become a very active field in optimisation. The main driving force for this development is that many problems encountered in real life require an evaluation according to various, often conflicting objectives. In multi-objective combinatorial optimisation problems (MCOPs), candidate solutions are evaluated according to an *objective function vector* $\vec{f} = (f_1, \ldots, f_d)$ with d 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 $\mathbb{R}^d$ (d 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 $u_i \leq v_i,\ i = 1, \ldots, d$. We say that $\vec{u}$ *weakly dominates* $\vec{v}$ ($\vec{u} \preceq \vec{v}$) iff $u_i \leq v_i,\ i = 1, \ldots, d$. We say that $\vec{u}$ and $\vec{v}$ are

mutually *non-dominated* iff $\vec{u} \not\prec \vec{v}$ and $\vec{v} \not\prec \vec{u}$. We also say that $\vec{u}$ and $\vec{v}$ are *non weakly dominated* if $\vec{u} \not\preceq \vec{v}$ and $\vec{v} \not\preceq \vec{u}$. 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 [14], 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 [41]. 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 $A \not\lhd B$, $B \not\lhd A$, and $A \neq B$, then A and B are *incomparable* ($A \parallel B$), and none of the two sets is preferred over the other according only to Pareto optimality.

2.2 Permutation flow-shop scheduling problem

The flow-shop scheduling problem (FSP) is one of the most widely studied scheduling problems since it was proposed in 1954 by Johnson [20]. 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. Hence, a candidate solution is then a permutation of the jobs and 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 i^{th} 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 recursive formula:

$$C_{\pi_0 j} = 0 \qquad j = 1, \ldots, m, \tag{2.1}$$

$$C_{\pi_i 0} = 0 \qquad i = 1, \ldots, n, \tag{2.2}$$

$$C_{\pi_i j} = \max\{C_{\pi_{i-1} j}, C_{\pi_i j-1}\} + p_{ij}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m. \tag{2.3}$$

For simplicity, in the remainder of the report, 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 minimisation as *PFSP-*$C_{\max}$. Despite the fact that the *PFSP-*$C_{\max}$ can be solved in polynomial time for two machines, for $m \geq 3$ the problem is $\mathcal{NP}$-hard in the strong sense [16].

The other objectives studied in this work are the minimisation of the *sum of flowtimes* and the minimisation of the *weighted tardiness*. Because all jobs are assumed to be available at time 0, 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 minimisation as *PFSP-SFT*. It is strongly $\mathcal{NP}$-hard even for only two machines [16]. Each job can have an associated due date d_i. The tardiness of a jobs J_i is defined as $T_i = \max\{C_i - d_i, 0\}$ and the total weighted tardiness is given by $\sum_{i=1}^{n} w_i \cdot T_i$, where w_i is a weight assigned to job J_i. This problem is denoted as *PFSP-WT*. If all weights w_i are set to the same value, we say that the objective is the *total tardiness*, and we denote it as *PFSP-TT*. The *PFSP-TT* and the *PFSP-WT* are strongly $\mathcal{NP}$-hard even for a single machine [8]. Most of the studies on the tardiness objective, including the review of Minella et al. [23], focus only on the non-weighted variant. However, in this work, we consider both, the *PFSP-TT* and the *PFSP-WT* problems.

We tackle the bi-objective PFSP problems that result from five possible pairs of objectives (we do not consider the combination of the total and weighted tardiness). We denote these five bi-objective problems as follows. *PFSP-*($C_{\max}$, *SFT*) denotes the minimisation of the makespan and the sum of flowtimes, *PFSP-*($C_{\max}$, *TT*) and *PFSP-*($C_{\max}$, *WT*) denote the minimisation of the makespan and, the total tardiness and weighted tardiness, respectively; *PFSP-*(*SFT*, *TT*) and *PFSP-*(*SFT*, *WT*) denote the minimisation 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. [23] 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) [37] as the best performing algorithm for all combinations of objectives. They also point out a multi-objective genetic local search (MOGLS) [3] as the highest performing alternative. Given their conclusions, these two multi-objective algorithms are currently the state-of-the-art for the bi-objective PFSPs tackled in this report.

Algorithm 1 Two-Phase Local Search

1: $\pi_1 := \mathsf{SLS}_1()$
2: $\pi_2 := \mathsf{SLS}_2()$
3: Add π_1, π_2 to Archive
4: **if** *1to2* **then** $\pi' := \pi_1$ **else** $\pi' := \pi_2$
5: **for** each weight λ **do**
6: $\quad \pi' := \mathsf{SLS}_\Sigma(\pi', \lambda)$
7: $\quad$ Add π' to Archive
8: **end for**
9: Filter(Archive)
10: **Output:** Archive

2.3 Two-Phase and Double Two-Phase Local Search

TPLS [29] is a general algorithmic framework for multi-objective optimisation 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. In this report, we focus on bi-objective problems. Hence, 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

$$f_\lambda(s) = \lambda \cdot f_1(s) + (1-\lambda) \cdot f_2(s). \tag{2.4}$$

The motivation for solving a scalarized problem is to exploit the effectiveness of single-objective algorithms. 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 tries to exploit the connectedness of solutions, that is, solutions that are close to each other in the solution space are expected to be close in the objective space. There are two main TPLS strategies in the literature:

Single direction (*1to2* or *2to1*). The simplest way to define a sequence of scalarizations is to use a regular sequence of weight vectors from either the first objective to the second or vice-versa. We call these alternatives *1to2* or *2to1*, depending on the direction followed. For example, the successive scalarizations in *1to2* are defined by the weights $\lambda_i = 1 - ((i-1)/(N_{\text{scalar}} - 1))$, $i = 1, \ldots, N_{\text{scalar}}$, where N_{scalar} is the number of scalarizations. (For simplicity, we henceforth denote weight vectors by their first component.) In *2to1* the sequence would be reversed. There are two drawbacks of this strategy. First, the direction chosen gives a clear advantage to the starting objective, that is, the Pareto front approximation will be better on the starting side. Second, one needs to know in advance the computation time that is available, in order to define appropriately the number

of scalarizations and the time spent on each scalarization. Algorithm 1 gives the pseudo-code of the single direction TPLS. We denote by SLS_1 and SLS_2 the SLS algorithms to minimize the first and the second single objectives, respectively. SLS_Σ is the SLS algorithm to minimize the scalarized problem. In our implementation, we first generate a very good solution for each objective because we have good algorithms for each single-objective problem, but we only use one of them as a starting seed for further scalarizations.

Double strategy. We denote as Double TPLS (D-TPLS) the strategy that first goes sequentially from one objective to the other one, as in the usual TPLS. Then, another sequence of scalarizations is generated starting from the second objective back to the first one. This is, in fact, a combination of *1to2* and *2to1*, where half of the scalarizations are defined sequentially from one objective to the other, and the other half in the other direction. This approach, proposed also by Paquete and Stützle [29], tries to avoid giving advantage to the starting objective. To introduce more variability, in our D-TPLS implementation, the weights used in the first TPLS pass are alternated with the weights used for the second TPLS pass. D-TPLS still requires to define the number of weights, and hence, the computation time, in advance.

2.4 Pareto local search

PLS [28] is an iterative improvement method for solving MCOPs that can be seen as an extension of iterative improvement algorithms for single-objective problems to the multi-objective context. 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 neighbourhood of s is fully explored and all neighbours that are not weakly dominated by s or by any solution in the archive are added to the archive (lines 7 to 12). Solutions in the archive dominated by the newly added solutions are removed (procedure Filter on line 13). Once the neighbourhood of s has been fully explored, s is marked as visited (line 14). The algorithm stops when all solutions in the archive have been visited.

We have also implemented a restricted version of PLS called *component-wise step* (CW-step). CW-step adds non-dominated solutions in the neighbourhood of the initial solutions to the archive, but it does not explore the neighbourhood 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

Algorithm 2 Pareto Local Search

Input: An initial set of non-dominated solutions $\mathcal{A}$
$\mathsf{explored}(s) := \text{FALSE} \quad \forall s \in \mathcal{A}$
$\mathcal{A}_0 := \mathcal{A}$
while $\mathcal{A}_0 \neq \emptyset$ **do**
  $s :=$ extract solution randomly from $\mathcal{A}_0$
  $\mathcal{A}_0 := \mathcal{A}_0 \setminus \{s\}$
  for each $s' \in \mathcal{N}(s)$ **do**
    if $s \not\prec s'$ **then**
      $\mathsf{explored}(s') := \text{FALSE}$
      $\mathsf{Add}(\mathcal{A}, s')$
    end if
  end for
  $\mathsf{Filter}(\mathcal{A})$
  $\mathsf{explored}(s) := \text{TRUE}$
  $\mathcal{A}_0 := \{s \in \mathcal{A} \mid \mathsf{explored}(s) = \text{FALSE}\}$
end while
Output: $\mathcal{A}$

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 [27, 29, 31].

2.5 Quality assessment of multi-objective algorithms

The problem of evaluating the quality of multi-objective algorithms is still a current research topic [41]. 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 2.1. 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 analyse the results of several runs to assess the expected behaviour of the algorithm. To overcome these issues, researchers have devised quality indicators, which are scalar values calculated for each non-dominated set (unary indicators), or pairs of sets (binary indicators).

The hypervolume [15, 41] is an 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 2.1.

Good quality indicators as the hypervolume reflect desirable character-

istics 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 behaviour of the algorithm. A different approach that is well-suited for bi-objective problems is to 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 [18]. 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. 4.2 on Page 38, where each side shows the EAF differences in favour 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. [21] provide a more detailed explanation of these graphical tools.

Chapter 3

Anytime Improvements of TPLS

As we have seen in Section 2.3, TPLS starts with a high quality solution for a single objective (first phase) and then solves a sequence of scalarizations of the multi-objective problem (second phase). In the original version [29], each successive scalarization uses a slightly modified weight vector and starts from the best solution found by the previous scalarization. This approach has shown promising results for a number of multi-objective problems.

A drawback of the weight setting strategy used in TPLS is that the number of weights (or the overall available computation time) must be known in advance in order to equally distribute the computational effort among the different scalarizations. This also means that interrupting TPLS before it has explored all scalarizations is likely to produce a poor quality approximation of the Pareto front. In this sense, we may assume that TPLS has poor *anytime* properties. In fact, a central idea behind *anytime algorithms* [39] is to make an effort to produce a result with as high quality as possible, independently of the computation time allowed. In other words, an algorithm should be designed in such a way that for each possible stopping time it reaches a performance as high as possible. In this chapter, we propose new anytime weight setting strategies that ensure a fair distribution of the computational effort along the nondominated front independently of a priori knowledge about the number of scalarizations.

We study two types of anytime weight setting schemes. The first one, described in Section 3.1 and evaluated in Section 3.2, uses a regular distribution of the weight vectors. However, we noticed that its performance could further be improved by dynamically adapting the weights in dependence of the shape of the nondominated front. Our *adaptive* TPLS, described in Section 3.3, is inspired by the *dichotomic* method of Aneja and Nair [1] for obtaining all extreme supported non-dominated solutions in bi-objective optimisation problems. We extend the dichotomic method in two significant

ways. First, our adaptive approach tries to fill as early as possible the larger gaps in the nondominated front in order to satisfy the anytime property. Second, we extend the algorithm to use solutions found by previous scalarizations as seeds for further scalarizations, therefore, providing an adaptive TPLS strategy.

We test these strategies on the PFSP, for which we study two bi-objective problems: minimize both makespan and total flowtime, and minimize both makespan and total tardiness. We present in Section 3.4 an evaluation of our adaptive TPLS; it reaches results as good as the original D-TPLS while having good anytime behaviour.

3.1 The Regular "Anytime" Weight Setting Strategy

The strategy of defining successive weight vectors based on a strategy of minimal weight vector changes allows to generate very good approximations to the Pareto front of the areas "covered" by the weight vectors [29, 31]. However, if stopped prematurely, it leaves other areas of the Pareto front essentially unexplored. Here, we propose a first, rather straightforward *anytime* TPLS strategy. This *regular anytime* TPLS strategy (RA-TPLS) uses a regular distribution of the weights, similar to the traditional TPLS-like strategies, but gradually intensifies the search while ensuring a fair distribution of the computational effort along the Pareto frontier. At each iteration, a new weight is defined in the middle of the interval of two previous consecutive weights. This procedure defines a series of levels, each level providing a finer approximation to the Pareto front. The sequence of weights used by RA-TPLS is:

Level 0: $\lambda = 0, 1$ % (initial solutions)

Level 1: $\lambda = 0.5$

Level 2: $\lambda = 0.25, 0.75$

Level 3: $\lambda = 0.125, 0.375, 0.625, 0.875$

....

At each level, the number of weights and, hence, scalarizations, increases and the exploration of the Pareto front becomes successively more intense, in some sense filling the gaps in the Pareto front. Once the algorithm has completely finished a level of the search, the computational effort has been equally distributed in all directions. If the search stops before exploring all scalarizations at a certain level, the search would explore some areas of the

Algorithm 3 RA-TPLS

1: $s_1 := \mathsf{SLS}_1()$
2: $s_2 := \mathsf{SLS}_2()$
3: Add s_1, s_2 to Archive
4: $L_0 := \{(1,0)\}$; $L_i := \emptyset \quad \forall i > 0$
5: $S := \{(s_1,1),(s_2,0)\}$
6: $i := 0$
7: **while** not stopping criteria met **do**
8: $\quad(\lambda_{\text{sup}}, \lambda_{\text{inf}}) :=$ extract randomly from L_i
9: $\quad L_i := L_i \setminus (\lambda_{\text{sup}}, \lambda_{\text{inf}})$
10: $\quad\lambda := (\lambda_{\text{sup}} + \lambda_{\text{inf}})/2$
11: $\quad s := \mathsf{ChooseSeed}(S, \lambda)$
12: $\quad s' := \mathsf{SLS}_\Sigma(s, \lambda)$
13: $\quad$Add s' to Archive
14: $\quad L_{i+1} := L_{i+1} \cup (\lambda_{\text{sup}}, \lambda) \cup (\lambda, \lambda_{\text{inf}})$
15: $\quad S := S \cup (s', \lambda)$
16: $\quad\mathsf{Filter}(S)$
17: $\quad$**if** $L_i = \emptyset$ **then** $i := i + 1$
18: **end while**
19: Filter(Archive)
20: **Output:** Archive

Pareto front more thoroughly than others. In order to minimise this effect, RA-TPLS considers the weights within a level in a random order.

In RA-TPLS, following the principles of TPLS, each new scalarization (using a new weight) starts from a solution generated by a previous scalarization. In particular, RA-TPLS selects the seed of a new scalarization among the two solutions that were obtained by the previous scalarizations using the two weight vectors closer to the new weight. The algorithm computes the scalar values of these two solutions according to the new weight, and selects the solution with the best value as the seed for the new scalarization.

Algorithm 3 describes RA-TPLS in detail. There are three main data structures: L_i is a set of pairs of weights used in the previous iteration of the algorithm; S is a potential set of initial solutions together with the corresponding weight that was used to generate them; Archive is the archive of nondominated solutions. First, one initial solution is obtained for each objective using appropriate single-objective algorithms, $\mathsf{SLS}_1()$ and $\mathsf{SLS}_2()$. These new solutions and their corresponding weights, $\lambda = 1$ and $\lambda = 0$, respectively, are used to initialize L_0 and S. At each iteration, a pair of weights $(\lambda_{\text{sup}}, \lambda_{\text{inf}})$ is subtracted randomly from L_i and used to calculate a new weight given by $\lambda = (\lambda_{\text{sup}} + \lambda_{\text{inf}})/2$. Then, procedure $\mathsf{ChooseSeed}$ uses this weight to choose a solution from the set of seeds.

ChooseSeed finds, given a weight λ, the two solutions from the set of seeds S that have been generated using the weights closest to λ:

$$s_{\text{inf}} = \left\{ s_i \mid \max_{(s_i,\lambda_i)\in S} \{\lambda_i : \lambda_i < \lambda\} \right\} \qquad s_{\text{sup}} = \left\{ s_i \mid \min_{(s_i,\lambda_i)\in S} \{\lambda_i : \lambda_i > \lambda\} \right\} \tag{3.1}$$

Next, ChooseSeed calculates the scalar value of s_{sup} and s_{inf}, following Eq. (2.4), for the new weight λ, and returns the solution with the smaller scalar value. The chosen seed is the initial solution for SLS_Σ, the SLS algorithm used to tackle the scalarizations. The set of weights for the next iteration L_{i+1} is extended with the new pairs $(\lambda_{\text{sup}}, \lambda)$ and $(\lambda, \lambda_{\text{inf}})$. Moreover, the new solution s' and its corresponding weight is added to the set of seeds, which is filtered to keep only nondominated seeds. If the current set of weights L_i is empty, the search starts using weights from level L_{i+1}. This procedure may continue indefinitely.

3.2 Experimental Evaluation of RA-TPLS

As a test problem, we use bi-objective formulations of the permutation flowshop scheduling problem (PFSP. This problem has been presented in Section 2.2. We considered here two combinations of objectives, ($C_{\max}$,$\sum C_i$) and ($C_{\max}$,$\sum T_i$).

The two initial solutions for the weights $\lambda = 1$ and $\lambda = 0$ were obtained by running the IG algorithms for 1 000 iterations. In addition to these two solutions, we used 30 scalarizations each of 500 IG iterations. All algorithms were implemented in `C++`, and the experiments were run on a single core of Intel Xeon E5410 CPUs, running at 2.33 Ghz with 6MB of cache size, under Cluster Rocks Linux version 4.2.1/CentOS 4. For the experiments, we used 10 benchmark instances of size `50x20` and 10 instances of size `100x20` generated following the procedure described by Minella et al. [23]. Given the large discrepancies in the range of the various objectives, all objectives are dynamically normalised using the maximum and minimum values found during each run for each objective.

We examine the quality of the results in terms of the hypervolume unary measure [40, 15]. In the bi-objective space, the hypervolume measures the area of the objective space weakly dominated by the solutions in a nondominated set. This area is bounded by a reference point that is worse in all objectives than all points in the nondominated front. The larger is this area, the better is a nondominated set. To compute the hypervolume, the objective values of nondominated solutions are first normalized to the range $[1, 2]$, being the values corresponding the limits of the interval slightly smaller and larger, respectively, than the minimum and the maximum values ever found for each objective. We can therefore use $(2, 2)$ as the reference point for computing the hypervolume. We present the average hypervolume

as measured, for each strategy, across 15 independent runs.

3.2.1 Evaluation of RA-TPLS using hypervolume

We first study the potential benefits of the *anytime* property by comparing RA-TPLS with *1to2*, *2to1*, and D-TPLS. We compute the hypervolume value after each scalarization and give plots of the development of the hypervolume over the scalarization counter in Figure 3.1 on two instances of each combination of objectives (two) and instance size (two), that is, on eight instances.[1]

As expected, the hypervolume of the Pareto fronts generated by the three strategies *1to2*, *2to1*, and D-TPLS is rather poor for a small number of scalarizations, in particular, before the algorithms are allowed to reach the other objective. Interestingly, the PFSP biobjective problems also show clearly that the starting objective of TPLS can make a significant difference not only in the anytime performance but also in the final performance. Among the three, D-TPLS is clearly the one with the best final performance and also it improves the hypervolume faster than *1to2* and *2to1*. The latter indicates that for this problem, it is better to change the weights in larger steps.

By comparison, RA-TPLS quickly improves the hypervolume in very few scalarizations, and then continues to improve reaching a similar quality than D-TPLS, when the latter has performed half of its scalarizations; it is always significantly better than *1to2* and often better than *2to1*. This means that RA-TPLS outperforms the *1to2* strategy, and it produces a better result than D-TPLS and *2to1* whenever they are stopped early. However, in many instances, D-TPLS further improves the result during its second pass and, hence, we believe there is still room for improvement over RA-TPLS.

3.3 Adaptive Weight Setting Strategies

All weight setting strategies discussed so far generate a regular sequence of weights. That is, the weight vectors are predefined and they depend only on the number of scalarizations. In this section, we propose to dynamically generate weights in such a way that the algorithm adapts to the shape of the Pareto front. This *adaptive* strategy is inspired by the *dichotomic* scheme proposed by Aneja and Nair [1] for exact algorithms; recently, this scheme has also been adapted for the approximate case by Lust and Teghem [22]. The *dichotomic* scheme does not define the weights in advance but determines them in dependence of the solutions already found. More formally, given a pair of solutions (s_1, s_2), the new weight λ is perpendicular to the

[1] Additional plots for all instances are available from `http://iridia.ulb.ac.be/supp/IridiaSupp2009-009`.

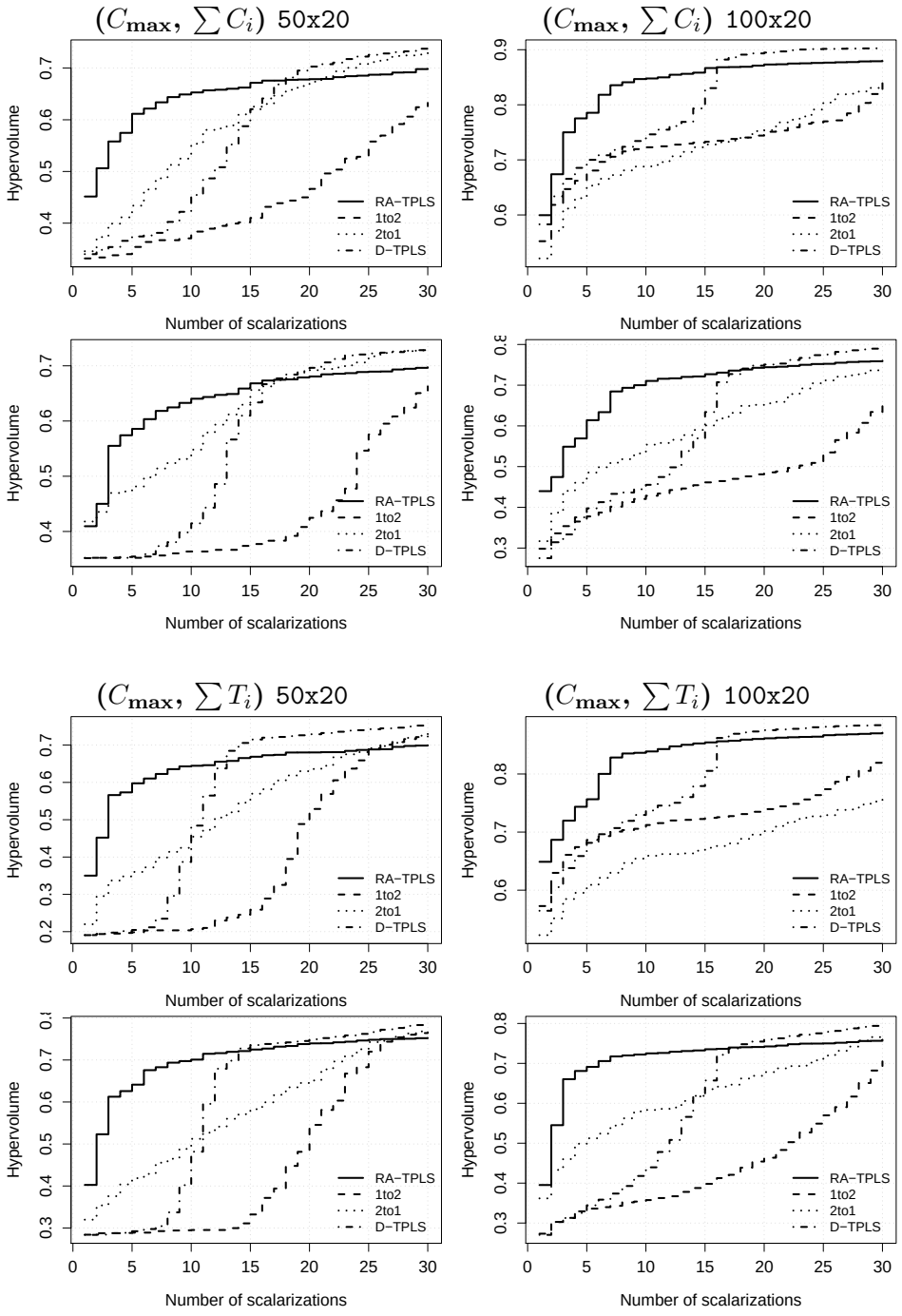

Figure 3.1: Development of the hypervolume over the number of scalarizations for *1to2*, *2to1*, *D-TPLS* , and *RA-TPLS* . Results are given for four instances of size 50x20 (left column) and four instances of size 100x20 (right column). The objective combinations are $C_{\max}$, $\sum C_i$ (top four plots) and $C_{\max}$, $\sum T_i$ (bottom four plots).

Algorithm 4 Adaptive "Anytime" TPLS Strategy

1: $s_1 := \mathsf{SLS}_1()$
2: $s_2 := \mathsf{SLS}_2()$
3: Add s_1, s_2 to Archive
4: $S := \{(s_1, s_2)\}$
5: **while** not stopping criteria met **do**
6: $(s_{\text{sup}}, s_{\text{inf}}) := \arg\max_{(s,s')\in S}\{\overline{\vec{f}(s)\vec{f}(s')}\}$
7: Calculate λ perpendicular to $\overline{\vec{f}(s_{\text{sup}})\vec{f}(s_{\text{inf}})}$ following Eq. (3.2)
8: $s'_{\text{sup}} := \mathsf{SLS}_\Sigma(s_{\text{sup}}, \lambda)$
9: $s'_{\text{inf}} := \mathsf{SLS}_\Sigma(s_{\text{inf}}, \lambda)$
10: Add s'_{sup} and s'_{inf} to Archive
11: $\mathsf{Update}(S, s'_{\text{sup}})$
12: $\mathsf{Update}(S, s'_{\text{inf}})$
13: **end while**
14: Filter(Archive)
15: **Output:** Archive

segment defined by s_1 and s_2, that is:

$$\lambda = \frac{f_2(s_1) - f_2(s_2)}{f_2(s_1) - f_2(s_2) + f_1(s_2) - f_1(s_1)} \tag{3.2}$$

The exact algorithm used by Aneja and Nair [1] is deterministic, and, hence, applying the same weight results in the same solution. In the scheme of Lust and Teghem [22], later scalarizations do not use as seeds the solutions found by previous scalarizations. Finally, the *dichotomic* scheme used in these two earlier reports has a natural stopping criterion, and it progresses recursively depth-first. As a result, if stopped early, it would assign an uneven computational effort along the front, possibly leading to a bad distribution of solutions and, hence, to bad anytime behaviour.

We extend the *dichotomic* strategy to the TPLS framework with the goals of (i) making effective use of solutions found by previous scalarizations to seed later scalarizations; and (ii) satisfying the *anytime* property. Our resulting *adaptive* strategy is described in Algorithm 4. The main data structure is a set S of pairs of solutions found in previous scalarizations. This set is initialized with the solutions found by optimizing each single-objective using $\mathsf{SLS}_1()$ and $\mathsf{SLS}_2()$. At each iteration, the algorithm selects the pair of solutions $(s_{\text{sup}}, s_{\text{inf}}) \in S$ that define the longest segment in the objective space, using the Euclidean distance with the normalized values of each objective. The idea is to focus the search on the largest gap in the Pareto frontier in order to obtain a well-spread set of nondominated solutions. This is different from the original *dichotomic* scheme, which explores all segments recursively. Then, the algorithm calculates a new weight λ per-

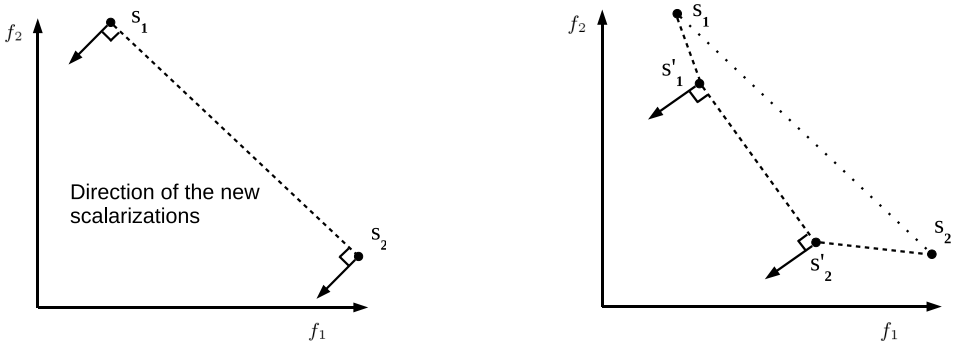

Figure 3.2: A single iteration of the AN-TPLS algorithm. On the left the state before the iteration and on the right after S has been updated. The next segment which will be considered is (s'_1, s'_2) because of its larger distance.

pendicular to the segment defined by s_{sup} and s_{inf} in the objective space, following Eq. (3.2). Next, the underlying single-objective SLS algorithm, SLS_Σ, is run two times using the weight λ, one time starting from solution s_{sup} and one time from solution s_{inf}. This is different from the *dichotomic* strategy [1, 22], where a scalarization is tackled only once without using as initial solution any of the previously found ones.

In the last step of an iteration, procedure Update updates the set of seeds S using the new solutions found. If s' is a new solution, any single solution in S dominated by s' is replaced with s', and any pair of solutions (weakly) dominated by s' is removed. The *dichotomic* scheme [1, 22] only accepts solutions for inclusion in S if they lie *within* the triangle defined by the solutions s_{sup} and s_{inf}, and their local ideal point (see Fig. 3.3). Solutions outside the gray area are either dominated or not supported (not optimal for any scalarization). Heuristic algorithms may, however, generate supported solutions that are in the gray area *outside* the triangle; therefore, our *adaptive* strategy accepts *all* solutions in the gray area for inclusion in S. If a solution s' is accepted for inclusion in S, then the segment $(s_1, s_2) \in S$ with $f_1(s_1) < f_1(s') < f_1(s_2)$ is removed, and two new segments (s_1, s') and (s', s_2) are added to S. Since each iteration produces two new solutions (s'_{sup} and s'_{inf}), a maximum of three new segments are added to S every iteration. Figure 3.2 shows an example of the update of S after one iteration of the *adaptive* algorithm. We call this algorithm AN-TPLS in what follows (for adaptive normal TPLS).

We call *Adaptive focus* (AF-TPLS) a small variation of AN-TPLS. This variation is motivated by the fact that if two adjacent segments in S are almost parallel, two scalarizations will be solved using the same seed (the solution shared by the two segments) and very similar weights since also the two vectors perpendicular to the segments will be almost parallel. By

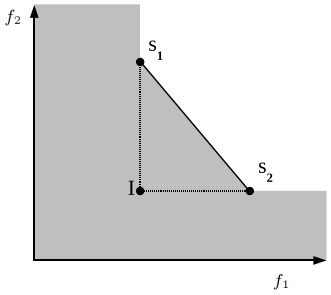

Figure 3.3: Only solutions in the gray area are accepted as seeds for further scalarizations (See text for details).

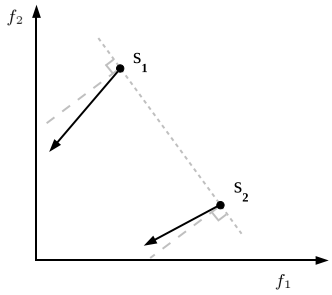

Figure 3.4: AF-TPLS strategy.

carefully analysing the execution of AN-TPLS, we observed that such two scalarizations often generate the same solution. In order to avoid this waste of computational effort and to focus the search towards the center of each segment, we modified the calculation of the search direction of each scalarization. Given a segment $(s_1, s_2) \in S$, with $f_1(s_1) < f_1(s_2)$, we generate two weights λ_1 and λ_2 as:

$$\lambda_1 = \lambda - \theta \cdot \lambda \quad \text{and} \quad \lambda_2 = \lambda + \theta(1 - \lambda) \ . \tag{3.3}$$

where λ is the weight perpendicular to the segment calculated by Eq. (3.2), and θ is a parameter that modifies λ towards the center of the segment (see Fig. 3.4).

These two new weights replace the perpendicular weight λ in Algorithm 4. That is, the run of the SLS algorithm that uses s_1 as seed solves a scalarization according to weight λ_1, while the run seeded with s_2 uses the weight λ_2. A value of $\theta = 0$ would reproduce the AN-TPLS strategy.

3.4 Experimental Evaluation of the Adaptive TPLS Strategies

For AN-TPLS and AF-TPLS we perform the same experiments using the same experimental setup as described in Section 3.2; for AF-TPLS we use a setting of $\theta = 0.25$.

3.4.1 Evaluation of adaptive strategies using hypervolume

Figure 3.5 shows that the adaptive strategies greatly improve over the RA-TPLS strategy from the very start of the algorithm, thereby further enhancing the *anytime* behaviour. In addition, the final quality is also greatly improved, reaching the hypervolume value obtained by D-TPLS. This means

that the *adaptive* TPLS strategies can effectively replace D-TPLS, the former providing results similar to the latter if both are run for sufficiently long time, and much better results if the algorithms are stopped after few scalarizations. With respect to the value of θ in the *adaptive* strategy, there is not a completely clear conclusion. The *adaptive* strategy with $\theta = 0.25$ (AF-TPLS) is clearly better than AN-TPLS ($\theta = 0$) on few instances and never clearly worse. However, for most instances, and especially for the larger ones, the difference is small.

3.4.2 Statistical Analysis

We perform a statistical analysis of the approaches. The analysis is based on the Friedman test for analysing non-parametric unreplicated complete block designs, and its associated post-test for multiple comparisons [6]. We perform the following procedure separately for each combination of objectives, each instance size `50x20` and `100x20`, and stopping criteria of 10, 20 and 30 scalarizations. First, we calculate the median hypervolume of the 15 runs of each algorithm for each instance. Then, we perform a Friedman test using the ten instances as the blocking factor, and the different strategies as the treatment factor. In all cases, the Friedman test rejects the null hypothesis with a p-value much lower than 0.05. Hence, we proceed to calculate the minimum difference between the sum of ranks of two strategies that is statistically significant (ΔR_α), given a significance level of $\alpha = 0.05$. We examine which strategies are not significantly different to the one with the lowest sum of ranks. Table 3.1 summarises the results of the statistical analysis. It shows the value of ΔR_α for $\alpha = 0.05$, the strategies sorted by increasing sum of ranks; and the difference between the sum of ranks of each strategy and the best strategy (ΔR). The strategies that are not significantly different from the best are shown in boldface. The table shows that AF-TPLS is often the best, and never significantly different from the best. For a low number of scalarizations, the *adaptive* strategies (AN-TPLS and AF-TPLS) are always superior to the classical TPLS strategies. Moreover, AF-TPLS is never significantly worse than D-TPLS, when the latter runs until completion (30 scalarizations). In conclusion, AF-TPLS would be the strategy to be chosen.

3.5 Summary of results

In this chapter, we addressed a perceived weakness of the TPLS framework, namely, that the number of scalarizations must be specified in advance and that stopping the algorithm earlier results in poor performance. We proposed weight setting strategies that try to fulfill the *anytime* property, that is, they can be stopped at any time during its execution, and the result would be a well-spread approximation of the Pareto front. Our first proposal, the

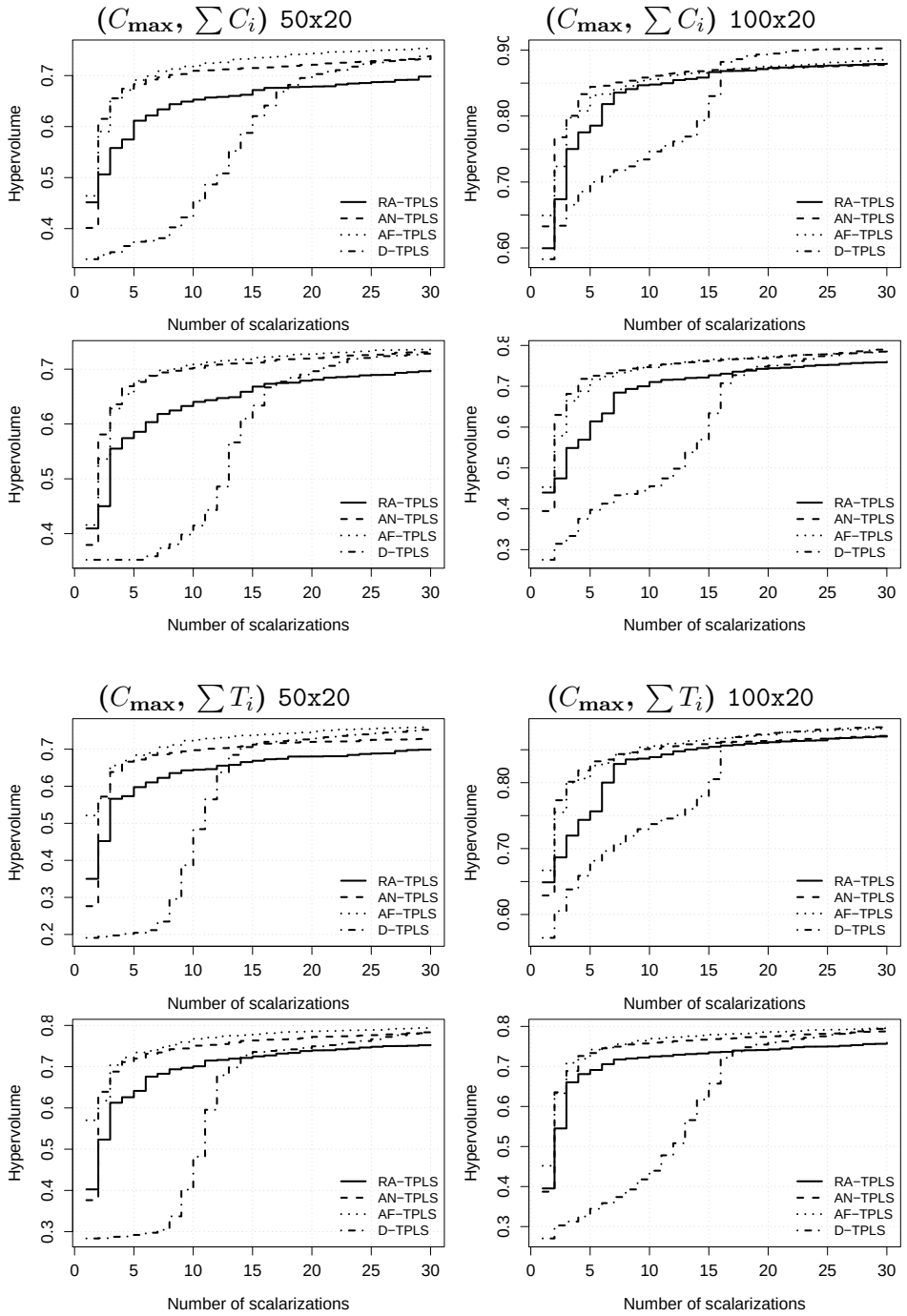

Figure 3.5: Development of the hypervolume over the number of scalarizations for *D-TPLS* , RA-TPLS, AN-TPLS, AF-TPLS. Results are given for four instances of size 50x20 (left column) and four instances of size 100x20 (right column). The objective combinations are $C_{\max}$, $\sum C_i$ (top four plots) and $C_{\max}$, $\sum T_i$ (bottom four plots).

Table 3.1: For each number of scalarizations, strategies are ordered according to the rank obtained. The numbers in parenthesis are the difference of ranks relative to the best strategy. Strategies which are not significantly different to the best one are indicated in bold face. See text for details.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
($C_{\max}$, $\sum C_i$) 50x20		
10	2.94	**AN-TPLS (0), AF-TPLS (2),** RA-TPLS (16), 2to1(26), D-TPLS (36), 1to2(46)
20	5.23	**AN-TPLS (0), AF-TPLS (0),** D-TPLS (12), RA-TPLS (28), 2to1(30), 1to2(44)
30	6.49	**AN-TPLS (0), AF-TPLS (1), D-TPLS (3),** 2to1(20), RA-TPLS (31), 1to2(41)
($C_{\max}$, $\sum C_i$) 100x20		
10	5.20	**AF-TPLS (0), AN-TPLS (4),** RA-TPLS (14), 2to1(28), D-TPLS (35), 1to2(46)
20	6.76	**AF-TPLS (0),** AN-TPLS (8), D-TPLS (15), RA-TPLS (21), 2to1(36), 1to2(46)
30	8.40	**D-TPLS (0), AF-TPLS (7),** AN-TPLS (12), 2to1(27), RA-TPLS (30), 1to2(44)
($C_{\max}$, $\sum T_i$) 50x20		
10	3.79	**AF-TPLS (0),** AN-TPLS (6), RA-TPLS (18), 2to1(32), D-TPLS (34), 1to2(48)
20	3.12	**AF-TPLS (0),** AN-TPLS (10), D-TPLS (17), RA-TPLS (29), 2to1(39), 1to2(49)
30	7.10	**D-TPLS (0), AF-TPLS (1),** AN-TPLS (15), 2to1(21), 1to2(37) , RA-TPLS (40)
($C_{\max}$, $\sum T_i$) 100x20		
10	3.60	**AF-TPLS (0),** AN-TPLS (8), RA-TPLS (19), 2to1(31), D-TPLS (38), 1to2(48)
20	7.50	**AF-TPLS (0),** AN-TPLS (11), D-TPLS (13), RA-TPLS (20), 2to1(37), 1to2(45)
30	6.43	**D-TPLS (0), AF-TPLS (3),** AN-TPLS (20), RA-TPLS (24), 2to1(34), 1to2(45)

RA-TPLS strategy, has the *anytime* property and outperforms the classical TPLS strategies if the algorithms are stopped before completion. However, its final quality is not as good as that of D-TPLS. Our second proposal is an adaptive strategy that has the *anytime* property, and it can be further fine-tuned through a parameter θ. The two adaptive variants studied here, AN-TPLS and AF-TPLS ($\theta = 0.25$), outperform the classical TPLS strategies at any time during their execution. In fact, the adaptive strategies proposed here should replace the classical TPLS strategies in situations where the computation time available is not known in advance.

Chapter 4

Hybridization of TPLS and PLS for PFSP

In this chapter we present a new hybrid algorithm for the bi-objective PFSP that combines the TPLS and the PLS frameworks, the steps followed during its design and its final evaluation. To design this multi-objective SLS algorithm, we follow a bottom-up engineering approach, and our first step, described in Section 4.1, 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 [33], which is a state-of-the-art SLS algorithm for the PFSP when minimising the makespan. 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 SLS algorithm engineering, presented in Section 4.2, we further extend 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 neighbourhood 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 the two frameworks.

We evaluate the quality of the resulting hybrid algorithm, TP+PLS, and present the results in Section 4.3. In order to perform a fair evaluation, we reimplemented MOSA [37] and MOGLS [3], two state-of-the-art algorithms for bi-objective PFSPs [23]. Our experimental results show the excellent performance of our TP+PLS algorithm: it often finds Pareto fronts that completely dominate those found by these two state-of-the-art algorithms.

The contributions presented in this chapter are several high-performing single-objective algorithms for various PFSPs, improved variants of the TPLS and PLS frameworks, and a methodology to engineer SLS algorithms

Algorithm 5 Iterated Greedy

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

for bi-objective problems. We further combine these individual contributions to produce a new hybrid algorithm that advances the state-of-the-art for all five bi-objective PFSPs we studied.

4.1 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.

4.1.1 SLS algorithm for *PFSP*-$C_{\max}$

For the *PFSP*-$C_{\max}$, we use the iterated greedy (IG) algorithm (*IG*-$C_{\max}$) by Ruiz and Stützle [33]. The IG algorithm has shown to be very competitive when compared to more complex SLS algorithms, and it requires very few parameters to be set. Algorithm 5 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.

IG-$C_{\max}$ uses the well known NEH constructive heuristic [25] 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 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* neighbourhood. This neighbourhood is defined such that all π' are neighbours of π if they can be obtained from π by removing a job π_i and inserting it at a different position j. For each job, the best position to be inserted in is determined. If this best move improves the objective value, it is immediately applied. These first-improvement *insertion* moves are applied until a local optimum is found. A speed-up proposed by Taillard [34] is used that allows 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:

$$T = T_{\mathrm{c}} \cdot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}, \tag{4.1}$$

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 [33] report some experiments to identify good parameter settings. According to their findings, the algorithm is quite robust to different parameter settings. They finally set $d = 4$ and $T_{\mathrm{c}} = 0.4$, and we use the same parameter settings.

4.1.2 SLS algorithm for *PFSP-SFT*

Given the very good performance of IG for makespan minimisation (*PFSP-*$C_{\max}$), we decided to adapt the IG algorithm to tackle the *PFSP-SFT*. Although the main outline of IG (Algorithm 5) 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 neighbourhood is only valid for makespan minimisation. Without this technique, the complexity of exploring the full insertion neighbourhood becomes $O(mn^3)$, and, hence, there is no clear a priori advantage of using this neighbourhood operator over pairwise exchanges of jobs at different positions. Therefore, we implement and test three neighbourhood operators based on the following moves: *insertion*, which is the same as for makespan minimisation but without the speed-up; *exchange*, which exchanges the positions of any two 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 neighbourhood search. Experimental tests [9, 10], which are not reported here, showed that the neighbourhood 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 remainder possible moves in the sequence. 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 sequential evaluations before reaching a local optimum. For this purpose, we add a parameter N_{LS} that limits the number of sequential evaluations. Experimental tests suggest that the local search often finds a local optimum in less than five sequential evaluations. Therefore, we test possible settings of $N_{\mathrm{LS}} = \{1, 2, 3, 4, \infty\}$. In case $N_{\mathrm{LS}} = \infty$, the search stops at a local optimum, no matter how many iterations 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 [38].

We modified the formula for computing the temperature in the acceptance criterion (Eq. 4.2) because of the different range of objective values. We experimentally found [10] that good results are produced by using the fomula:

$$T = T_{\mathrm{c}} \cdot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{m \cdot 10}, \tag{4.2}$$

which is the same as Eq. 4.1 but multiplied by n.

4.1.3 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 [36] 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 [9], 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 minimises 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. 4.1), as we did 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 is obtained from the relative difference between the objective value of the current and the new solution:

$$p = \exp\left(-100 \cdot (f(\pi') - f(\pi))/f(\pi))/T_{\mathrm{c}}\right), \tag{4.3}$$

where T_{c} is a constant parameter.

4.1.4 SLS algorithms for the scalarized problems

Given the very good performance of our IG algorithms to minimize each objective [9, 10], we also use IG to solve each scalarized problem. To define the acceptance probability in the procedure AcceptanceCriterion (Algorithm 5, line 6), we use the same formula as for the tardiness objective (Eq. 4.3). An other important adaptation of IG for solving the scalarized problems is the normalization of the objectives 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.

4.1.5 Parameter tuning

Since we do not know good values for the parameters of each of the IG variants, we calibrated them using an automatic tuning procedure.

We fine-tune the parameters d, T_c and N_{LS} of each IG variant for the single-objective problems *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 $[0, 100]$ (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 minimisation problem (*PFSP-*C_{max}), because Ruiz and Stützle [33] have already proposed good parameter settings (see Section 4.1.1).

We used iterative F-Race [4] for the automatic tuning. For this purpose, we generated 100 instances for each number of jobs in {`20,50,100,200`}, following the procedure described by Minella et al. [23]. These instances have `20` machines, since they are the hardest considered in this report. For each problem and each instance size, we performed 5 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 using a time limit of $(0.1 \cdot n \cdot m)/30$ seconds, that is the time used by Minella et al. [23] divided by 30. This is done based on the assumption that the TPLS phase of the final algorithm is allocated half of the total time and that we use 15 scalarizations overall.

We considered the possibility of using different parameters to tackle the scalarized problems based on the weights that define these problems. We tuned the algorithms for the weights $\lambda = \{0.25, 0.5, 0.75\}$. The best configurations determined by iterative F-Race were not very different for each weight, therefore one combination of parameter was enough for each scalarized problems.

To assess the importance of the parameter tuning on the solution quality, we compared two versions of our algorithm: one uses parameter settings specific for each number of jobs, obtained with different tuning runs for each number of jobs, whereas the other uses the same parameter settings for all instances size of a problem, obtained by tuning the algorithm using all instances size. The common settings produce only slightly worse results [12] and they are arguably more robust when applied to instances of intermediate size not considered in the tuning. Table 4.1 describes the parameter settings of IG we used to produce all results given in this report. We provide as supplementary material [12] results obtained using size-specific parameters.

Table 4.1: IG parameter settings. The settings for *PFSP-*$C_{\max}$ are taken from Ruiz and Stützle [33], in particular the neighbourhood they use is based on best insertion moves and is stoppped when reaching a local optimum. The other settings were found by means of automatic tuning (Section 4.1.5).

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

4.2 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 report to analyse and compare multi-objective algorithms. Next, we empirically analyse the main components of the two multi-objective frameworks used in this report, 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 is be compared in Section 4.3 with the state-of-the-art algorithms for the bi-objective PFSPs.

4.2.1 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 neighbourhood operator used for generating new solutions. We also discuss a simple way to improve the *anytime* property [39] 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 NEH heuristic appropriate for each objective (Section 4.1), and (*iii*) two solutions obtained by IG (Section 4.1) for each objective after 10 000 iterations. Figure 4.1 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 solutions produces better non-dominated sets;

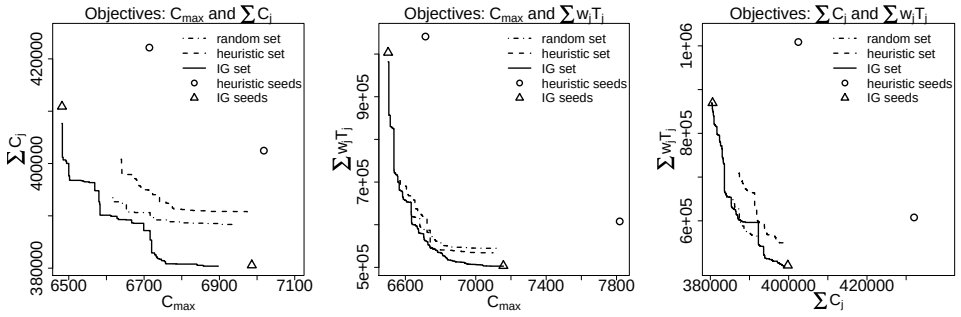

Figure 4.1: In each plot is given one non-dominated set obtained by PLS using different quality of seeds for instance `100x20_3`. The randomly generated solutions are outside the range shown.

Table 4.2: Confidence Intervals (with level=0.95) of the computation time (seconds) of PLS for different kinds of seeding solutions (Random, Heuristic, or IG seeds). For details see the text.

Problems	n x m	Random	Heuristic	IG
PFSP-($C_{\max}$, *SFT*)	50x20	[7.794, 9.896]	[5.596, 6.871]	[4.291, 4.837]
	100x20	[161.4, 193.4]	[134.6, 149.9]	[149.0, 175.3]
PFSP-($C_{\max}$, *WT*)	50x20	[29.17, 34.05]	[31.94, 35.77]	[22.5, 25.53]
	100x20	[577.5, 706.5]	[690.4, 844.1]	[590.4, 662.5]
PFSP-(*SFT*, *WT*)	50x20	[24.27, 29.16]	[27.5, 28.85]	[22.36, 25.03]
	100x20	[673.5, 811.4]	[776.5, 839]	[831.7, 958.8]

this result is strongest for the bi-objective problem minimizing makespan and sum of flowtime ($C_{\max}, \sum C_i$). Although not shown here for brevity, but available as supplementary material [12], the differences between the EAFs obtained for each kind of seed across 10 runs confirm this result: The best non-dominated sets, in terms of a wider range of the Pareto front and higher quality, are obtained when using the IG seeds. We also examined the computation time required by PLS in dependence of the initial seed in Table 4.2.

From the results of the plots and the table, we conclude that seeding PLS with solutions of higher quality does not strongly affect the computation time required by PLS but it does (positively) influence the quality of the non-dominated sets generated by PLS. Therefore, we expect that seeding PLS with solutions obtained by TPLS will further enhance the quality of the results without an excessive computation time overhead.

Table 4.3: Confidence intervales (with level=0.95) of the computation time (seconds) of PLS using different neighbourhood operators. For details see the text.

Problems	n x m	Insertion	Exchange	Ex. + Ins.
PFSP-($C_{\max}$, *SFT*)	50x20	[1.379, 1.766]	[1.973, 2.448]	[4.378, 5.303]
	100x20	[65.93, 75.89]	[71.11, 84.01]	[147.6, 167.7]
PFSP-($C_{\max}$, *WT*)	50x20	[9.451, 10.77]	[11.87, 14.02]	[21.67, 24.4]
	100x20	[236.6, 267.1]	[292.9, 336.4]	[577.3, 645.9]
PFSP-(*SFT*, *WT*)	50x20	[8.682, 10.35]	[13.04, 15.45]	[22.04, 25.39]
	100x20	[204.4, 273.7]	[458, 527.9]	[799.1, 945.5]

Neighborhood operator. We test variants of PLS based on three different neighbourhoods: (*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 neighbourhoods. We measure the computation time of PLS with each neighbourhood operator for different combinations of objectives in Table 4.3. The computation time of the combined exchange and insertion neighbourhood is slightly more than the sum of the computation times for the exchange and the insertion neighbourhoods separately. For comparing the quality of the results, we examine the EAF differences of 10 independent runs. Figure 4.2 gives two representative examples. Typically, the exchange and insertion neighbourhoods 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 neighbourhood (bottom plot). Given the complementarity of exchange and insertion to perform well in different regions, we decided to use the combined neighbourhood 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 over, and, in that case, the remaining time would be wasted. We modify PLS to continue exploring the search space up to the time limit in the following manner. After exploring the immediate neighbourhood of all solutions in the archive, our time-bounded PLS variant continues the search on the extended neighbourhood that results from applying two times the neighbourhood operator to the solutions in the archive. That is, PLS extends the search to the neighbourhood of those solutions that were initially discarded because they were dominated. This approach, however, only improves slightly the quality of the results for the smallest instances, because in larger instances

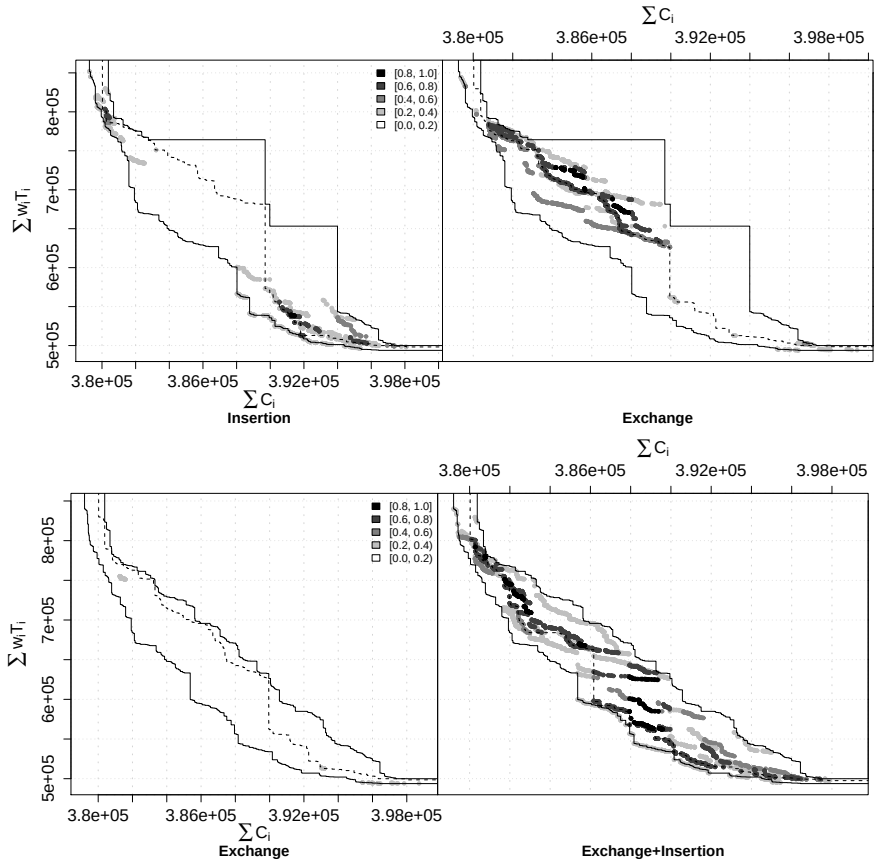

Figure 4.2: EAF differences for (*top*) insertion vs. exchange and (*bottom*) exchange vs. exchange and insertion, for *PFSP*-(*SFT*, *WT*). Dashed lines are the median attainment surfaces of each algorithm. Black lines correspond to the overall best and overall worst attainment surfaces of both algorithms.

the time required by PLS to explore the neighbourhood of non-dominated solutions is already larger than the computation time limit used in the final evaluation.

4.2.2 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, and we have presented an *adaptive anytime* weight setting strategy that we recently proposed [11] and compared positively to the classical deterministic strategies [29]. We examine wether TPLS performs better than a *restart* strategy where the initial solution for each scalarization is generated anew, independently of previously found solutions. Finally, we discuss appropriate settings for the number of scalarizations.

Weight Setting Strategy. The original TPLS [29] (see Section 2.3) used a regular sequence of equally distributed weights defined from either the first objective to the second (*1to2*) or viceversa (*2to1*). The *Double* TPLS (D-TPLS) strategy [29] performs the first half of the scalarizations sequentially from one objective to the other, and another half in the inverse direction.

We proposed in Chapter 3 TPLS variants [11] that try to improve the *anytime* property so that for each possible stopping time they reach a performance as high as possible. Among several proposed anytime variants of TPLS, the *Adaptive Focus* variant, henceforth simply called *adaptive* TPLS (A-TPLS), was found to perform overall best, achieving a performance similar to D-TPLS. 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 TP+PLS algorithm.

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

We implemented a *Restart* strategy derived from the *adaptive* TPLS discussed above. In this *Restart* strategy, the initial solution of each scalarization is generated by variants of the NEH heuristic for the scalarized prob-

lems [1] The *Restart* strategy solves only one scalarization for each pair of solutions, however, in our experiments it still executes the same number of scalarizations as A-TPLS. We tested the algorithms on 5 randomly generated instances of size `20x20`, `50x20`, `100x20`. 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. Figure 4.3 illustrates these differences in two particular instances for one combination of objectives, but we obtain similar results for the other bi-objective problems. 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 carry 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 one 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 one of these, and vice-versa. We test the A-TPLS algorithm with different number of scalarizations $N_{\text{scalar}} = \{10, 20, 40, 80\}$. We used the hypervolume indicator [41, 15] to compare the quality obtained for the four values for N_{scalar}, the objectives values are normalized to the range $[1, 2)$ in such a way that 2 corresponds to the worst value 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 `50x20_1` and `100x20_1` and 25 independent runs of A-TPLS per instance. We performed an analysis of variance (ANOVA) in order to determine if there is significant difference among the results. The difference in the solution quality appeared to be rather small, showing that A-TPLS is rather robust to the change of this parameter. However, significant differences are

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

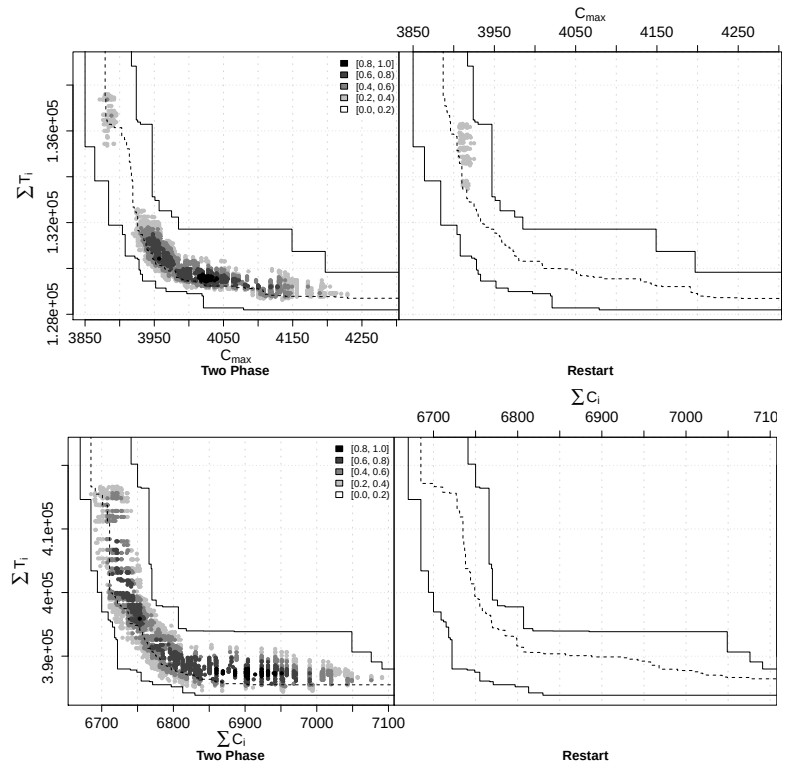

Figure 4.3: EAF differences between TPLS on the left and *Restart* on the right. Results are shown for objectives $C_{\max}$ and $\sum C_i$, and instances `50x20_1` (top) and `100x20_1` (bottom).

Table 4.4: Confidence intervals (with level=0.95) of the computation time (seconds) for CW-step and PLS seeding with the output of TPLS. For details see the text.

Problems	n x m	CW-step	PLS
PFSP-($C_{\max}$, *SFT*)	50x20	[0.1934, 0.2186]	[1.956, 2.555]
	100x20	[1.374, 1.56]	[53.66, 73.59]
PFSP-($C_{\max}$, *WT*)	50x20	[0.3542, 0.3808]	[6.484, 7.959]
	100x20	[2.289, 2.57]	[209.8, 275.3]
PFSP-(*SFT*, *WT*)	50x20	[0.334, 0.3563]	[7.909, 9.446]
	100x20	[2.349, 2.569]	[323.7, 385.8]

never in disfavour of 10 and 20 scalarizations, and therefore, we focus on a rather small number of scalarizations in our final algorithm.

4.2.3 TPLS + CW-step, TPLS + PLS

As a final step of our algorithm engineering process, we compare the performance trade-offs incurred by starting either PLS or the CW-step from the set of solutions returned by TPLS. For all instances, we generated 10 initial sets of solutions by running TPLS for 30 scalarizations each of 1000 iterations of IG. Then, we apply CW-step and PLS each starting from the same set of initial solutions in order to reduce variance.

Table 4.4 gives the computation time that is incurred by PLS and the CW-step after TPLS has finished. The results clearly show that the CW-step incurs only a minor overhead with respect to TPLS, while PLS requires considerably longer times, especially on larger instances. However, the times required for PLS to finish are much lower than when seeding it with only two very good solutions (compare with Table 4.2 on Page 36). With respect to solution quality, Figure 4.4 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 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.

4.2.4 Hybrid TP+PLS algorithm

We design 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 we

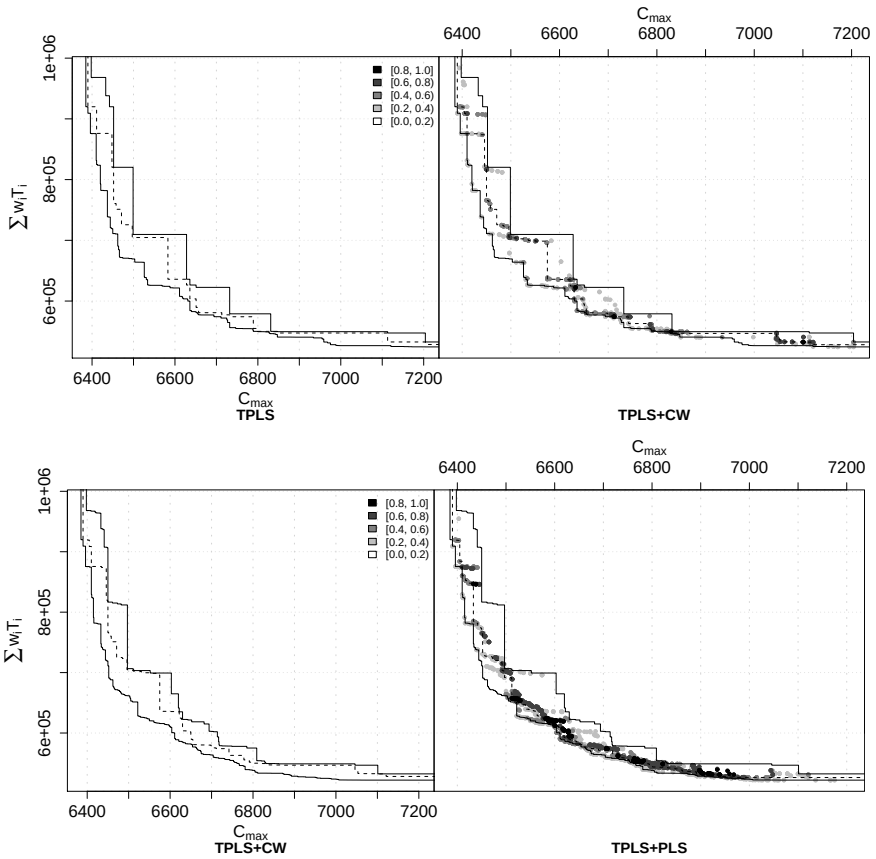

Figure 4.4: EAF differences between (top) simple TPLS vs. TPLS + CW-step, and (bottom) TPLS + CW-step vs. TPLS + PLS. Objectives are $C_{\max}$ and $\sum w_i T_i$.

use A-TPLS to perform a series of scalarizations and 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 neighbourhood 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.

4.3 Performance evaluation of TP+PLS

4.3.1 Experimental Setup

For the experimental analysis of TP+PLS, we use the same benchmark instances as Minella et al. [23]. This benchmark set consists of 10 instances of size {20,50,100}x{5,10,20} and {200}x{10,20}, originally proposed by Taillard [35] and augmented with due dates by Minella et al.

All code is implemented in C++ and compiled with gcc version 3.4.6 using -O3 flag. Experiments are run on a Intel Xeon E5410 CPU 2.33 Ghz with 6MB cache, under Cluster Rocks Linux, each process uses one single core due to the sequential implementation of the algorithm.

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. [23]. Each experiment is repeated 25 times with different random seeds. The main parameters of our TP+PLS 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 assign to $\text{IG}_{\{1,2\}}$ being 1.5 the time assigned to IG_{Λ}. Once all scalarizations are finished, the remaining time is spent on PLS.

Table 4.5 gives the value of these parameters for each instance size. We obtain these values as follows. 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 PLS 200 seconds for instances of 200x20, 100 seconds for instances of 100x20, and 10 seconds for instances of {20,50}x20. We use 12 scalarizations for all instance sizes. Nonetheless, TP+PLS appears to be very robust with respect to variations of these settings.

We discussed in Section 4.1.5 the assumptions made in order to choose the computation time limit used in the automatic fine-tuning of the IG algorithms. Given the final settings shown in Table 4.5, these assumptions appear to be only roughly accurate in comparison with the actual time allocated to TPLS and the number of scalarizations. Despite these discrepancies, we believe that the IG algorithms were sufficiently fine-tuned and we expect these parameters to be reasonably accurate.

4.3.2 Comparison with reference sets

We first compare the results of our TP+PLS with the reference sets provided by Minella et al. [23]. 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

Table 4.5: Settings for the components of TP+PLS. $IG_{\{1,2\}}$ 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. 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.

Instance size	Time for $IG_{\{1,2\}}$	Time for IG_{Λ}	N_{scalar}	Overall time
20x5	0.75	0.5	12	10
20x10	1.5	1.0	12	20
20x20	3.0	2.0	12	40
50x5	2.25	1.5	12	25
50x10	4.5	3	12	50
50x20	9	6	12	100
100x5	2.5	1.66	12	50
100x10	5.0	3.33	12	100
100x20	10.0	6.66	12	200
200x10	10.0	6.66	12	200
200x20	20.0	13.33	12	400

were obtained on an Intel Dual Core E6600 CPU running at 2.4Ghz, which is similar in speed to the one we use. As illustrative examples of the comparison between TP+PLS and the reference sets, Fig. 4.5 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 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 ten 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.

4.3.3 Comparison with MOSA and MOGLS

Given the good quality of TP+PLS suggested by the comparison with reference sets, we then compare the results of TP+PLS with Multi-Objective Simulated Annealing [37] and Multi-Objective Genetic Local Search [3], two algorithms that have recently been shown to be state of the art for the bi-objective PFSPs [23]. To make this comparison more fair and account for possible differences in implementations and computing environment, we have reimplemented these algorithms. We describe first the implementation

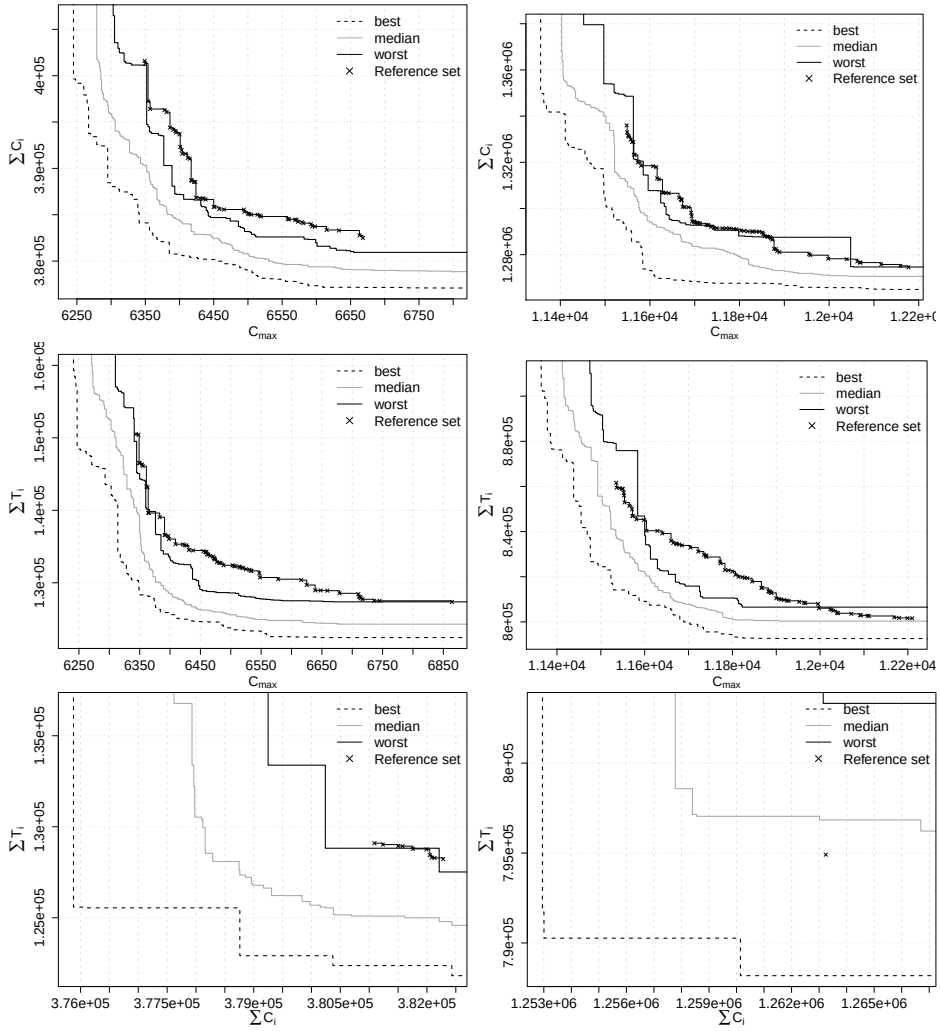

Figure 4.5: Attainment surfaces of TP+PLS against the reference set for instances `DD_Ta082 (100x20)` on the left and `DD_Ta102 (200x20)` on the right, for top: *PFSP*-($C_{\max}$, SFT), middle: *PFSP*-($C_{\max}$, TT) and bottom: *PFSP*-(SFT, TT).

of these two algorithms, and we present later the results of our experimental analysis.

Multi-Objective Simulated Annealing (MOSA). Varadharajan and Rajendran [37] designed MOSA for the bi-objective *PFSP*-($C_{\max}$, *SFT*) (minimisation of the makespan and sum of flowtimes). Recently, Minella et al. [23] 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 [37] 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 classical simulated annealing algorithm, henceforth denoted single-SA, that 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 an 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 [32], 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 over the makespan being $(1, 0.83, 0.66, 0.5)$. The acceptance criterion is similar to the one defined in Eq. 4.3 on Page 33.

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 4.1.1). Moreover, since the stopping criterion of MOSA is a temperature threshold, we further modify the algorithm to stop after a certain computa-

tion 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 [3], was the second-best algorithm for the bi-objective PFSPs studied in the review of Minella et al. [23]. 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 [2] 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 [17] generates the remaining solutions. The original MOGLS—and, as far as we know, the implementation of Minella et al. [23]—uses the version of non-dominated sorting proposed by Deb et al. [7] 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 [19]. 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 [24]. 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 neighbourhood, 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 remainder 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. [23]. 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 [12]). Since the differences in implementation language and computation environment with respect to Minella et al. [23] 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) as TP+PLS. We perform 25 independent runs of each algorithm for each instance. We give in Table 4.6 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 algorithm is better in the Pareto sense (in the sense of $\lhd$, see Section 2.1) 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 4.7 between TP+PLS and MOGLS. Detailed tables with percentage values for each instance are available as supplementary material. 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 4.6 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 combination of objectives, and it cannot be said that any algorithm is clearly better than the other. Nevertheless, for all the remaining instances, Tables 4.6 and 4.7 show the excellent results of our TP+PLS algorithm, with very high percentages in its favor, whereas the percentages of MOSA and MOGLS are negligible.

Beyond the fact that TP+PLS often dominates MOSA and MOGLS (Tables 4.6 and 4.7), 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 2.5). Plots in Figures 4.6, 4.7 and 4.8 show some examples of these differences for three different problems. 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 themselves are important.

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 [12].

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 in this case. 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.

Table 4.6: 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. 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 .

	PFSP-($C_{\max}$, *SFT*)		*PFSP*-($C_{\max}$, *TT*)		*PFSP*-($C_{\max}$, *WT*)		*PFSP*-(*SFT*, *TT*)		*PFSP*-(*SFT*, *WT*)	
nxm	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA	TP+PLS	MOSA
20x5	4.66	5.83	6.1	1.34	14.95	0.18	10.19	26.31	0.02	20.15
20x10	1.87	9.2	0.07	0.26	0.02	0.06	0.19	0.63	0.03	0.07
20x20	0.13	1.23	1.27	1.57	1.99	2.32	3.63	5.55	4.2	10.09
50x5	89.49	0	84.33	0	79.22	0	98.13	0.08	33.67	0
50x10	72.92	0	63.17	0	63.24	0	94.07	0	20.53	0
50x20	75.94	0	61.11	0	63.01	0	5.79	0	14.72	0
100x5	84.97	0	70.5	0	67.12	0	93.66	2.54	9.72	0
100x10	76.94	0.05	69.86	0	37.49	0	95.38	0.58	16.84	0
100x20	73.17	0	63.29	0	23.81	0	97.35	0	15.31	0
200x10	18.04	0.16	24.5	0	4.15	0	91.77	3.72	0.02	0
200x20	15.16	0	37.83	0	0.25	0	78.23	6.28	1.04	0.02

Table 4.7: 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 MOGLS. 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 .

	PFSP-($C_{\max}$, *SFT*)		*PFSP*-($C_{\max}$, *TT*)		*PFSP*-($C_{\max}$, *WT*)		*PFSP*-(*SFT*, *TT*)		*PFSP*-(*SFT*, *WT*)	
nxm	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS	TP+PLS	MOGLS
20x5	18.35	0	26.39	0	28.36	0	57.52	0.14	26.64	0
20x10	18.79	0	11.52	0	5.46	0	20.7	0	17.63	0
20x20	13.82	0	19.58	0.13	25.47	0.06	20.44	0	18.83	0
50x5	39.45	0	58.11	0	75.38	0	99.29	0	95.1	0
50x10	60.28	0	70.46	0	81.08	0	96.76	0	98.21	0
50x20	74.77	0	74.44	0	70.3	0	97.85	0	97.75	0
100x5	24.97	1.12	87.79	0	76.11	0	91	4.5	42.3	0
100x10	62.43	0.27	93.02	0	79.17	0	96.21	0.04	97.4	0
100x20	83.88	0	83.42	0	68.14	0	99.55	0	98.57	0
200x10	9.55	0	81.6	0	60.03	0	94.73	1.88	28.91	0
200x20	33.37	0	83.3	0	35.45	0	96.72	0	83.19	0

4.4 Summary of results

We illustrated in this chapter the engineering of a multi-objective SLS algorithm for five bi-objective permutation flow-shop problems. We have proposed high-performing, single-objective algorithms for each of the objectives that make up the bi-objective problems, and for the weighted sum (scalarization) of pairs of objectives. These single-objective algorithms are one of several components of two complementary multi-objective SLS strategies that we studied. These strategies are TPLS and PLS, and we have proposed enhancements for each of them. In particular, we have proposed a weight setting strategy for TPLS (A-TPLS) that is both adaptive to the shape of the Pareto front and tries to maximise the quality of its output independently of its stopping time, that is, it improves the anytime behaviour of TPLS. We have empirically tested that, for the problems that concern us here, A-TPLS is a better strategy than a simpler restart strategy. Moreover, we have shown that A-TPLS is not very sensitive to the number of scalarizations solved.

We have also studied algorithmic components of PLS. We have shown that seeding PLS with good solutions leads to much better final quality without substantially increasing computation time. We have examined several neighbourhood operators for the bi-objective PFSPs tackled here, and decided that a combination of exchange plus insertion was the most advantageous. Finally, we have also taken into account the scenario in which PLS finishes before the given computation time limit. In such case, we extend PLS to examine, until it runs out of time, the neighbourhood of solutions previously found but not explored because they were dominated.

The insights gathered in the study of the above components have lead us to propose a hybrid SLS algorithm that combines the TPLS and PLS strategies, TP+PLS. Our 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 improves significantly upon the state-of-the-art.

Our results and other similar success stories recently reported in the literature [22] indicate the large potential of hybrid algorithms combining the TPLS and PLS frameworks. We believe that the engineering methodology followed here is effective and applicable to other bi-objective problems in order to reach high-quality algorithms.

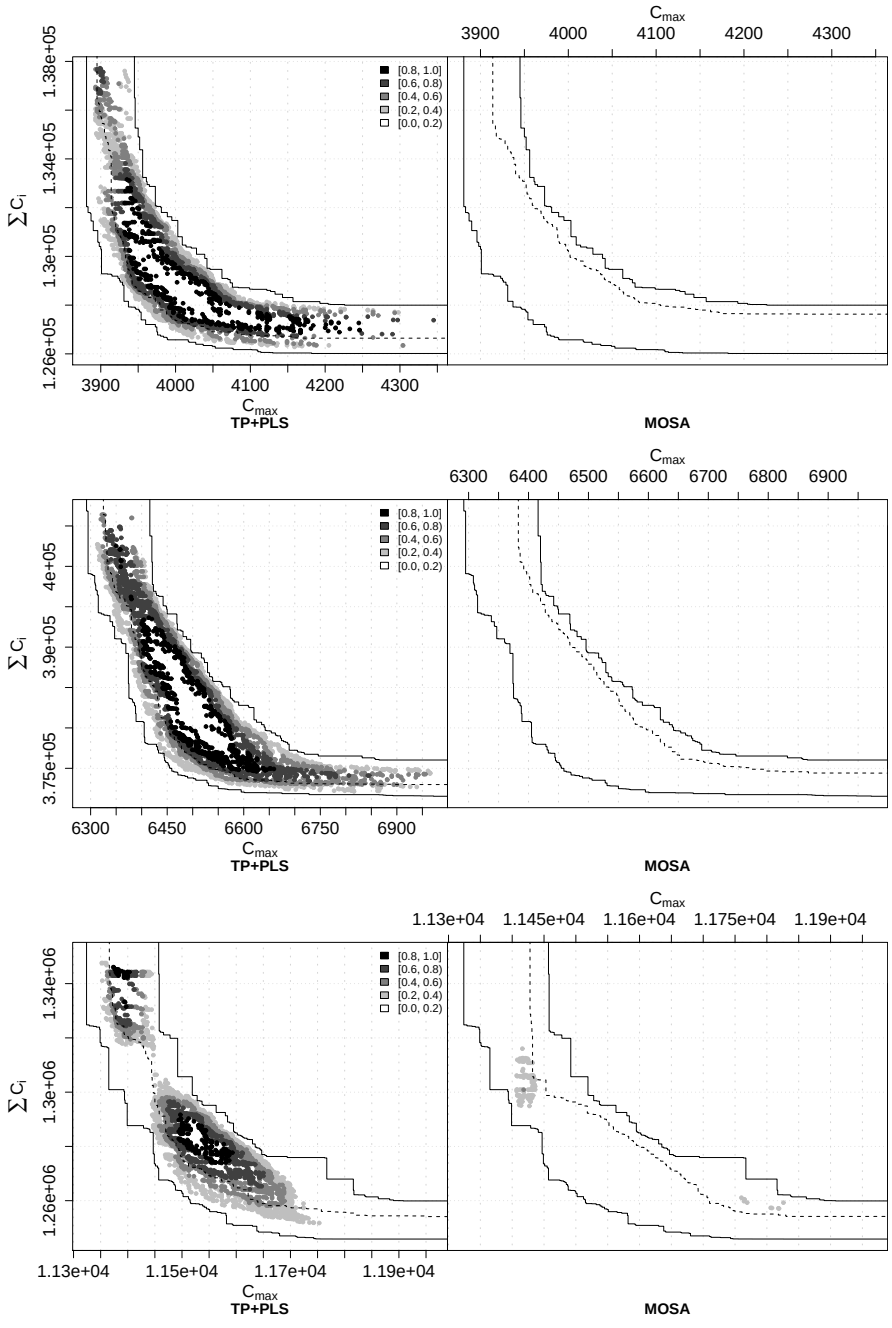

Figure 4.6: EAF difference for *PFSP*-($C_{\max}$, *SFT*) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

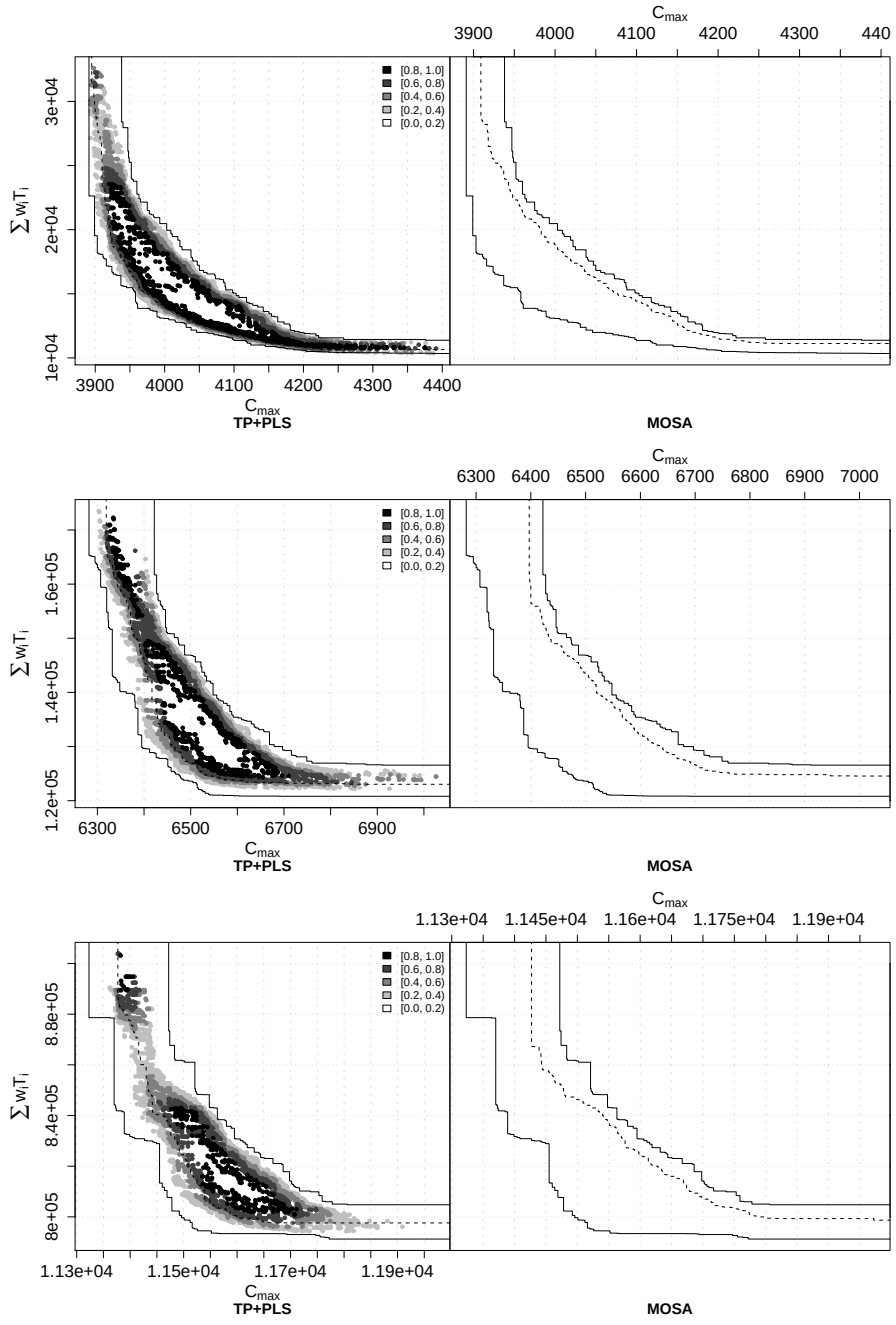

Figure 4.7: EAF difference for *PFSP*-($C_{\max}$, TT) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

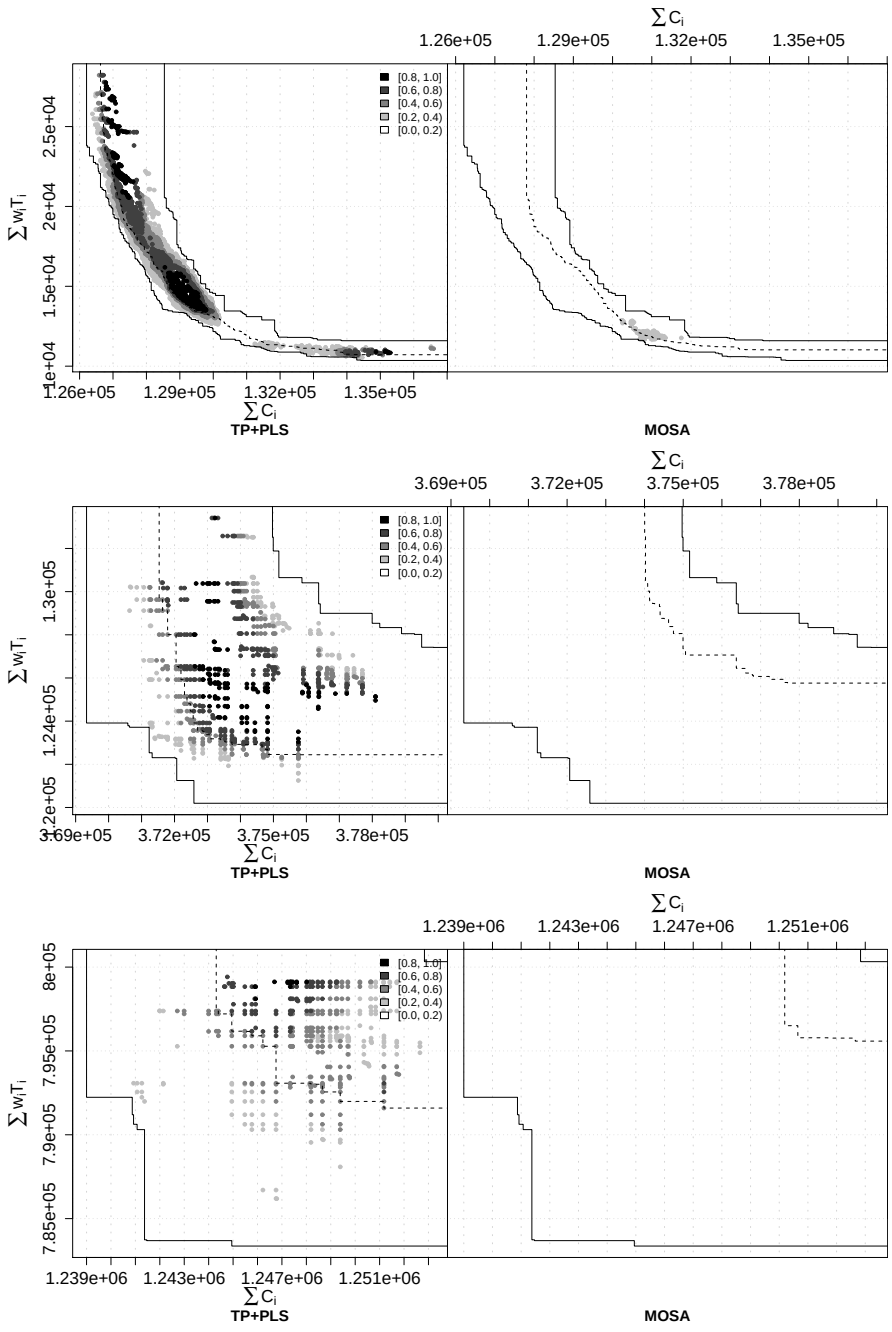

Figure 4.8: EAF difference for *PFSP*-(*SFT*, *TT*) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

In this report, we proposed numerous improvements to advance the state-of-the-art in multi-objective combinatorial optimisation.

In Chapter 3, we addressed a drawback of the Two-Phase Local Search (TPLS) framework, namely, that the number of scalarizations, and, thus, the overall computation time must be specified in advance; stopping the algorithm earlier results in poor results. In this sense, the weight setting strategies previously proposed for TPLS lead to weak *anytime* properties, and are not useful for practical applications where the computation time is not known in advance. We propose new weight setting strategies that can be stopped at any time during their execution, and the result would be a well-spread approximation of the Pareto front. That is, these strategies fulfill strong *anytime* properties [39]. Our second proposal is an adaptive strategy that can be further fine-tuned through a parameter θ. The two adaptive variants using two different values studied here, AN-TPLS ($\theta = 0$) and AF-TPLS ($\theta = 0.25$), outperform the classical TPLS strategies at any time during their execution. Thus, these adaptive strategies are perfectly suited for situations where the computation time available is not known in advance. Moreover, we believe AF-TPLS should be the strategy to be chosen even in "normal" situations, and, therefore, we use it in our hybrid algorithm for PFSP.

This hybrid algorithm is described in Chapter 4. The aim of TPLS is to extend the effectiveness of single-objective algorithms to tackle multi-objective problems. Therefore, the first step of our algorithm design, described in Section 4.1, has been to design high-quality algorithms for each of the single-objective problems, and for each weighted sum aggregation of the bi-objective problems. These algorithms are adapted from an Iterated

Greedy state-of-the-art for makespan minimisation [33]. They reach high-quality results.

Once we had very effective algorithms to minimize each objective, we then approached five multi-objective problems derived from each pairwise combination of objectives. The design of the multi-objective algorithm is described in Section 4.2. We used our new AF-TPLS variant as a framework to tackle multi-objective problems, relying on our single-objective algorithms as underlying components. In order to possibly yield better results, we implemented also Pareto Local Search (PLS). We carefully studied the different components of these two frameworks and their effects on the final quality.

From the insights obtained through this experimental analysis we design a final hybrid algorithm. To evaluate its efficiency, we reimplemented the two best existing algorithms for bi-objective PFSPs. Rarely more than one bi-objective combination has been studied in the same paper. By comparing our final algorithm to the two best existing ones, we show without ambiguity that our algorithm improves very clearly upon the state-of-the-art.

5.2 Future Work

Extension of A-TPLS to more than two objectives. We plan to extend the adaptive weight setting strategies for TPLS to three objectives in a first step, and more dimensions later. The original TPLS in 3-dimension uses Gray code, and the possible number of weights grows exponentially. Therefore, the lack of *anytime* property of the original TPLS increases exponentially with the number of objectives considered, and the design of *anytime* strategies could be particularly useful for more than two objectives. Moreover, it is well possible that such a strategy appears to be more effective than the original one for three objectives or more. An implementation for three objectives is currently ongoing, in such a way that its subsequent "instantiation" for different problems should be easy. We believe that, additionally to the *anytime* advantage, this could lead to better algorithms with respect to the resulting solution quality.

Exploration of tuning methods for multi-objective algorithms.
Most algorithms for optimisation problems have numerous parameters, either numerical or categorical ones. To reach state-of-the-art performance, the parameter settings of algorithms need to be fine-tuned, the best values being problem and instance dependent. To tackle real-life size instances of hard problems, one typically can rely only on approximate algorithms that often involve stochastic decisions to guide the search process. This stochasticity makes the task of finding the best parameter values even more challenging and this task can be seen itself as a hard optimisation problem.

Some methods have been already proposed for the automatic configuration of algorithms, and one of my main goals is to improve them.

One of the most known and most effective methods is iterative F-Race [4]. This automatic tuning algorithm repeatedly applies F-Race [5], new candidate configurations that are generated, taking into account the best performing configurations in previous iterations. However, implementation of this additional layout made its modification and/or extension hard and confusing. Moreover, it did not handle features such as conditional parameters and it makes even simple comparison between iterative F-Race and other tuning methods difficult or impossible. We reimplemented entirely the "iterative" layout of iterative F-Race in order to implement missing features, and make its use or adaptation by ourselves and other people much easier. A software package should be soon made publicly available. The next step will be to study all adaptations that could be required or pertinent for the automatic configuration of multi-objective algorithms. In a long-term view, other methods will be studied and new ones explored in order to obtain effective automatic tuning procedures for multi-objective problems. As a side effect, this could lead to some improvements also for the tuning of single-objective algorithms.

General research directions. The hybrid algorithms we designed for bi-objective PFSPs reach very high performance, improving clearly upon the previous state-of-the-art. We believe that hybridization of search methods based on a component-wise acceptance criterion (CWAC search model) and methods based on a scalarized acceptance criterion (SAC search model) can be very effective, in general. To the best of our knowledge, such hybridizations has not been properly explored and we believe that they are likely to improve upon the upon the state-of-the-art for several well-known multi-objective problems,. The engineering methodology we followed (see Chapter 4) appears to be very efficient and could be applied to some of these problems. However, such a methodology of systematic study of each component is costly in human efforts. Additionnaly, it often does not take properly into account the possible interactions between the components.

A very suited way to save human time is the use of an automatic tuning procedure. This is also a good way to explore a much larger design space, and, in doing so, to reach a better final quality and to deduce useful information from interactions that can appear.

Therefore, these two directions for future work, the design of hybrid algorithms for multi-objective problems and the study of automatic tuning and configuration procedures for multi-objective algorithms can be seen as complementary and oriented toward one same goal: to design faster and in a more automatized way new multi-objective algorithms that reach state-of-the-art quality or improve upon it.

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.

Bibliography

[1] Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979.

[2] J. E. Arroyo and V. A. Armentano. A partial enumeration heuristic for multi-objective flowshop scheduling problems. *Journal of the Operational Research Society*, 55(9):1000–1007, 2004.

[3] J. E. Arroyo and V. A. Armentano. Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, 167(3):717–738, 2005.

[4] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *HM 2007*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, Heidelberg, 2007.

[5] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[6] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, third edition, 1999.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):181–197, 2002.

[8] J. Du and J. Y.-T. Leung. Minimizing total tardiness on one machine is NP–hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.

[9] J. Dubois-Lacoste. 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.

[10] J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In *Hybrid Metaheuristics – 6th International Workshop, HM 2009*, volume 5818 of *Lecture Notes in Computer Science*, pages 100–114. Springer, Heidelberg, 2009.

[11] J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Adaptive "anytime" two-phase local search. In *Learning and Intelligent Optimization*, volume 6073 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 2010. Accepted.

[12] J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Supplementary material: A Hybrid TPLS+PLS Algorithm for Bi-objective Flow-shop Scheduling Problems. `http://iridia.ulb.ac.be/supp/IridiaSupp2010-001`, 2010.

[13] M. Ehrgott. *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 2000.

[14] M. Ehrgott and X. Gandibleux. Approximative solution methods for combinatorial multicriteria optimization. *TOP*, 12(1):1–88, 2004.

[15] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *IEEE Congress on Evolutionary Computation*, pages 1157–1163. IEEE Press, July 2006.

[16] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[17] F. Glover. *Artificial Evolution*, chapter A Template for Scatter Search and Path Relinking, pages 1–51. Lecture Notes in Computer Science. Springer, Heidelberg, 1998.

[18] V. Grunert da Fonseca, C. M. Fonseca, and A. O. Hall. Inferential performance assessment of stochastic optimisers and the attainment function. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello, and D. Corne, editors, *Proceedings of EMO 2001*, volume 1993 of *Lecture Notes in Computer Science*, pages 213–225. Springer, Heidelberg, 2001.

[19] M. T. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, 2003.

[20] D. S. Johnson. Optimal two- and three-stage production scheduling with setup times included. *Naval Research Logistics Quarterly*, 1:61–68, 1954.

[21] M. López-Ibáñez, L. Paquete, and T. Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuß, editors, *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010. To appear.

[22] T. Lust and J. Teghem. Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 2009. To appear.

[23] G. Minella, R. Ruiz, and M. Ciavotta. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20(3):451–471, 2008.

[24] J. N. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers & Operations Research*, 7(1-2):55–66, 1980.

[25] M. Nawaz, E. Enscore, Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA*, 11(1):91–95, 1983.

[26] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[27] L. Paquete. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis*. PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2005.

[28] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In X. Gandibleux et al., editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*, pages 177–200. Springer, 2004.

[29] L. Paquete and T. Stützle. A two-phase local search for the biobjective traveling salesman problem. In C. M. Fonseca et al., editors, *Proceedings of EMO 2003*, volume 2632 of *Lecture Notes in Computer Science*, pages 479–493. Springer, Heidelberg, 2003.

[30] L. Paquete and T. Stützle. Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 29–1—29–15. Chapman & Hall/CRC, 2007.

[31] L. Paquete and T. Stützle. Design and analysis of stochastic local search for the multiobjective traveling salesman problem. *Computers & Operations Research*, 36(9):2619–2631, 2009.

[32] C. Rajendran. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73, 1993.

[33] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[34] É. D. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[35] É. D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[36] E. Vallada, R. Ruiz, and G. Minella. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373, 2008.

[37] T. K. Varadharajan and C. Rajendran. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research*, 167(3):772–795, 2005.

[38] H. Woo and D. Yim. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers & Operations Research*, 25(3):175–182, 1998.

[39] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[40] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[41] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.