ECOLE
POLYTECHNIQUE
DE BRUXELLES

UNIVERSITÉ LIBRE DE BRUXELLES
Ecole Polytechnique de Bruxelles
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Anytime Local Search for Multi-Objective Combinatorial Optimization: Design, Analysis and Automatic Configuration

Jérémie DUBOIS–LACOSTE

Promoteur de Thèse :

Dr. Thomas Stützle

Co-promoteurs de Thèse :

Dr. Mauro Birattari
Dr. Manuel López-Ibáñez

Thèse présentée en vue de l'obtention du titre de Docteur en Sciences de l'Ingénieur
Année Académique 2013/2014

Abstract

Our world is becoming increasingly complex and many systems are key to the well-functioning of our modern society. They can be any sort of technological systems, processes, logistical organization in the industry, distribution of goods, planning and scheduling of trains and planes, design of a city road map, control of traffic signals, and so on.

In all these situations, the decisions taken have a strong impact on efficiency, be it a matter of time, resources, or any other element to be optimized. Once the problems have been fully identified and the objective has been clearly established, the actual task of finding a good (or the optimal) solution is typically very hard for humans. In fact, even assisted by computers an entire class of problems remains very hard to solve to optimality (problems that in theoretical Computer Science have been identified as $\mathcal{NP}$-hard). In this case, the practical goal to be achieved is often to find satisfactory solutions in reasonable time. Algorithms designed for this task are called *heuristic* optimization algorithms.

At the beginning of the algorithmic optimization field, most of the research efforts were devoted to problems with a single objective to be optimized. However, many problems in real situations can be considered as having several objectives. For instance the schedule of public transports can aim at (i) minimizing the time required for users to reach their destination, or (ii) minimizing the cost of the transportation system so that it remains economically profitable or at least feasible. In this example, as in most real-world problems, the objectives that are relevant are conflicting with each other so that optimizing one is likely to worsen another. Therefore, research in multi-objective optimization aims at finding efficient methods and algorithms that take into account several objectives, and return a set of solutions that are trade-offs between objectives. In

this thesis, we are interested in the design and study of heuristic algorithms for multi-objective optimization.

Two aspects are key to our work and make it novel in the multi-objective optimization field: the use of *automatic configuration* techniques, and the consideration of the *anytime behavior*. Automatic configuration techniques free the human designer from the time-consuming task of setting parameter, and have already been successfully applied to single-objective algorithms. As we show in this thesis, they are both applicable and desirable for multi-objective algorithms as well. The anytime behavior of an algorithm is its ability to return as high-quality solutions as possible at any moment of its execution. Having algorithms that have a good anytime behavior is highly desirable in situations where the computation time is unknown a priori or when it changes each time the algorithm is launched. The optimization literature is mostly focused on the quality reached by an algorithm after a given time defined a priori, and the outcomes of such studies may poorly extend to situations where the anytime behavior is relevant. In this thesis, besides studying the anytime behavior of multi-objective algorithms, we also test the application of automatic configuration techniques in order to obtain automatically good "anytime" algorithms.

All algorithms are evaluated on well-known combinatorial problems by means of statistical and graphical tools. Our results, published in several international journals and conferences, improve over the state-of-the-art for relevant and widely studied problems, in terms of anytime behavior and also in terms of final quality.

Acknowledgments

First of all I would like to express my gratitude to Dr. Thomas Stützle for its outstanding supervision during these years. It was a great pleasure for me to share his passion and knowledge. I also wish to deeply thank Dr. Manuel López-Ibáñez with whom I had many interesting discussions and joint works.

IRIDIA is a wonderful place and I feel truly honored to be a part of it. This laboratory is unique by all the people that made and makes it. I was lucky to meet all those whom I met, some here today and some that left long ago already. You have vastly broadened my horizon. I am profoundly grateful to all Iridians for this, and for all the nice moments together.

Additionally to Thomas, I want to thank the "backbone" of IRIDIA for maintaining a nice environment and high-quality standards of teaching and research: Dr. Mauro Birattari, Pr. Hugues Bersini and Pr. Marco Dorigo.

During my PhD, I received support from the European Commission through the project MIBISOC. I am grateful to all people who managed this project, in particular Pr. Oscar Cordón who did an amazing job with a truly communicative enthusiasm. I do not think I need to express my warm sentiment to my fellow mibisockers, I will make sure to travel wherever they are to remind them from time to time.

Rossella, thank you for your patience and for sharing me so much with "my algorithms". Somehow, we made it together.

A mes parents: merci pour votre confiance depuis toujours. Sans elle et la liberté qui va avec, cette thèse n'aurait pas vu le jour.

Jérémie

This work has been mainly funded by the European Commission under the **MIBISOC** project (Grant Agreement: 238819), within the action Marie Curie Initial Training Network of the Seventh Framework Program (7FP).

I also acknowledge support from the **META-X** project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium.

“ À Anne et Vincent,

Camille, Christiane, Jacques, Jacques”

Contents

Chapter 1

Introduction

Many problems that arise in today's world benefit from some form of optimization to reduce costs, increase efficiency, or in general, to simply *do better*. Thus, in a wide sense, optimization can contribute to increase the well-being of our society. Optimization problems arise in many different fields of high social or economic importance. An example is the topological design and schedule of a public transport network in a city, a task whose importance can be immediately grasped by anybody. Another example is the assignment of planes to airports gates, a crucial aspect required for airports to sustain the ever-increasing demand for aerial transportation. Other examples are the reconstruction of MRI models in three dimensions to support diagnosis in hospitals, the scheduling of classes in schools or universities, the organization of production lines in factories, the design of environment-friendly buildings, the scheduling of trains at different scales from a station to a country, the loading of containers on cargo boats, the routing of vehicles for pick-up or delivery of goods, the design of water-supply networks for cities, the sustainable use of forest resources, the design of airplanes, the routing of information in computer networks, etc. Many of these problems are *combinatorial*, because possible solutions to them are combinations of a discrete set of items, similarly to a puzzle whose pieces could be arranged in different manners.

In all these situations, the decisions taken or the configurations chosen have an impact on the systems' performance. Once the problem under concern has been identified, understood and modeled, once the objective has been clearly established, the task that typically is hard for humans is that of finding a solution to the problem at hand that is proven to be the best possible, that is, the optimal solution.

In fact, for many problems this task remains very hard even when relying on high computing power as it is today available. This is not related to the programming skills of the person facing the problem, but it is an intrinsic property of a whole class of problems, called $\mathcal{NP}$-hard problems (Garey and Johnson, 1979). Despite being unproven at the time of writing (the demonstration being considered as one of the most challenging open problems in mathematics and informatics), a conjecture that is widely accepted is fundamental to this work. This conjecture states that there does not exist an algorithm that can always find the optimal solution in polynomial time w.r.t. to the size of the instance of an $\mathcal{NP}$-hard problem. In other words, the time necessary to find the optimal solution in the worst case is exponential in the size of the instance, which makes an optimality proof infeasible in practice as real-world instances are often too large.

Therefore, for $\mathcal{NP}$-hard problems, often one cannot rely on exact algorithms that deliver the optimal solution, but instead one must try to find a solution as good as possible (not necessarily optimal), in acceptable time. Techniques that are designed for this task are called *heuristic*; they can deliver quickly solutions to the problem but without the proof of optimality.

For several decades after the birth of the algorithmic optimization field, problems were considered that had only a single objective to be optimized. This is probably due to the inherent difficulty of considering several objectives at the same time: the objectives are typically conflicting and therefore optimizing them at the same time is not possible. In this case, a common approach is to tackle a problem in the *Pareto sense*, aiming at returning a set of solutions that are different trade-offs between the objective values. If we take the example of designing an energy-efficient building, the objective to be optimized could be the heat diffusion of this building, so energy could be saved to warm it in winter or refresh it in summer. However, having this sole objective in mind could lead to solutions that are possibly far from anything realistic or desirable. In reality, another objective can not be neglected: the cost of the construction. The aim when considering these two goals is then to find a whole set of possible trade-off designs between a very efficient energy-saving design, and a very cheap design.

Approaches for multi-objective optimization fit into two radically different paradigms. The first paradigm is to tackle several single-objective problems, whose solutions found are different solutions to the original, multi-objective problem. Considering

the example of the building design, two single-objective problems could be tackled independently: first a problem that focuses solely on the price to obtain the cheapest possible design, second a problem that focuses solely on the energy-efficiency of the design for instance by using a plant-covered roof. After considering these two problems with a single-objective method, one would obtain two different solutions that would be also valid for the multi-objective problem. A third solution could be obtained by giving an equal importance to each of the two objectives, defining a third single-objective problem. The process of mapping the multi-objective problem into single-objective ones is called *scalarization*, and the idea behind this type of methods is to rely on scalarizations to tackle multi-objective problems. This kind of approaches are hereafter called *scalarization-based* methods. The second paradigm is to tackle problems in a purely multi-objective way. The search process is performed exclusively in a multi-objective space, and solutions are compared using exclusively the Pareto dominance relation. Hereafter, such approaches are called *dominance-based*.

In this thesis, we focus on heuristic algorithms for multi-objective optimization. We study canonical algorithms of each paradigm for designing multi-objective optimization algorithms. For each paradigm we propose new algorithms that improve over previous ones, we combine algorithms together in a generic framework, and we demonstrate the efficiency of this framework by tackling well-known benchmark problems.

Approximate algorithms have *parameters* that strongly impact their behavior. In practice, the setting of the parameters to a proper value has a huge impact on the efficiency of an approximate algorithm on a given problem. For decades, optimization algorithm designers, including researchers, have set the parameter values manually, which is a time-consuming task involving a cycle of trial and error, yielding results with low reliability and prone to biases. A recent and nowadays "hot" trend in designing optimization algorithms, and especially heuristic ones, is to use automatic configuration techniques to set the parameter values in an automatic manner. The advantage is that the repetitive tasks are given to computers, tasks at which they are often better than humans; humans in turn can focus on tasks on which they are very good at, that is, all the tasks that involve creativity. This mind-shifting design approach is followed in large parts of this thesis, and in the appendix we present an algorithm for automatic configuration that we implemented and on which we rely for design tasks throughout the thesis.

Another important aspect of our approach to multi-objective optimization, and optimization in general, is to value the *anytime behavior* of algorithms. In many real-world situations, the computation time available before a solution needs to be given can vary, or even be unknown a priori. In such cases, algorithms that deliver results of the highest possible quality at any time during their execution are desirable. However, the goal of having good anytime behavior is rarely considered in the design of algorithms. In fact, in the literature, algorithms are very often designed and compared for a single computation time target.

In the rest of this chapter, we highlight the goals and the contributions of this thesis, we present the list of scientific publications that arose from this work, and we detail the structure of the rest of the thesis.

1.1 Goals of this Thesis

In this thesis, we study in depth the behavior and effectiveness of two canonical algorithms based on local search that belong to each of the two main search paradigms for multi-objective optimization with local search. The first one, Two-phase Local Search (TPLS), is based on solving scalarized problems. The second one, Pareto Local Search (PLS), is a stochastic local search algorithm that explores a search space using Pareto dominance to compare solutions.

Despite being often rather effective, each of these two algorithms has weaknesses. In this thesis, we propose new algorithms that improve over the original version of TPLS and PLS. In particular, we improve them from an anytime optimization perspective.

Our goal is also to highlight the importance of automatic configuration as a new promising perspective for the design of optimization algorithms. We promote its development by the implementation of easy to use and effective tools for automatic configuration, and by using it in many practical configuration tasks.

1.2 Contributions of this Thesis

This thesis has led to a number of contributions. The most important ones are summarized here.

- We demonstrate the potential of combining the two *paradigms* scalarization-based and dominance-based for multi-objective combinatorial optimization.
- Often, optimization algorithms are developed with a sole predefined computation time in mind, without considering the quality reached prior or after this computation time. On the contrary, an algorithm with a good anytime behavior can be stopped at any time and still deliver high-quality results, an important aspect for every real-world situation where the computation time available is unknown in advance. This is rarely done in the optimization literature, and a contribution of this thesis is to highlight the importance of anytime behavior in multi-objective context.
- A simple algorithm, belonging to the scalarization-based paradigm, that serves as the basis of numerous effective algorithms for bi-objective optimization is TPLS. In this thesis, we propose a new algorithm based on TPLS, that has the advantage to direct the search process based on the state of the current results. We demonstrate the potential of this new algorithm by means of a careful experimental analysis. In particular, we show that this new algorithm has a strong anytime behavior.
- Another simple algorithm, belonging to the second paradigm, is PLS. We propose an in-depth study of its different components, and we propose new variants that improve over the original PLS algorithm in terms of anytime behavior.
- A new aspect of this thesis is the demonstration of the ability of automatic algorithm configuration techniques to deal with the configuration of frameworks for multi-objective optimization and to obtain new state-of-the-art multi-objective optimizers. As a side contribution, we implemented a public version of the `irace` configuration software, released under the GPL license. Apart from our own work, this software has been used in numerous research papers.
- By combining all the aspects studied in the thesis, that is, the new algorithms proposed for each paradigm, their combination, and the use of automatic config-

uration techniques, we propose new state-of-the-art algorithms for well-known and widely-studied benchmark problems from the combinatorial multi-objective optimization community.

1.3 Publications

This thesis has led to a number of publications in international journals, book chapters, international conferences, or articles that are currently under review. The list below reports all major publications, some of which being just accepted or submitted at the time of writing.

International Journals

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011a). Improving the anytime behavior of two-phase local search. *Annals of Mathematics and Artificial Intelligence*, 61(2):125–154

  In this paper, an extended version of the LION 2010 (Dubois-Lacoste *et al.*, 2010c) conference paper, we propose a new algorithm inspired by the original Two-phase Local Search and by exact algorithms. The results demonstrate that the proposed algorithm not only has a better anytime behavior but also improves the final results that can be obtained.

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011b). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236

  This paper presents a combination of the scalarization-based and the dominance-based paradigms for solving bi-objective permutation flowshop scheduling problems. More precisely, the original TPLS and PLS algorithms are combined with newly designed effective algorithms to tackle each objective. Experimental results demonstrate the large improvement over previous state-of-the-art algorithms.

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2014). Anytime Pareto local search. *European Journal of Operational Research*. Submitted

  This paper presents an in-depth study of the algorithmic components of Pareto Local Search, some new variants and their impact on the anytime behavior of the resulting PLS-inspired algorithms. The best variants found improve strongly over the original PLS algorithm.

- F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. (2014b). Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*. Accepted subject to minor revisions

  This paper focuses on automatic configuration used for the actual design of algorithms (rather than just the setting of a subset of parameters). Another popular approach for this is to use Grammatical Evolution, and this paper compares the two approaches in terms of the resulting algorithm effectiveness.

Book Chapters

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2013b). Combining two search paradigms for multi-objective optimization: Two-phase and Pareto local search. In *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 97–117. Springer, Berlin/Heidelberg

  In this book chapter we propose a study of the relevant literature for multi-objective combinatorial optimization seen with the dichotomy between the scalarization-based and the dominance-based paradigms in mind. In particular, we focus on algorithms that are hybrids of both paradigms and we present a summary of experimental results demonstrating the high potential of such hybrids.

International Peer-reviewed Conferences

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2009). Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In

Hybrid Metaheuristics, volume 5818 of *Lecture Notes in Computer Science*, pages 100–114. Springer, Heidelberg, Germany

This paper presents experimental results obtained by combining TPLS and PLS for bi-objective permutation flowshop problems, and a preliminary comparison to state-of-the-art algorithms.

- **This paper received the best paper award of the LION 4 conference:**
  J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2010c). Adaptive "anytime" two-phase local search. In *Learning and Intelligent Optimization, 4th International Conference, LION 4*, volume 6073 of *Lecture Notes in Computer Science*, pages 52–67. Springer, Heidelberg, Germany

  This paper proposed a study and improvement of the TPLS algorithm. In particular, we showed how to improve TPLS anytime behavior. The quality of the final results has been improved as well.

- **This paper was nominated for the best paper award of the self-* track of GECCO 2011:**
  J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011c). Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2019–2026. ACM Press, New York, NY

  This paper is an experimental study that shows the potential of automatic configuration techniques for the design of multi-objective algorithms.

- Y. S. G. Nashed, P. Mesejo, R. Ugolotti, J. Dubois-Lacoste, and S. Cagnoni. (2012). A comparative study of three GPU-based metaheuristics. In *PPSN 2012, Part II*, volume 7492 of *Lecture Notes in Computer Science*, pages 398–407. Springer, Heidelberg, Germany

  This paper focuses on the design of meta-heuristic implementations for parallel environments, that use the *CUDA* language extension of `C` to exploit the potential of Graphical Processing Units (GPUs). The experimental study relies on automatic configuration techniques, namely the `irace` software, to perform a fair comparison of the different algorithms considered.

- J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2012). Pareto local search algorithms for anytime bi-objective optimization. In *Proceedings of EvoCOP 2012 – 12th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 206–217. Springer, Heidelberg, Germany

  In this paper, we present preliminary results on the study of PLS and its improvement in terms of anytime behavior.

- F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. (2013). From grammars to parameters: Automatic iterated greedy design for the permutation flowshop problem with weighted tardiness. In *Learning and Intelligent Optimization, 7th International Conference, LION 7*, volume 7997 of *Lecture Notes in Computer Science*, pages 321–334. Springer, Heidelberg, Germany

  This paper presents a comparison of Grammatical Evolution and automatic configuration by use of the `irace` software, to design automatically algorithms for permutation flowshop problems.

- A. Valsecchi, J. Dubois-Lacoste, T. Stützle, S. Damas, J. Santamiaría, and L. Marrakchi-Kacem. (2013). Evolutionary medical image registration using automatic parameter tuning. In *Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)*, pages 1326–1333. IEEE Press, Piscataway, NJ

  This paper applies an automatic configuration technique, namely `irace`, to algorithms designed for the registration of medical images, in order to compare the different algorithms in an unbiased way.

- F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, M.-E. Marmion, and T. Stützle. (2014a). Algorithm comparisons by automatically configurable metaheuristic frameworks: a case study using flow-shop scheduling problems. In *Hybrid Metaheuristics*, Lecture Notes in Computer Science. Springer, Heidelberg, Germany. Accepted

  This paper explores the use of automatic configuration tools as a mean to compare algorithms with a focus on flowshop scheduling problems.

1.4 Structure of this Thesis

In Chapter 2, we present the basic concepts that underlie this work, and review the relevant previous work in the literature. In Chapter 3, we focus on algorithms based on the scalarization principle, and in particular the canonical representative that is Two-phase Local Search. In Chapter 4, we study the performance obtained by the combination of both search paradigms, by designing an hybrid algorithm that combines Two-phase Local Search and Pareto Local Search. Chapter 5 focuses on improving the Pareto Local Search algorithm, studying different ways to do so and evaluating their effects on the anytime behavior of PLS. In Chapter 6, we make use of automatic configuration methods to design multi-objective algorithms that improve over those obtained with manual settings, and, thus, improve over the state-of-the-art. In particular, we demonstrate the ability of these methods to obtain significantly better results both in terms of final quality and anytime behavior. Finally, in Chapter 7, we summarize the contributions of this thesis, and we discuss promising directions for future research.

Chapter 2

Background

This chapter introduces the basic concepts and definitions necessary to give the context of our work. We start by presenting the characteristics of combinatorial optimization problems, the class of problems on which we focus. We present the traveling salesman problem, that we use as a benchmark problem to assess the performance of the methods we designed in this thesis. Next, we introduce the concept of local search. Then, we explain the fundamental differences between multi-objective optimization and the "classical" single-objective optimization, and the definitions necessary in the multi-objective context. Multi-objective optimization offers many challenges, one of which is the assessment of the quality of the output that multi-objective algorithms produce, and we present the methods that we use for this task. We then present the two main paradigms that can be used to tackle problems in the multi-objective context by heuristic algorithms, and we will give a literature overview of the works that are relevant to ours. In particular, for each paradigm, we will present an algorithm that is prototypical for it. Finally, we will present the concept and the challenges behind the *anytime* approach to optimization, which is key to our work.

2.1 Combinatorial Problems

Combinatorial problems are ubiquitous in many different fields such as engineering, computer science, logistics, bioinformatics, etc. Combinatorial problems involve finding orderings or groupings of a set of discrete elements. These discrete elements are the

components of the possible solutions, called *candidate solutions*. Candidate solutions are potential solutions that are valid for the problem at hand, but not necessarily the best possible ones.

A *problem* is a general abstraction of the input data that define a particular type of constraints that potential solutions must fulfill in order to be considered a valid candidate solution. In an optimization problem, additionally we are given a way to measure the objective value of candidate solutions, and a goal, which is to find a solution with the smallest (for a minimization problem) or greatest (for a maximization problem) objective value. In this thesis, we consider minimization problems, but all the methods that we develop can be applied to maximization problems as well. In fact, a maximization problem can be easily mapped to a minimization version of the same problem by multiplying the objective function by "-1".

An *instance* of a problem is one possible instantiation of it, describing the specific data necessary to derive the objective value of a solution from its components.

For a given instance, the cardinality of the set of candidate solutions is often exponential in the instance size, which is typically described by the number of solution components or some abstraction thereof.

2.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a prototypical combinatorial problem, and one of the most famous benchmark problems (Lawler *et al.*, 1985; Reinelt, 1994; Johnson and McGeoch, 1997; Applegate, 2006). The central role of the TSP is due to its complexity (it belongs to the class of $\mathcal{NP}$-hard problems (Garey and Johnson, 1979)) despite being a problem that is very easy to understand, and due to its relevance in practice as it arises in many different applications.

In the TSP, we are given a complete weighted graph $G = (V, E, C)$ with a set of vertices $V = \{v_1, \ldots, v_n\}$, a set of edges $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ that fully connects the vertices with associated costs $C = \{c_{ij}\}$. The goal is to find a Hamiltonian cycle (hereafter called a "tour") that has a minimum sum of edge costs. A candidate

solution can be represented as a permutation of the vertices, and the sequence of the vertices in the permutation defines the order in which they are visited.

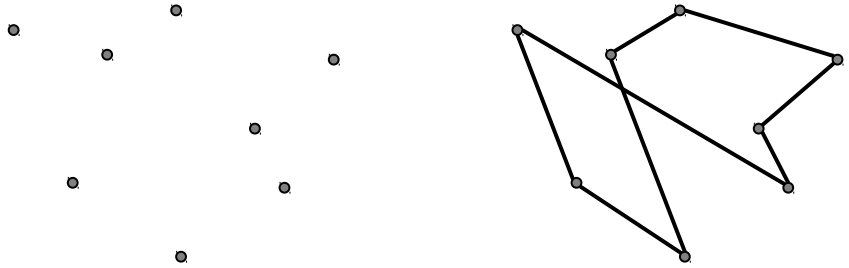

Figure 2.1: On the left is a graphical illustration of one TSP instance. This instance is Euclidean, as the vertices are points distributed in the Euclidean plane. The edge costs, which are not shown explicitly here, are the Euclidean distances between points. On the right, a candidate solution is given.

TSP instances often have certain characteristics. A common one is the symmetry of the costs between pairs of vertices: if for all pairs of edges (v_i, v_j) and (v_j, v_i) it holds $c_{ij} = c_{ji}$, then the TSP instance is called symmetric. Instances can be defined such that vertices are associated with points in the Euclidean plane, and the costs between two vertices is the Euclidean distance between the associated points. Figure 2.1 shows graphically an Euclidean TSP instance and one possible solution for it.

2.3 Local Search

In this thesis, we focus on algorithms that are based on *local search*. Local search algorithms, and, in particular, stochastic local search (SLS) ones, play an important role in optimization whenever an exact algorithm is not useful in practice (Hoos and Stützle, 2005). They underlie many state-of-the-art heuristic algorithms for a wide variety of problems. Local search algorithms are based on the idea that solutions can be improved little by little by applying small modifications. In this way, one can possibly find a better quality solution and continue the process from there, trying to apply small modifications to the new solution to find even better ones.

This can be formally described as follows. Let $\mathbb{S}$ be the space of candidate solutions. Applying a small modification to a solution $s \in \mathbb{S}$ is done by a *neighborhood operator* N.

The term "neighborhood" refers to the proximity in $\mathbb{S}$ of the initial solution s and all the solutions obtained by applying N to s. The space of solutions $\mathbb{S}$, combined with N forms what is called a neighborhood graph whose vertices are solutions, and an edge exists between two solutions s and s', iff s' can be obtained by applying N to s. SLS algorithms explore the space of solutions by following a path in the neighborhood graph from solution to solution (changing the incumbent solution to one of its neighbors is often called a "move").

An important aspect of neighborhood operators is the concept of *delta-evaluation* (also called incremental updates). The evaluation of a solution is the process of computing its objective value from its components. It can be computationally expensive and requires typically a significant part of the time devoted to the optimization. Delta-evaluation consists of computing the objective value of a solution s' by using the objective value of s and accounting for the solution components in which s and s' differ. The difference of the computational complexity between a full evaluation and a delta-evaluation can be large, from a constant factor to being proportional to a polynomial or even exponential time of the instance size, and is often a key point to consider in the design of effective algorithms.

2.3.1 Neighborhood Operators: an Example for the Traveling Salesman Problem

In this section, we give an example of the concept of neighborhood operators using the TSP.

Probably the most common type of neighborhood operators for the TSP are those that make use of k-*exchange* moves: two solutions s and s' are neighbors if s' can be obtained from s by deleting a set of k edges and adding a new set of k edges to re-wire the resulting fragments into a new tour.

A graphical illustration of a 2-exchange move is shown in Fig. 2.2 in a schematic way (top) and applied to the solution presented in Fig. 2.1 (bottom). In the latter case, the new solution produced improves over the original one.

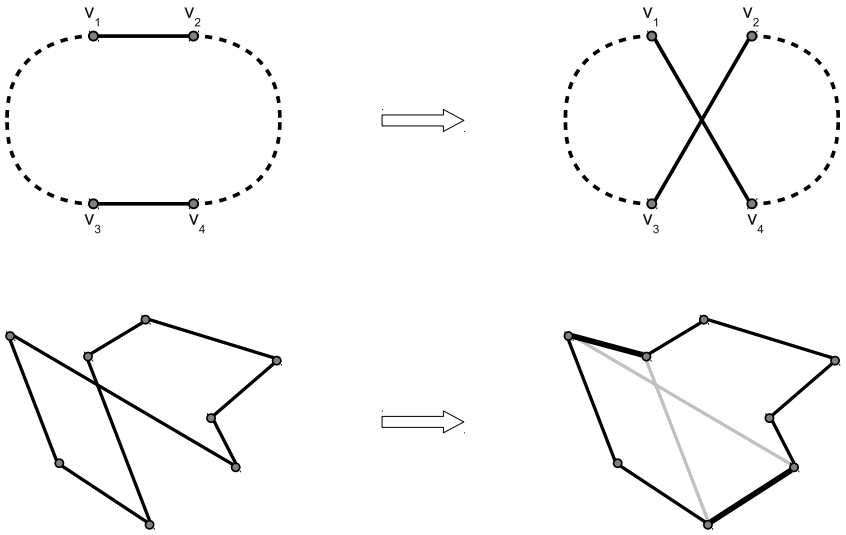

Figure 2.2: On top is a schematic representation of a 2-exchange move. Two edges (v_1, v_2) and (v_4, v_3) are removed from the original solution on the left, and the two resulting fragments are wired together by adding the two new edges (v_1, v_4) and (v_2, v_3) (this is the unique way to connect the two fragments obtaining a tour different from the original one). Note that one of the fragments is then reversed in the tour. On the bottom, a 2-exchange move is applied to the solution presented in Fig. 2.1. The new solution shown on the right is then better than the initial one.

Neighborhood operators based on k-exchange moves iteratively consider all possible combinations of edges for the moves, which defines a set of neighbors to the incumbent solution. A solution obtained by use of these operators, which cannot be improved anymore by applying any move, is said to be k-optimal. For this reason, such neighborhood operators are called *k-opt*. The most common ones are the *2-* and *3-opt* neighborhood operators. The computational complexity to produce the whole neighborhood with a k-opt operator is in $O(n^k)$, where n is the number of vertices of the instance considered, thus, higher values of k quickly make neighborhood exploration costly.

The advantage of such moves is that the delta-evaluation of a solution can be done easily. Applied to a solution $s = [\ldots, v_1, v_2, \ldots, v_4, v_3, \ldots]$ whose objective value $f(s)$ is known, a 2-opt move applied to s by removing the edges (v_1, v_2) and (v_4, v_3) generates a solution s' with: $f(s') = f(s) - (c_{12} + c_{34}) + (c_{14} + c_{23})$. That is, the delta-evaluation of a solution can be done in constant time while the full evaluation requires $O(n)$ atomic operations, where n is the length of a solution.

2.3.2 Pivot Rule

The pivot rule defines how a neighborhood operator is used to define the path of solutions followed by a local search algorithm in the search space.

The *first-improvement* pivot rule applies a move as soon as the neighborhood examination finds an improving one. In other words, a neighbor that is found to be better than the incumbent solution immediately becomes the new incumbent one.

With the *best-improvement* pivot rule, on the other hand, first all neighbors are produced by a neighborhood operator, and then only the best one is chosen to become the new incumbent solution.

The trade-off between the first and the best-improvement pivot rules are the following. While the first-improvement rule moves faster than the best-improvement rule, the improvement by each of its steps is typically smaller; thus a first-improvement rule usually needs more steps to be applied to reach a local optimum.

The most efficient pivot rule for a given algorithm and a given problem is typically unknown a priori, and some experimental evaluation is required to determine the most appropriate one in practice (Hoos and Stützle, 2005).

2.4 Metaheuristics

Neighborhood operators are useful to explore the search space locally around the incumbent solution. If at each step only improving moves are accepted, we obtain a simple local search algorithm that is called *iterative improvement*. It follows a path in the search space by replacing the incumbent solution with one of its neighbors that is better.

Algorithm 1 presents an iterative improvement algorithm in pseudo-code for a minimization algorithm. The solution finally returned is called a *local optimum* as none of its neighbors can improve over it. Unfortunately, local optima can be of poor quality and are many in the search space of typical optimization problems. In practice, an algorithm that is not able to continue the search process if it encounters a local optimum is of little use.

Algorithm 1 Iterative Improvement

Input: Initial solution s
while $\exists s' \in \mathcal{N}(s) : f(s') < f(s)$ **do**
  $s := s'$
end while
return s

Additional mechanisms are required to develop more efficient algorithms, and have led to the design of *metaheuristics*. Metaheuristics are general purpose methods that can be applied to different problems with limited modifications. Many of the most effective metaheuristics for combinatorial optimization are stochastic local search methods, and can be seen as extensions of the iterative improvement algorithm (Hoos and Stützle, 2005). In what follows, we review the most common metaheuristics that remain nowadays the main elements that are combined, modified, and improved in order to find even more effective algorithms. All the algorithms presented in pseudo-code are given for minimization problems.

2.4.1 Iterated Local Search

Iterated Local Search (ILS, see Lourenço *et al.* (2002)) can be seen as an extension of the iterative improvement algorithm, that uses a *perturbation* procedure to escape from local optima. A perturbation is a limited modification of the incumbent solution's components, regardless of the resulting objective value (it usually worsens the solution).

Algorithm 2 highlights the ILS metaheuristic in pseudo-code. Procedure localSearch improves the current solution by use of local search. A possible example would be to employ the iterative improvement algorithm.

Algorithm 2 Iterated Local Search

```
Input: Initial solution s
s := localSearch(s)
while ! termination criterion do
  s' := s
  s' := perturbation(s')
  s' := localSearch(s')
  if f(s') < f(s) or s' satisfies the acceptance criterion then
    s := s'
  end if
end while
return s
```

2.4.2 Simulated Annealing

The Simulated Annealing (SA, see Kirkpatrick *et al.* (1983)) metaheuristic has been historically one of the first metaheuristics, and remains nowadays one of the most used ones in practice. The name and inspiration come from the industrial process of cooling molded elements down at a controlled rate, so that it avoids the formation of cracks and the atomic organization becomes as resistant as possible by reaching a crystal structure with the lowest possible energy. Many metaheuristics are similarly inspired from physical, biological or natural processes.

Algorithm 3 presents the pseudo-code of the SA metaheuristic. A parameter `temperature` is used in the acceptance criterion that can accept worse solutions with a probability depending on a parameter called `temperature` and how much worse is the new

solution. The value of the parameter `temperature` is decreased according to a given schedule (often called temperature cooling schedule), which makes acceptance of worse solutions harder over time. If `temperature` reaches 0, only better solutions can be accepted and the algorithm acts like an iterative improvement algorithm.

Algorithm 3 Simulated Annealing

Input: Initial solution s
Input: Initial temperature T and a temperature schedule
while ! termination criterion **do**
  Choose s' randomly in $N(s)$
  if $f(s') < f(s)$ or s' satisfies probabilistic acceptance criterion (depending on T) **then**
    $s := s'$
  end if
  Update T according to its temperature schedule
end while
return s

2.4.3 Tabu Search

Tabu Search is a metaheuristic that escapes from local optima by using a memory of the recent search (Glover, 1989). This memory records some solution components (or moves applied to solution components) that are made "tabu", as to avoid solutions that have been previously visited.

Tabu search is presented in pseudo-code in Algorithm 4. In practical implementations, having an efficient move evaluation is crucial, and the details of how the tabu list is handled in memory typically depends on how solutions or moves are represented, and on data structures that make the check of the tabu status as efficient as possible.

2.4.4 Greedy Randomized Adaptive Search Procedures

The Greedy Randomized Adaptive Search Procedure (GRASP) is based on greedy construction methods (Feo and Resende, 1995). These methods are useful to produce quickly reasonably good solution, and are often used to provide the initial solutions for further,

Algorithm 4 Tabu Search

Input: Initial solution s
while ! termination criterion **do**
  Choose the best neighbor $s' \in \{N(s) : s' \text{ not tabu}\}$
  $s := s'$
  Update tabu list with s
end while
return s

more elaborated, components. The drawback of greedy construction methods when used as stand-alone is that they are deterministic, and typically they cannot produce many different solutions. Thus, they lack the capability of exploring the search space sufficiently to find even better solutions. GRASP was proposed to overcome this aspect: during the construction, the selection of the next component to be added to the partial solution is chosen randomly, using a probability distribution biased by the greedy criterion. Once the solution is complete, a local search step (for instance an iterative improvement procedure) can be used to attempt to improve it further.

An algorithmic outline of GRASP is presented in Algorithm 5.

Algorithm 5 Greedy Randomized Adaptive Search Procedure

$s := NULL$
while ! termination criterion **do**
  $s' := \{\}$
  while s' is not complete **do**
    Select c in remainingComponents(s) according to greedy randomized criterion
    Add c to s
  end while
  $s' :=$ localSearch(s')
  if $f(s') < f(s)$ **then**
    $s := s'$
  end if
end while
return s

2.4.5 Population-Based Metaheuristics

Several metaheuristics deal with a population of solutions instead of following a single path in the search space. Here, we present briefly the main families of population-based metaheuristics, as covering all variants would be beyond the scope of this thesis.

Evolutionary algorithms are probably the best known ones, and historically one of the first metaheuristics proposed for optimization. Evolutionary algorithms are inspired by the Darwinian evolutionary process of species, and, in particular, by the evolution of DNA sequences as a result of the degree of adaptation of individuals to their environment. The analogy translates individuals to solutions, and fitness of individuals to the solutions' objective value. Such algorithms rely on two key components that mimic two real aspects of the perpetuation of DNA: (i) mutation within one individual, and (ii) combination between individuals to create offspring. We refer the reader to Back *et al.* (1997) and Reeves (2010) for more details on evolutionary algorithms.

Particle Swarm Optimization algorithms (Eberhart and Kennedy, 1995) have been derived from what was originally an attempt to mimic the behaviors of bird flocks or fish schools. Such natural behaviors result from relatively simple rules followed by each individual, such as remaining within a given distance range of neighboring individuals or following a direction that is the average of their neighbors directions. The analogy is to consider that solutions in the search space are particles that follow such simple rules, with the addition of a bias towards particles that are of higher quality. The goal is to guide the search of the particle swarm towards better solutions while ensuring a minimum exploration capability.

Ant Colony Optimization (ACO) has been proposed as a general way to construct solutions using a probability distribution defined over their components, that evolves over time (Dorigo *et al.*, 1991, 1996; Gambardella and Dorigo, 1995). ACO is inspired by the behavior of ants, which by use of pheromones are able to actually find the shortest paths between two locations. The probability distribution that is used in the construction of new solutions is induced by artificial pheromones that are modified during the algorithm runtime, and possibly heuristic information. After each iteration the pheromones are updated by biasing them towards the components of the best solutions found during the search.

Many variants or combinations of these main types of population-based metaheuristics have been proposed, often using different names suggesting different natural or physical analogies, even-though the relevance of the algorithmic contributions themselves are sometimes questionable (Sörensen, 2013).

2.5 Multi-objective Optimization

Historically, optimization has been mainly done considering a single objective function value. In this case, two solutions can be compared leading to only two possible outputs: either one solution is better than the other, or the two solutions have the same objective function.

Multi-objective optimization has been later considered to tackle problems whose solutions are evaluated using different objectives. These objectives are often conflicting (if they are not, the problem can be considered with a single-objective approach), which leads to the following fundamental difference: two solutions can have different objectives' values, representing different trade-offs of the objectives, and none is better than the other.

When dealing with several, possibly conflicting objectives, two approaches can be considered, the *a priori* approach and the *a posteriori* approach. An approach that combines them, called *interactive*, is also possible though less frequent (Geoffrion *et al.*, 1972; Deb and Chaudhuri, 2007).

In the *a priori* approach, the decision maker provides preferences over the different objectives, which often allows tackling a problem using single-objective optimization methods. In this thesis, we focus on the second, the *a posteriori* approach, in which the preferences of the decision maker are unknown. In this case, the notion of optimal solution from single-objective optimization does not apply anymore, and instead one must rely on the notion of Pareto dominance. A solution s is better in the Pareto sense than another solution s' if s is better than s' for at least one objective and not worse for any of the remaining ones. If none of the two solutions is better than the other, they represent two different trade-offs of the objectives function that, without knowledge of the decision maker's preferences, are considered to be equally valuable. The goal of an

algorithm tackling a multi-objective problem in the Pareto sense is then to return all solutions representing different trade-offs among which the decision maker can choose the preferred one. In what follows, we define more formally the notion of Pareto dominance between solutions, and between sets of solutions.

2.5.1 Relations between Solutions

Let us consider a multi-objective combinatorial problem (MCOP) with p objectives, all to be minimized. We call $\vec{f}(s) \in \mathbb{R}^p$ the vector of the objective function values of solution s; $f_k(s)$ denotes the specific value of objective k, that is, the k-th component of the objective function vector $\vec{f}(s)$.

Definition 1 (Dominance). *A solution s_1 is said to dominate a solution s_2 ($s_1 \prec s_2$) if and only if $f_k(s_1) \leqslant f_k(s_2)\ \forall k = 1, \ldots, p$ and $\exists\, j \in \{1, \ldots, p\}$ such that $f_j(s_1) < f_j(s_2)$.*

Definition 2 (Weak dominance). *A solution s_1 is said to weakly dominate a solution s_2 ($s_1 \preceq s_2$) if and only if $f_k(s_1) \leqslant f_k(s_2)\ \forall k = 1, \ldots, p$.*

Definition 3 (Incomparable solutions). *Two solutions s_1 and s_2 are said to be incomparable ($s_1 \parallel s_2$) if and only if neither $s_1 \preceq s_2$ nor $s_2 \preceq s_1$, and $s_1 \neq s_2$.*

The fact that solutions can be incomparable is a fundamental difference to single-objective optimization, where a total ordering of the solutions exists. In the multi-objective context, the output of an algorithm is a set of solutions that are mutually incomparable, called a non-dominated set.

Definition 4 (Pareto global optimum solution). *Let $\mathbb{S}$ denote the set of all feasible solutions. A solution $s_1 \in \mathbb{S}$ is a Pareto global optimum if and only if $\nexists\, s_2 \in \mathbb{S}$ such that $s_2 \prec s_1$. Such solutions are also called efficient.*

Several Pareto global optimum solutions can have the same objective vector in the Pareto front. In practice, it is common to return only one solution for each objective vector. Such a set is also called strict Pareto global optimum set (Paquete *et al.*, 2007) or

strictly Pareto optimal set (Ehrgott, 2000).

Definition 5 (Pareto front). *Let $\mathbb{S}$ denote the set of all feasible solutions. A set S is a Pareto global optimum set if and only if it contains all the Pareto global optimum solutions of $\mathbb{S}$ and only these solutions. The set of objective vectors of the Pareto global optimum is called the Pareto front.*

2.5.2 Relations between Sets of Solutions

The dominance relations can be extended to compare sets of solutions. In the following, S and S' stand for two sets of solutions.

Definition 6 (Dominance on sets). *A set S is said to dominate a set S' ($S \prec S'$) if and only if $\forall\, s_i \in S'$, $\exists\, s_j \in S$, such that $s_j \prec s_i$.*

Definition 7 (Weak dominance on sets). *A set S is said to weakly dominate a set S' ($S \preceq S'$) if and only if $\forall\, s_i \in S'$, $\exists\, s_j \in S$, such that $s_j \preceq s_i$.*

Definition 8 (Better relation on sets). *A set S is said to be better than a set S' ($S \lhd S'$) if and only if $\forall\, s_i \in S'$, $\exists\, s_j \in S$, such that $s_j \preceq s_i$, and $S \neq S'$.*

Definition 9 (Incomparable sets). *S and S' are said to be incomparable ($S \parallel S'$) if and only if neither $S \lhd S'$ nor $S' \lhd S$, and $S \neq S'$.*

Evaluation of sets, and, thus, the evaluation of algorithms that search for such sets should first check for the dominance relations among sets. Especially when focusing on high-performance algorithms, however, it will be rarely the case that one set dominates the other; in such cases, it is common that the differences are not large enough and sets are incomparable. In the next section, we present additional measures that can be used to compare multi-objective algorithms and assess their performance.

2.6 Performance Assessment of Multi-objective Algorithms

If the dominance relations between sets are not sufficient to compare the output of multi-objective optimizers, other methods are required to assess the quality of the non-dominated sets of solutions and, thus, of the optimization algorithms themselves. These methods, called *indicators*, are of two kinds: unary indicators or binary ones. Unary indicators summarize the quality of a non-dominated set into a single scalar value, while binary indicators summarize the difference between two sets of non-dominated solutions. An important criterion for such indicators is that they should be consistent with Pareto dominance relations on sets. That is, if a set S' weakly dominates a set S according to Pareto dominance (extended to sets as seen in Def. 7), the quality of S' according to a given indicator should not be worse than that of S. For more information on the different existing indicators for the assessment of non-dominated sets of solutions, we refer to Zitzler *et al.* (2003).

In this thesis, we use the hypervolume indicator (Zitzler and Thiele, 1999; Fonseca *et al.*, 2006). The hypervolume indicator is among the only unary indicators (with the epsilon-indicator) that are fully compliant with the Pareto dominance relation.

2.6.1 Hypervolume Indicator

In the bi-objective space, the hypervolume measures the area of the objective space that is weakly dominated by the image of the solutions of a non-dominated set in the objective space. This area is bounded by a reference point that is worse in all objectives than all points in all non-dominated sets measured. The larger is this area, the better is a non-dominated set according to the hypervolume indicator. It is common to compute the hypervolume of non-dominated sets in a normalized objective space. Each objective value is normalized to an interval that is the same for all objectives, so that it does not favor objectives that have a higher absolute value or range. Figure 2.3 shows a graphical illustration of the normalization of a non-dominated set and of its hypervolume. In this example, the absolute values for the first objective range from 2 to 10 while the second objective values range from 1 to 5. Computing the hypervolume with these absolute

values would clearly favor the first objective over the second one. In this thesis we normalize objective values to the range $[1, 2]$, and we choose (arbitrarily) the point $[2.1, 2.1]$ as reference point.

The computational complexity for computing the hypervolume measure for a set of n solutions with two or three objectives is $O(n \cdot log(n))$, and increases exponentially with a larger number of objectives (Beume and Rudolph, 2006; Paquete *et al.*, 2006; Fonseca *et al.*, 2006; Beume *et al.*, 2009).

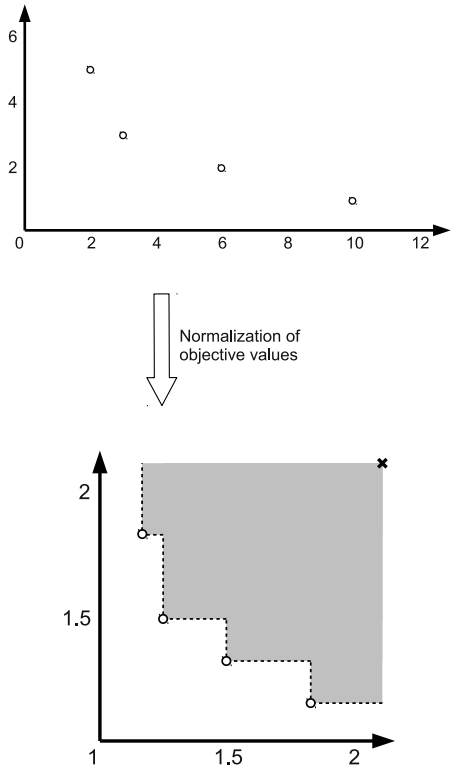

Figure 2.3: Graphical representation of a bi-objective non-dominated set. On top is the image of the set in the objective space. The range of the objective values differs significantly and computing the hypervolume without normalization would favor the first objective (on the x-axis). At the bottom, the same non-dominated set is shown after the objective values have been normalized to the interval $[1, 2]$. The area in gray is a representation of the hypervolume of this (normalized) non-dominated set, bounded by the reference point $[2.1, 2.1]$ shown as a cross.

The advantage of quality indicators, namely that they provide a single scalar value, is also their weakness: they infer the quality of the algorithm output as a whole, but are not meaningful as to the relative quality of this output in different regions of the objective space. In the next section, we introduce a method that allows us to provide such information when analyzing the output of an algorithm.

2.6.2 Graphical Quality Assessment: Empirical Attainment Function

The attainment function is a function over the objective space that defines the ability of an algorithm to generate a set that weakly dominates points of the objective space. If the algorithm is stochastic (and thus its output is stochastic, too), the outcome of the attainment function is not a binary value for each point, but rather it provides the probability of an arbitrary point in the objective space to be weakly dominated by a solution obtained by a single run of the algorithm (Grunert da Fonseca *et al.*, 2001).

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 attainment surfaces allows to assess the likely location of the output of an algorithm. In addition to this, differences between attainment functions 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 attainment functions 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.

The advantage of the attainment function is that it provides a graphical representation (with 2 objectives, or even 3 objectives) of the performance of a stochastic multi-objective algorithm in different regions of the objective space.

However, the attainment function of an algorithm is typically unknown a priori, and therefore it must be estimated empirically from several runs of the algorithm. The resulting estimation is called the empirical attainment function (EAF) (Grunert da Fonseca *et al.*, 2001).

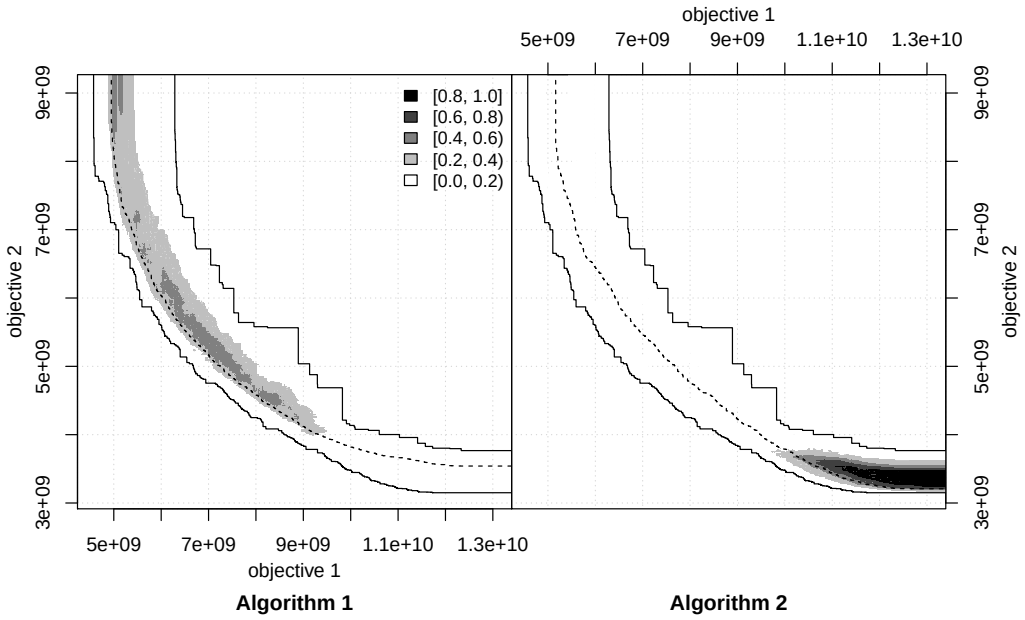

Figure 2.4: An example (from `http://iridia.ulb.ac.be/~manuel/eaftools`) of the difference between the empirical attainment functions of two algorithms. On the left is shown the area of the objective space where this difference is in favor of Algorithm 1, the darker the stronger. Conversely, on the right is shown the area where Algorithm 2 performs better. Dashed lines are the median attainment surfaces of algorithm 1 (left side) and algorithm 2 (right side), respectively. Black lines correspond to the overall best and overall worst attainment surfaces of both algorithms.

An example of the difference between the EAFs of two algorithms is shown in Fig. 2.4, 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 always attained 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 within these two lines. The dashed lines on each side are different, and they correspond to the median attainment surface of each algorithm, the median location of the output of each algorithm. We refer to Grunert da Fonseca *et al.* (2001) and López-Ibáñez *et al.* (2010) for a thorough explanation of these graphical tools.

2.7 Local Search Approaches to Multi-objective Optimization

In this section, we present the two paradigms for multi-objective optimization that we use and extend in this thesis. First we present the concept of scalarization that underlies the scalarization-based paradigm. We present the Two-phase Local Search approach and we give an overview of the literature of other multi-objective algorithms that also belong to this paradigm (Paquete and Stützle, 2007). Then we present a canonical representative of the second, dominance-based search paradigm, called Pareto Local Search, and give an overview of other algorithms that also belong to this second paradigm.

2.7.1 Scalarization of Multi-objective Problems

Algorithms for multi-objective optimization that belong to the scalarization-based paradigm are based on tackling a set of *scalarizations*. Scalarizations are single-objective problems that are defined from the original multi-objective problem to be tackled.

By using scalarizations, candidate solutions can be compared by scalar values, resulting in a total ordering of solutions. In other words, an aggregation transforms the multi-objective problem into a (*scalarized*) single-objective problem, often called simply *scalarization*. This drops the need to consider Pareto dominance relations, and any efficient single-objective algorithm can be used directly to tackle each scalarization. Tackling each of these scalarized problems provides a potential solution to the multi-objective one.

There are many ways of how to scalarize multi-objective problems. However, most often, few standard methods are used for their simplicity and desirable properties. We present here the most common methods in use in the context of heuristic algorithms; for other possibilities, we refer the interested reader to Ehrgott (2000) and Coello Coello *et al.* (2007). Note that all these methods are typically applied to normalized objective values, as described in Section 2.6.1.

- **Linear aggregation.** This method defines a linear, weighted aggregation of the objectives that is commonly used to define the preferences of the decision maker with an a priori approach (so only one single-objective problem must be tackled). Due to the linear weights, it is often called *weighted sum* aggregation. A linear aggregation can be used when the problem is tackled with an a posteriori approach to define scalarizations for tackling several single-objective problems defined from the multi-objective one. A weight vector is used to give the relative importance to each objective. Let us consider a solution s whose objective function vector is:

  $$\vec{f}(s) = (f_1(s), f_2(s), \ldots, f_p(s)).$$

  We assume that, without loss of generality, the components of $\vec{f}(s)$ are non-negative. The scalar value for this solution and a weight vector $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_p)$ is then:

  $$f_{\vec{\lambda}}(s) = \sum_{1 \leqslant i \leqslant p} \lambda_i \cdot f_i(s).$$

  Since the different components of the weight vector have an effect that is relative to the value of each other, there exist infinitely many different weight vectors that define the same scalarization. Therefore, it is common to use normalized weight vectors, whose components sum up to one. An optimal solution for the scalarized problem is known to be a *supported* Pareto global optimum, that is, its objective vector is located on the convex hull of the Pareto front.

- **Tchebycheff aggregation** The Tchebycheff method requires to define a *reference point*, r, that dominates any feasible solution. Some weights, additionally, can be assigned to each objective by a weight vector $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_p)$. The scalar value

for a solution s is then:

$$f_{\vec{\lambda}}(s) = \max\{\lambda_i \cdot |f_i(s) - f_i(r)|\}, \quad i = 1 \dots p.$$

Hence, the goal becomes to find a solution as close as possible to the reference point r, using the Tchebycheff distance as a measure of "closeness". Optimal solutions for the Tchebycheff aggregation problems are located on the Pareto front (Bowman and Joseph, 1976), but not necessarily on the convex hull of it.

- **Lexicographic ordering**. This method requires to define an order of the objectives in decreasing importance and, thus, it defines a total ordering of the solutions: If some solutions have the same value for objectives 1 to k, objective $k+1$ is used to break ties. The number of different orders of the objectives is limited, and therefore is also the number of solutions that can be obtained with this method. For instance, for bi-objective problems there are only two possible orderings of the objectives and thus only two different scalarized problems can be defined, preventing to return more than two solutions. Thus, when tackling a problem in the Pareto sense, a lexicographic ordering can be used to provide some initial solutions only, and must be used in combination with another technique.

Scalarized problems are often used when the multi-objective problem is tackled *a priori*, that is, the decision maker is able to define the relative importance of the objectives before the optimization. If the problem is considered using an a posteriori approach, multi-objective algorithms can make use of scalarizations, by using several weight vectors *during* the optimization process, with the aim of providing several non-dominated solutions.

In the next section, we present Two-phase Local Search, an algorithmic template for an a posteriori approach that is based on tackling a sequence of weighted-sum scalarizations.

2.7.2 A Scalarization-based Algorithm: Two-phase Local Search

Two-phase local search (TPLS), proposed in Paquete and Stützle (2003), is a canonical representative of a scalarization-based multi-objective algorithm and it is key to our work.

In its original form, TPLS considers a regular sequence of weights. Two main weight-setting strategies (or "directions") have been originally proposed to define the order in which these weights are selected. The simplest way to define a sequence of scalarizations is to use a regular sequence of weight vectors from one objective to the other. Let N_{scalar} be the number of weight vectors. Then, a regular sequence of weights from the first objective to the second can be defined as:

$$\lambda_1^i = \frac{N_{\text{scalar}} - i}{N_{\text{scalar}} - 1} \quad \text{and} \quad \lambda_2^i = 1 - \lambda_1^i, \quad \forall i = 1, \cdots, N_{\text{scalar}}.$$

This strategy is called *1to2*, and if the weights are defined in the opposite direction, the strategy is called *2to1*.

TPLS is a general algorithmic framework that, as the name suggests, is composed of two phases. Algorithm 6 presents the algorithmic outline of TPLS in pseudo-code. In the first phase, a single-objective algorithm generates a high-quality solution for one of the objectives considered individually (lines 1 and 2). The choice of the objective depends on the "direction", which can be either *1to2* (first to second objective) or *2to1* (second to first objective). This solution serves as the starting point of the second phase, where a sequence of scalarizations is tackled (lines 5 to 8) in the given direction. Each scalarization uses the solution found by tackling the previous scalarization as the solution to start from (line 6). Finally, solutions that are dominated are removed from the archive (line 9). TPLS will be successful if the underlying single-objective algorithms are high-performing, and if solutions that are close to each other in the solution space also have objective function vectors that are close to each other in the objective space.

However, this version of TPLS potentially introduces a bias towards the region of the objective space where the first scalarizations are performed, and against the region where the last ones are performed (Paquete and Stützle, 2003). To avoid this effect, a

Algorithm 6 The original version of Two-phase Local Search

1: **if** *1to2* **then** $s := \mathsf{SLS}_1()$
2: **else if** *2to1* **then** $s := \mathsf{SLS}_2()$
3: Add s to Archive
4: $s' := s$
5: **for** each weight λ_i **do**
6: $\quad s' := \mathsf{SLS}_\Sigma(s', \lambda_i)$
7: $\quad$ Add s' to Archive
8: **end for**
9: RemoveDominated(Archive)
10: **Output:** Archive

double weight setting strategy has been proposed: first a sequence of scalarizations is performed from one objective to the other and then a second sequence of scalarizations is performed in the opposite direction (Paquete and Stützle, 2003).

Different from the proposal in Paquete and Stützle (2003), in this thesis, first a solution is found for all single objective problems (whereas in Paquete and Stützle (2003) it is done only for one objective), because some high performing algorithms may be available for them, and we want to be consistent with our other improved TPLS variants (they will be developed in Chapter 3). However, we use only one of the solutions as a starting solution for further scalarizations.

The advantage of scalarization-based algorithms is that they can make use of any algorithm known to solve the scalarized problems effectively, and make use of its effectiveness in the multi-objective context. Typically, this allows scalarization-based algorithms to find high-quality solutions relatively quickly. The drawback is that solutions are found one at a time, and the total number of non-dominated solutions returned is at most the same as the number of scalarizations considered, which may be small compared to what is desirable.

In the next section, we review other relevant algorithms that are also based on scalarizing multi-objective problems.

2.7.3 Scalarization-Based Paradigm: Literature Review

Other multi-objective algorithms that follow the same rationale as TPLS, that is to tackle multi-objective problems by tackling a set of scalarized problems, have been proposed.

The changing horizon efficient set search (CHESS) has been proposed by Borges (2000). It uses the Tchebycheff distance, but not in the standard way presented previously in Section 2.7.1. Instead, the goal for each single-objective problem is to find a new solution that maximizes the Tchebycheff distance between this new solution and the closest one in the archive. As in TPLS, the idea is to obtain a set of well-spread solutions in the objective space.

Other methods have been proposed that are more specific than the general idea behind TPLS. They are still, however, general-purpose and can be applied to different problems. An example is Multi-objective Tabu Search (MOTS) proposed by Hansen (1997). MOTS is an extension of tabu search (Glover, 1989) (see Section 2.4.3) to multi-objective problems. MOTS keeps a set of non-dominated solutions and tries to improve each solution in a direction that moves its objective vector away from other non-dominated solutions. To do so, it updates the weights for a given solution based on all other non-dominated solutions (the closer solutions are, the higher is their mutual influence). The purpose of this behavior is to obtain a set of solutions that is as spread as possible in the objective space along the Pareto front. The optimization of solutions toward different directions is performed using tabu search principles, each solution dealing with its own tabu list.

There have been several adaptations of the Simulated Annealing (SA) principle to the multi-objective case. They usually use several runs of single-objective SA algorithms, and mainly differ by the acceptance rules of new solutions. The first SA for multi-objective problems, proposed by Serafini (1992), uses the following acceptance criterion. If the new solution dominates the current one, this new solution is accepted to replace it. Otherwise, the acceptance probability is computed based on a weighted sum of the objectives. Several runs are performed using different weight vectors, and some small random variations are applied to the weights each time a solution is considered. A similar method is MOSA, proposed by Ulungu *et al.* (1999). MOSA uses the same type of rule for the acceptance criterion and a similar set of predefined weight vec-

tors to define the single-objective problems. However, MOSA is not only returning one solution per weight vector: every time a solution is accepted as the new current one, it is potentially inserted in a set of non-dominated solutions. Each run of the single-objective SA maintains its own set of non-dominated solutions, and the sets are merged and filtered in a last step. Czyżżak and Jaszkiewicz (1998) proposed the Pareto Simulated Annealing (PSA). Several runs of a single-objective SA are performed *in parallel*, each run using a weight vector taking into account the closest solutions (in the objective space) obtained from other runs of the single-objective SA, with the goal to "escape" from them. Suppapitnarm *et al.* (2000) proposed another adaptation of the SA principle to the multi-objective case. The acceptance criterion is different from other SA adaptations. Their proposal uses a multiplicative function of the objectives, instead of a weighted sum, and a different *temperature* for each objective. The setting of the temperature does not follow a pre-scheduled decrease, but is automatically updated based on the variance of each objective among already accepted solutions. The algorithm is then restarted several times to provide several solutions.

Various population-based methods, such as evolutionary algorithms, have used scalarizations to direct the search towards the Pareto front. An early example is VEGA, proposed by Schaffer (1985). At each generation, the population is divided into sub-populations, and each of these sub-populations is assigned one of the objectives, which is used as fitness value without considering the other objectives. Thus, the algorithm converges towards a population made of clusters of solutions close to the best individual for each objective. The downside is that, by doing so, one can only produce a very limited number of solutions that optimize each objective but are not actual trade-offs between them. Ishibuchi and Murata (1998) proposed a memetic algorithm that only makes use of weighted sum scalarizations to evaluate solutions. In this algorithm, each time a new solution is generated, a random weight vector is used. First, the parents are selected from the population by taking the best solutions according to this random weight vector. Additionally, the new solution generated by the genetic operators (crossover and mutation) is potentially improved further by using a local search on the problem defined by the random weight vector. Jaszkiewicz proposed another memetic algorithm, also called MOGLS (Jaszkiewicz, 2002a,b). This method maintains separately a population of current solutions and an archive of the best solutions found so far, but it is similar to the MOGLS method of Ishibuchi and Murata in how new solu-

tions are generated. Ant Colony Optimization (ACO) algorithms frequently use some form of scalarized aggregation, for example, for combining pheromone (or heuristic) information specific to each objective (Angus and Woodward, 2009; García-Martínez *et al.*, 2007; López-Ibáñez and Stützle, 2012; Bezerra *et al.*, 2012).

In the next section, we focus on the second paradigm and we present a dominance-based algorithm that does not rely on scalarizations but uses Pareto dominance.

2.7.4 Pareto Dominance Approach for Multi-objective Optimization

A different paradigm for multi-objective optimization is to rely only on Pareto dominance. Unlike the algorithms that we have seen previously, those that are based on Pareto dominance cannot directly exploit algorithms from the single-objective field. However, as we will see in the next sections, they can be designed as conceptual extensions of single-objective algorithms.

We presented Pareto dominance in Section 2.5.1 as the criterion that is used to evaluate solutions when problems are tackled in the Pareto sense using an a posteriori approach. Algorithms that follow the dominance based paradigm make use of this relation not only to evaluate the final solutions to be returned, but *each time* solutions must be compared during the search process. One consequence is that, typically, this kind of algorithms does not record a single best solution seen so far, but rather a whole archive of solutions that are mutually non-dominated. In the next section, we will focus on Pareto Local Search, a general representative of this class of algorithms and a central algorithm in this thesis. Next, we will review the literature of similar algorithms.

2.7.5 A Dominance-based Algorithm: Pareto Local Search

PLS, proposed in Paquete *et al.* (2004), is a canonical representative of dominance-based multi-objective algorithms that improve solutions one at a time by the use of neighborhood search (Paquete *et al.*, 2007). While the original motivation for proposing PLS was to study the connectedness of solutions (Paquete *et al.*, 2004), PLS turned out to be also

an effective local search method for multi-objective problems. It should be noted that a very similar algorithm was proposed independently by Angel *et al.* (2004).

Pareto Local Search (PLS) is an iterative improvement method for solving MCOPs, that extends iterative improvement procedures (see Section 2.4) from the single-objective case to the multi-objective one by using a different acceptance criterion. While in the single-objective case an iterative improvement algorithm accepts a new solution if it is better than the current one, in the multi-objective case PLS accepts a new solution to enter an archive of solutions only if it is not dominated by any solution in the archive; additionally, PLS takes care that the archive contains only non-dominated solutions by filtering out dominated ones.

Algorithm 7 illustrates the PLS framework. Given an initial archive of non-dominated solutions, which are initially marked as unvisited, PLS iteratively applies the following steps. First, a solution s is randomly chosen among the unvisited ones in the archive (line 5). Then, the neighborhood of s is fully explored and all neighbors that are not weakly dominated by s or by any solution in the archive are added to the archive (lines 7 to 12). Solutions in the archive dominated by the newly added solutions are removed (line 14). Once the neighborhood of s has been fully explored, s is marked as explored (line 13). The algorithm stops when all solutions in the archive have been marked as explored.

An advantage of dominance-based algorithms is that they deal with an archive of solutions rather than a single solution. Therefore, they can return quickly numerous non-dominated solutions to the problem. However, this can also be a drawback since dealing with possibly many solutions can make the exploration of the search space slower in terms of progressing towards high-quality regions of the Pareto front.

In the next section, we present an overview of the literature of algorithms for multi-objective optimization that similarly to PLS use local search, and are purely based on Pareto dominance.

Algorithm 7 The original version of Pareto Local Search

Input: An initial set of non-dominated solutions A
explored(s) := FALSE $\quad \forall s \in A$
$A_0 := A$
while $A_0 \neq \emptyset$ **do**
  $s :=$ choose solution uniformly at random from A_0
  $A_0 := A_0 \backslash \{s\}$
  for each $s' \in \mathcal{N}(s)$ **do**
    if $\nexists s_1 \in A$ s.t. $s_1 \preceq s'$ **then**
      $explored(s') :=$ FALSE
      Add(A, s')
    end if
  end for
  $explored(s) :=$ TRUE
  RemoveDominated(A)
  $A_0 := \{s \in A \mid explored(s) =$ FALSE$\}$
end while
Output: A

2.7.6 Dominance-based Paradigm: Literature Review

We say that algorithms are dominance-based if they use some form of Pareto dominance for acceptance decisions on solutions. When comparing solutions using Pareto dominance, solutions may be mutually non-dominated; thus, there is only a partial order defined over solutions, which is a fundamental difference to the single-objective case. For this reason, dominance-based algorithms keep an archive of solutions instead of only a single solution as the best one found so far.

There are many algorithms for MCOPs that are purely dominance-based. We restrict our discussion to methods that are based on the iterative improvement of the set of non-dominated solutions by performing local search (or mutation) of solutions one at a time. We do not consider here population-based algorithms such as multi-objective evolutionary algorithms, for which we refer the interested reader to Deb (2001) and Coello Coello *et al.* (2007). It should be noted that these algorithms also often make direct or indirect use of Pareto dominance for directing the search, in particular, in the acceptance criterion or selection decisions on solutions, such as non-dominated sorting of solutions (Deb *et al.*, 2002). However, most evolutionary algorithms, contrary to PLS,

are not improving on single-solutions at a time but are population-based algorithms that produce several solutions at once.

Angel *et al.* (2004) proposed a method very similar to PLS. The difference lies in the selection step: in the method of Angel et al., contrary to PLS, the neighborhoods of all unexplored solutions are explored before updating the archive. This results in a stronger exploration capability than PLS because the archive is not updated immediately after the neighborhood of a single solution has been explored. Due to this, neighbors of solutions that otherwise would have become dominated can be examined in addition to those of non-dominated solutions.

Liefooghe *et al.* (2009, 2011) studied the performance of some variants of the PLS algorithm. The authors tested variants of the selection step that are obtained by restricting the overall exploration with a limit on the number of solutions to be selected, and variants of the neighborhood exploration itself, combining different ways of scanning the neighborhood and different acceptance criteria. The resulting variants are then compared experimentally using the same, predefined computation time limit for all variants. Variants that would finish before this computation time limit are restarted "from scratch" and a variant is judged by the final aggregated non-dominated set found across the multiple restarts. As a result, they highlight variants that are the most effective if the computation time is known a priori and the PLS algorithms are launched several times.

The fact that PLS stops upon finding a Pareto local optimum set can be a disadvantage if the algorithm finishes while there is still computation time available. A possibility is to keep the non-dominated solutions found in an external archive and to restart PLS from scratch (as in Liefooghe *et al.* (2011)). There were other extensions that aim at obtaining a possibly more efficient restart mechanism. Alsheddy and Tsang (2010) proposed an extension of PLS that continues the search when a Pareto local optimum set is found without restarting from different solutions. The idea is based on the *guided local search* (Voudouris and Tsang, 1999) strategy in the single-objective case: a penalty is applied to worsen the components of the objective vectors of solutions in the current archive, allowing the algorithm to escape from a Pareto local optimum set. Other strategies to continue the search focus on generating good solutions to restart the search. Solutions mutated from the ones in the Pareto local optimum set can be

used, resulting, in some sense in an extension of *iterated local search* (Lourenço *et al.*, 2010) for single-objective problems. A study of such strategies was done by Drugan and Thierens (2010). In that paper, the authors showed that the best results on the bi-objective quadratic assignment problem (which will be presented in detail later in the thesis) are attained when restarting PLS from new solutions on a "path" between two solutions in the Pareto local optimum set (this path is constructed in a manner similar to *path-relinking*, see Glover (1998)). Geiger (2011) proposed to apply a different neighborhood operator (w.r.t. the one used during the search that has lead to the local optimum set) when PLS converges, therefore allowing to find possibly new non-dominated solutions. This idea can be seen as an extension of *variable neighborhood search* (Hansen and Mladenovic, 2001).

Some evolutionary algorithms are similar to PLS. Pareto archived evolution strategy (PAES) has been proposed by Knowles and Corne (1999) as an algorithm whose simplicity aimed at making it a baseline for comparison to more complex evolutionary algorithms. In PAES, a solution is selected in the current archive of non-dominated solutions and a mutation operator is applied to obtain a new candidate solution. This new candidate is potentially inserted in the archive, which is then updated to keep only non-dominated solutions. The archive size is kept limited by using an archive bounding strategy. Contrary to PLS, this algorithm does not have a natural stopping criterion since solutions are never marked as explored. Laumanns *et al.* (2004) proposed Simple Evolutionary Multi-objective Optimizer (SEMO), which is similar to PAES but solutions are selected from an archive whose size is not limited. Differently from PAES, SEMO marks solutions as explored analogously to PLS. The authors also test variants of SEMO that differ in the selection step. These variants tend to balance the number of times solutions are selected for mutation, or they try to focus on the most recently found solutions.

2.7.7 Hybridization of the two Paradigms: Literature Review

We have previously presented one representative and gave an overview of the literature for each of the two search paradigms for multi-objective optimization. Each of these paradigms has its particular advantages and drawbacks. Dominance-based algorithms can return quickly a large number of non-dominated solutions; however, they progress rather slowly towards the Pareto front and they may require a long computation time

before reaching high-quality approximations to the Pareto front. Scalarization-based algorithms can exploit effective single-objective algorithms and they find quickly high-quality approximations to the Pareto front. However, they return only relatively few solutions and they may not be able to approximate well certain types of solutions. For example, heuristic algorithms based on weighted-sum scalarizations are not designed to identify non-supported solutions and, thus, they may leave "gaps" in the Pareto front approximation. Thus, combining the scalarization-based and the dominance-based search paradigms can be profitable, in order to exploit their respective advantages and to avoid as much as possible their respective disadvantages.

One can distinguish in the literature two different kinds of combinations. The first class of algorithms uses a *sequential* hybridization of the elements of each paradigm. The second class relies on an *iterative* hybridization, where elements of the two search paradigms are alternately applied numerous times.

In the following, we give a concise overview of some representative examples of sequential and iterative hybrids. For a more complete review, we refer the interested reader to Ehrgott and Gandibleux (2008).

Sequential Hybridization

Combining a scalarization-based and a dominance-based component by switching from one to the other is the most straightforward way of hybridizing the two search paradigms. This switching forms the basis of a sequential hybridization. A common usage of sequential hybrids is to first use an exact algorithm to solve scalarized problems to optimality and, thus, to provide some (or all) of the supported solutions. Then, in a second phase, a dominance-based component aims at finding some non-supported solutions. In the heuristic case, a scalarization-based component can provide a small set of high-quality solutions (not necessarily supported ones), and in a second step, a dominance-based component improves this set of solutions further. Here we describe some representative examples of such sequential hybrid algorithms.

In fact, a straightforward way to obtain a sequential hybrid is to rely on general algorithms such as TPLS and PLS. Such a simple form of an hybrid algorithm has been studied by Paquete and Stützle (2003), where they use a restricted form of PLS, the

component-wise step. Later, Lust and Teghem applied a sequential hybrid algorithm that runs the PLS phase to completion (Lust and Teghem, 2010b, 2012).

Hamacher and Ruhe (1994) combined the two search paradigms to tackle the bi-objective minimum spanning tree problem. A sequence of scalarizations is solved to optimality in the first phase; this is well feasible given that the minimum spanning tree problem is polynomially solvable. In a second phase, the neighborhood of all solutions obtained from the scalarizations is explored to search for additional, non-dominated solutions. Andersen *et al.* (1996) proposed a similar approach. They tested restrictions that consider only solutions that are neighbors of *two* different solutions in the set, and show that it may be useful for large scale problems since the number of solutions to consider is small.

Ulungu and Teghem (1995) proposed the *two-phases method*. This is a scheme for exactly solving MCOPs that works as follows. In a first phase, the whole set of supported solutions is determined using weighted sum scalarizations defined by the dichotomic scheme of Aneja and Nair (1979). In a second phase, this set of supported solutions is used to provide bounds to algorithms such as branch & bound, to find all non-dominated solutions. Despite being developed for exact solving, this approach has also inspired developments for heuristic solvers (Lust and Teghem, 2010b).

Gandibleux *et al.* (2003) proposed an algorithm for the bi-objective assignment problem that combined the two search paradigms as follows. First, an exact algorithm finds several supported solutions (a polynomial-time algorithm is known for the scalarized problems), and then the set of solutions obtained is further improved by seeding with this set a dominance based evolutionary algorithm, which is run for a few iterations.

Paquete and Stützle (2003) in a study on the TSP, further refine the set found by TPLS using a Pareto-dominance based component. The neighborhood of all solutions found by TPLS is explored to find additional non-dominated solutions. The empirical study shows that the quality of the results are significantly improved for the bi-objective TSP.

Parragh *et al.* (2009) designed an hybrid algorithm to solve the multi-objective dial-a-ride problem. A variable-neighborhood search algorithm is used to tackle weighted sum scalarizations defined by a regular sequence of weight vectors. In a second, dominance based phase, a path-relinking step is used to further improve the set of solutions.

Delorme *et al.* (2010) combined a greedy randomized adaptive search procedure (GRASP) (see Feo and Resende (1995)) with the strength Pareto evolutionary algorithm (SPEA) from Zitzler and Thiele (1999) to tackle the bi-objective set packing problem. GRASP is used to tackle a sequence of weighted sum scalarized problems, and then SPEA is used to improve further the set of solutions returned by GRASP.

Iterative Hybridization

A second possibility is to combine the scalarization and the dominance-based paradigms in an *iterative* way. In that case, typically a scalarization-based component is used for a specific step within a dominance-based algorithm. Such iterative algorithms are often implicit hybrids of the two search paradigms: algorithm designers seek the best possible performance and it leads them to include a scalarization based component within a dominance based algorithm (or vice-versa), without necessarily making the general concept behind this combination explicit.

Gandibleux *et al.* (1997) proposed an algorithm called MOTS (not to be confused with the MOTS algorithm by Hansen, mentioned previously in Section 2.7.3). This algorithm uses the tabu search principle to push solutions towards local *ideal* points. Once a solution becomes the new incumbent one in the tabu search process, its neighborhood is explored and Pareto dominance is used to add non-dominated solutions to the archive.

López-Ibáñez *et al.* (2006) tested the combination of a tabu search algorithm with a multi-objective ACO, and another combination with an evolutionary algorithm (SPEA2 Zitzler *et al.* (2002)). The ACO and the SPEA2 algorithms use the tabu search single-objective algorithm at each iteration to improve individual solutions by tackling scalarized problems defined from a regular sequence of weight vectors.

In the field of evolutionary algorithms, and in particular memetic ones (those that use a local search component additionally to the evolutionary process), numerous algorithms can be found that can also be seen as relying on the two search paradigms. As they are outside the scope of this thesis we refer the interested reader to Deb (2001) and Nerri and Cotta (2012).

In the next section, we present methods for the automatic setting of algorithms' parameters, on which we strongly rely in this thesis, and to which we also contributed ourselves.

2.8 Automatic Configuration of Algorithms

Heuristic algorithms typically have a number of parameters that strongly affect their performance on a particular problem, and even a particular instance. The setting of these parameters is a tedious task that has always been a significant part of the design process of effective heuristic algorithms. It requires expertise, several preliminary experiments based on trial-and-error and intuition from the algorithm designer. More recently, some tools have been developed that help to improve this time-consuming aspect, which has led to a research field of its own.

There are two different approaches making use of such tools. The first one is called *offline* configuration (Birattari, 2004). It consists of setting the parameters of an algorithm before it is deployed, using in a training process a set of instances that are representative of the instances to be tackled in the future. The offline approach direcly "replaces" the task of the manual parameter setting during the design process. The second one is called *online* configuration (Eiben *et al.*, 1999; Battiti *et al.*, 2008). It consists of adapting the parameters values during the search. It is, however, limited to a small subset of parameters as the algorithm needs to perform the search *and* at the same time "learn" effective values for the parameters.

In this thesis we make use of offline configuration. The advantages of such an approach are many.

- It avoids introducing a bias during the design phase, which is difficult to avoid as algorithm designers often, even unconsciously, favor what they a priori expect to work better (an algorithm, a specific component, etc.) by putting more effort into it.
- The exploration power is greatly increased and it allows evaluating many additional designs that a human would not have tested, because of a lack of time, or because these designs would have been judged worthless a priori.

- Because these methods are automatic, they transform a task that was highly time and effort consuming when performed by a human into a task that can be entirely performed on a computer. The algorithm designer, thus, can use all this free time to focus on high-level tasks. The whole design process then is closer to an intellectual task rather than a technical one, and the designer expertise is used where it is more valuable.

- It increases repeatability of experiments and results, since the parameter settings are not based on a subjective process.

The automatic configuration of algorithms is nowadays an active field of research, and several tools have been proposed in the last decade for this aim. In particular, in recent years a number of new algorithmic tools for the automatic *offline* configuration of parameterized algorithms have been developed. These include methods such as ParamILS (Hutter *et al.*, 2009b), gender-based genetic algorithms (Ansótegui *et al.*, 2009), CALIBRA (Adenso-Díaz and Laguna, 2006), SPO (Bartz-Beielstein, 2006), SPO$^+$ (Hutter *et al.*, 2009a), SMAC (Hutter *et al.*, 2011), extending over earlier research efforts (Birattari *et al.*, 2002; Coy *et al.*, 2001).

This approach has been demonstrated to be highly successful in recent years. The recent work on SATenstsein (KhudaBukhsh *et al.*, 2009) automatically configured a new state-of-the-art local search algorithm for the SAT problem. The automatic tuning of the CPLEX software package is also a noteworthy example, since CPLEX is a generic solver which has many parameters, the improvement brought by automatic configuration tools is of several orders of magnitude with respect to default settings (Hutter *et al.*, 2010). López-Ibáñez and Stützle (2010) have applied Iterated F-race to configure a multi-objective ant colony optimization (ACO) framework, leading to new multi-objective ACO algorithms that outperform previously proposed multi-objective ACO algorithms for the bi-objective traveling salesman problem. The tight integration of algorithm design and automatic configuration techniques has recently been coined "Programming by Optimization" (Hoos, 2012).

2.8.1 The `irace` Software for Offline Automatic Configuration

Birattari *et al.* (2002) (and later Birattari (2004, 2009)) proposed an automatic configuration approach, F-Race, based on *racing* (Maron and Moore, 1997) and Friedman's non-parametric two-way analysis of variance by ranks. This proposal was later improved by refining iteratively the sampling distribution and repeated applications of F-Race. The resulting automatic configuration approach was called Iterated F-race (I/F-Race) (Balaprakash *et al.*, 2007; Birattari *et al.*, 2010). Although a formal description of the I/F-Race procedure is given in the original publications, no implementation of it has been made publicly available. The `irace` package implements a general *iterated racing* procedure, which includes I/F-Race as a special case. It also implements several extensions already described by Birattari (2004, 2009), such as the use of the paired t-test instead of Friedman's test. We have also added several original contributions to improve the effectiveness of the tuning procedure.

One of the advantages of this tool is that it handles several parameter types: continuous, integer, categorical (those with discrete values and no implicit order), and ordinal (those with discrete values and an implicit order). Continuous and integer parameters take values within a range specified by the user. Categorical parameters can take any value among a set of possible ones explicitly given by the user. An ordinal parameter is a discrete parameter with a pre-defined strict order of its possible values (for instance "low", "medium" and "high"). This tool also allows to parallelize the configuration process, reducing considerably the amount of time required to obtain good results if an adequate hardware platform is available.

The general scheme of `irace` is shown in Fig. 2.5. The minimal requirement for the user is to provide (i) a set of training instances and (ii) a definition of the parameters by giving their types and possible values.

The irace software is also described in details in the appendix. In the next section, we present one of the key aspects of our work, namely the concept of anytime behavior of algorithms.

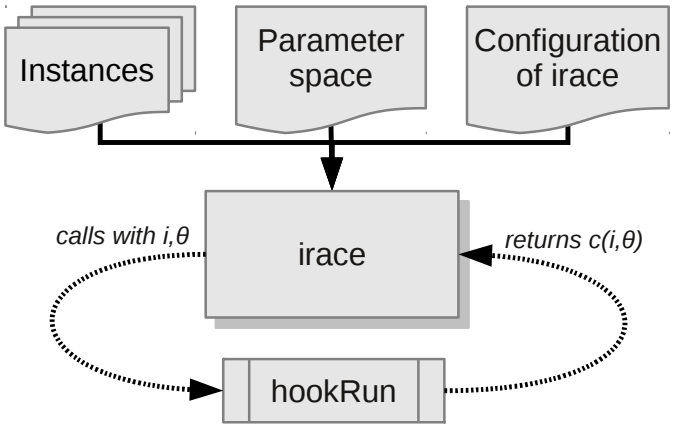

Figure 2.5: Scheme of `irace` flow of information.

2.9 Anytime Behavior of Algorithms

Optimization algorithms are often designed to produce the best possible outcome after a *specific* computation time. This setting is reasonable if an algorithm is to be deployed in a scenario where the computation environment and the computation time available are known a priori and do not vary each time a new instance is tackled. Unfortunately, in a significant part of the literature, optimization algorithms are designed in this way, *independently* of any real-world situations. This may be explained by the necessity to compare algorithms. Comparing algorithms in this way is relatively simple, because in this case one must deal with single values (the quality of the solutions found after the given computation time), and statistical tests can be used to easily assess the significance of the differences. However, such evaluation of algorithms may not be relevant in practice: (i) the computation environment may be different from the one used during the design phase, (ii) the termination criterion used when the algorithm is deployed may be different from the one used during the design phase, (iii) each time an instance must be tackled the termination criterion may change, (iv) the termination criterion may simply be unknown at the design time of the algorithm, or (v) results are expected while the algorithm is still running.

A different perspective is to evaluate algorithms independently of any given termination criterion, designing what is called *anytime algorithms*. Anytime algorithms aim at delivering as high quality results as possible independently of the computation

time (Zilberstein, 1996), continuously improving the quality of the results as computation time increases.

It must be noted that for some algorithms the quality obtained over time is reported to illustrate algorithm behavior. However, this type of evaluation must not be confused with the goal of obtaining an as good as possible anytime behavior. In fact, the observed behavior may be heavily dependent on some predefined computation time t, even before time t, because the algorithm design and its parameters have been chosen with the specific time t in mind. On the contrary, an algorithm whose aim is to obtain as good as possible anytime behavior is designed with this goal in mind from the very beginning. Such an algorithm will be said to be strictly better than a competitor if it produces better results at *any* time. The fact that this property is much more difficult to fulfill than using a single computation time may also explain why it is less encountered in the literature.

Although there has been some work on anytime algorithms for single-objective optimization problems (see, for instance, Zilberstein (1996); Loudni and Boizumault (2008)), there is very little research on anytime multi-objective optimization algorithms. This may be explained by the two characteristics that increase the difficulty of this case: (i) sets of non-dominated solutions can be incomparable, and (ii) the quality obtained by anytime algorithms over time may also be incomparable. A possibility is to use the hypervolume indicator (see Section 2.6.1) or any other indicator to compare non-dominated sets of solutions and, thus, to address the first difficulty. Fig. 2.6 illustrates this method; it shows the development of the obtained hypervolume over time for two algorithms. The plain curve shows an algorithm with good anytime behavior: the algorithm produces a quick increase of the quality at the beginning and then continuously improves until the end of the execution. The dotted curve shows an algorithm with a poor anytime behavior; its quality improves slowly and it is worse than that of the other algorithm at any time. In fact, each of these curves can be seen as a set of non-dominated solutions, where one objective is to maximize the hypervolume and the other one is to minimize the time, in which case the plain curve could be said to dominate (in the Pareto sense) the dotted curve.

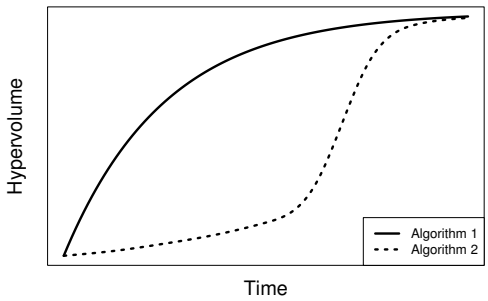

Figure 2.6: An example of the hypervolume attained by two algorithms over time. Algorithm 1 shows a good anytime behavior, while Algorithm 2 shows poor anytime behavior.

2.10 Summary

In this chapter, we presented the definitions, concepts, and previous works that underlie our work. We first presented the basic ideas: problems, instances, solutions, the concept of local search algorithms and heuristic algorithms, and we gave an overview of the most prominent metaheuristics. We then explained the fundamentals of multi-objective optimization, at the center of which is the Pareto dominance between solutions, and the Pareto dominance relations between sets of solutions. We explained the two methods that we use throughout this thesis to evaluate multi-objective algorithms, which are the hypervolume and the empirical attainment function. We reviewed the literature and summarized the work that are relevant to ours, using a conceptual dichotomy between two different search paradigms and their combination. We introduced methods for the automatic configuration of algorithms, one of which is a contribution of this thesis in itself, and a tool on which we rely heavily to design algorithms. Finally, we presented one key aspect of our work, the anytime behavior of algorithms, and why it matters.

Chapter 3

Anytime Two-phase Local Search

In this chapter, we focus on the scalarization-based paradigm that characterizes multi-objective optimization algorithms that rely on tackling scalarized single-objective problems. Our focus is on a canonical representative of such algorithms: Two-phase Local Search (TPLS). Our goal in this chapter, is to propose new algorithms that are based on the original TPLS and that, in particular, improve TPLS's anytime behavior.

This chapter is organized as follows. We first present formally the flowshop scheduling problem in Section 3.1. In Section 3.2 we look at TPLS from a more general point of view. Chapter 2 presented TPLS as proposed originally, here we will present a pseudo-code outline that highlights its different algorithmic components. Section 3.3 presents our first proposal to improve the anytime behavior of TPLS, and an empirical evaluation of the results that it achieves. Our second proposal is explained and evaluated in Section 3.4. Both variants and the original TPLS algorithm are compared experimentally and the results are statistically analyzed in Section 3.5. Section 3.6 presents a further algorithmic enhancement and its evaluation. We perform in Section 3.7 a graphical evaluation of the different algorithms by use of empirical attainment functions. Finally we summarize the contributions of this chapter in Section 3.8.

3.1 The Permutation Flow-Shop Scheduling Problem

The flow-shop scheduling problem is one of the most important scheduling problems and has been thoroughly studied since it was proposed by Johnson (1954). In the flow-shop scheduling problem, 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, and, hence, there are $n!$ possible schedules, which correspond to permutations of the jobs. In this case, 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; the completion times of all jobs on all machines are given by the recursive formula:

$$
\begin{aligned}
&C_{\pi_0 j} = 0 \\
&C_{\pi_i 0} = 0 \\
&C_{\pi_i j} = \max\{C_{\pi_{i-1} j}, C_{\pi_i j-1}\} + p_{ij}, \qquad i = 1, \ldots, n, \quad j = 1, \ldots, m.
\end{aligned}
$$

For simplicity, in the remainder of this thesis, C_i denotes the completion time of a job J_i on the last machine M_m. The *makespan* is the completion time of the last job in the permutation, that is, $C_{\max} = C_{\pi_n}$. In the following, we refer to the PFSP with makespan minimization as *PFSP-*$C_{\max}$. Despite the fact that the *PFSP-*$C_{\max}$ can be solved in polynomial time for two machines, for $m \geqslant 3$ the problem is $\mathcal{NP}$-hard in the strong sense (Garey *et al.*, 1976).

The other objectives that we study are the minimization of the *sum of flowtimes* and the minimization of the *weighted tardiness*. Because all jobs are assumed to be available at time 0, the sum of flowtimes is simply given by $\sum_{i=1}^{n} C_i$. This objective is also known as sum of completion times or total completion time. We refer to the PFSP with sum

of flowtimes minimization as *PFSP-SFT*. The *PFSP-SFT* is strongly $\mathcal{NP}$-hard even for only two machines (Garey *et al.*, 1976). When the problem includes the total (weighted) tardiness objective, each job has an associated due date d_i, the tardiness is defined as $T_i = \max\{C_i - d_i, 0\}$ and the total weighted tardiness is given by $\sum_{i=1}^{n} w_i \cdot T_i$, where w_i is a priority assigned to job J_i. If all weights w_i are set to the same value, we say that the objective is the *total tardiness*, and use *PFSP-TT* to denote the PSFP minimising this objective. Otherwise, we use *PFSP-WT* to denote the PFSP with weighted tardiness minimization. The *PFSP-TT* and the *PFSP-WT* are strongly $\mathcal{NP}$-hard even for a single machine (Du and Leung, 1990). Most of the studies involving tardiness focus only on the non-weighted variant. The review of Minella et al. (Minella *et al.*, 2008), for instance, does not consider the *PFSP-WT*. However, in this work, we consider both, the *PFSP-TT* and the *PFSP-WT* problem.

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 minimization of the makespan and the sum of flowtimes. *PFSP-*($C_{\max}$,TT) and *PFSP-*($C_{\max}$, *WT*) denote the minimization of the makespan and, respectively, total and weighted tardiness. *PFSP-*(*SFT*, *TT*) and *PFSP-*(*SFT*, *WT*) denote the minimization of the sum of flowtimes and, respectively, total and weighted tardiness. A number of algorithms have been proposed to tackle each of these bi-objective problems separately. Rarely, however, the same paper has addressed more than one combination. Minella *et al.* (2008) give a comprehensive overview of the literature on the three most commonly tackled problems (considering only the total tardiness), and presents the results of a sound and extensive experimental analysis of 23 algorithms. These algorithms are either PFSP-specific or more general and adapted by Minella et al. to tackle this problem. Their review identifies a multi-objective simulated annealing from Varadharajan and Rajendran (2005) as the best performing algorithm for all combinations of objectives. They also point out a multi-objective genetic local search proposed by Arroyo and Armentano (2005) as the highest performing alternative. Given their conclusions, these two multi-objective algorithms represented the state-of-the-art for the bi-objective PFSPs (bPFSPs) tackled in this chapter.

3.2 A Generalized View of Two-phase Local Search

In order to propose better algorithms that are based on TPLS, it is first necessary to have a clear view of the different components that underlie TPLS and that must be considered to be enhanced. This decomposition will be useful in the rest of this chapter as it covers in a common outline both the original TPLS algorithms and subsequent developments that we will propose.

We give in Algorithm 8 the pseudo-code of TPLS for bi-objective problems as seen from a general perspective. By presenting the algorithm in this way, one can decompose it into different pieces, which will be useful in the rest of this chapter as such an outline covers both the original TPLS and subsequent developments that we will propose, depending on how the different components are implemented.

First, high-quality solutions are generated for each objective (lines 1 and 2) using dedicated single-objective algorithms SLS_1 and SLS_2, and added to the archive (lines 3 and 4). Then, a sequence of scalarizations is solved (lines 5 to 10), based on strategies to generate a weight vector (procedure ChooseWeight on line 6) and to define how the previous solutions can be used as seed for further scalarizations (procedure ChooseSeed on line 7). Solutions are generated using a single-objective algorithm that tackles scalarized problems, SLS_Σ. The archive is updated with the solutions obtained for each scalarization (line 9). Note that the archive could include more information than just the solutions themselves, e.g., it could also include the fact that solutions have already been used as seeds, from which scalarization they have been obtained, etc.

The simplest way to define a sequence of scalarizations for TPLS is to use a regular sequence of weight vectors from the first objective to the second or from the second objective to the first one (see also Section 2.7.2, page 32). We call these alternatives *1to2* or *2to1*, depending on the direction followed. For example, the successive scalarizations in *1to2* are defined by the weights for objective 1 given by $\lambda_1^i = \frac{N_{\text{scalar}}-i}{N_{\text{scalar}}-1}, \forall i = 1, \cdots, N_{\text{scalar}}$, where N_{scalar} is the number of scalarizations. For simplicity, we henceforth refer to weight vectors by their first component only, since the second component can be derived from the first one in the bi-objective case by computing one minus the weight of the first objective. In *2to1* the sequence is reversed. Two drawbacks of this simple strategy are (i) that the direction chosen can give an advantage to the starting objective, that is, the

Algorithm 8 General Framework for Two-phase Local Search

1: $s_1 := \mathsf{SLS}_1()$
2: $s_2 := \mathsf{SLS}_2()$
3: $Archive := \mathsf{Update}(Archive, s_1)$
4: $Archive := \mathsf{Update}(Archive, s_2)$
5: **repeat**
6: $\quad \lambda := \mathsf{ChooseWeight}(Archive)$
7: $\quad s' := \mathsf{ChooseSeed}(\lambda, Archive)$
8: $\quad s' := \mathsf{SLS}_\Sigma(s', \lambda)$
9: $\quad Archive := \mathsf{Update}(Archive, s')$
10: **until termination criterion**
11: $\mathsf{RemoveDominated}(Archive)$
12: **Output:** $Archive$

Pareto front approximation will be better on the starting side; and (ii) that 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.

Double strategy. We denote as Double TPLS (D-TPLS, see Paquete and Stützle (2003)) 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 opposite direction. This approach tries to avoid the bias of a single starting objective. To introduce more variability, in our D-TPLS implementation, the weights (for the first objective) used in the second TPLS pass are located in-between the weights used for the first TPLS pass. D-TPLS still requires to define the number of weights, and, hence, the computation time, in advance.

3.3 Regular Anytime TPLS

The original strategy of TPLS, which is based on defining successive weight vectors with minimal weight changes, generates very good approximations to the areas of the Pareto front "covered" by the weight vectors (Paquete and Stützle, 2003, 2009b). However, if

TPLS is stopped prematurely, it leaves areas of the Pareto front unexplored. In this section, we present a first proposal to improve the anytime behavior of TPLS.

3.3.1 Regular Anytime Strategy

We propose a TPLS-like algorithm, called *regular anytime* TPLS (RA-TPLS), in which the weight for each new scalarization is defined in the middle of the interval of two previous consecutive weights. This strategy provides a finer approximation to the Pareto front as the number of scalarizations increases, ensuring a fair distribution of the computational effort along the Pareto front and gradually intensifying the search. The set of weights is defined as a sequence of progressively finer "levels" of 2^{k-1} scalarizations (at level k) with maximally dispersed weights Λ_k in the following manner: $\Lambda_1 = \{0.5\}$, $\Lambda_2 = \{0.25, 0.75\}$, $\Lambda_3 = \{0.125, 0.375, 0.625, 0.875\}$, and so on. Successive levels intensify the exploration of the objective space, filling the gaps in the Pareto front. The two initial solutions minimizing each objective could be seen as level 0: $\Lambda_0 = \{0, 1\}$. Once RA-TPLS completes one level, the computational effort has been equally distributed in all directions. However, if the search stops before exploring all scalarizations at a certain level, the search would explore some areas of the Pareto front more thoroughly than others. In order to minimize this effect, RA-TPLS considers the weights within one level in a random order.

In order to be an alternative to TPLS, RA-TPLS starts each new scalarization from a solution obtained from a previous scalarization. In particular, the initial solution of the new scalarization (using a new weight) is one of the two solutions that were obtained using the two weight vectors closest to the new weight. The algorithm computes the weighted sum scalar values of these two solutions according to the new weight, and selects the one with the better value as the initial solution of the new scalarization.

The implementation of RA-TPLS requires three main data structures: L_i is the set of pairs of weights used in previous scalarizations, where i determines the level of the search; *Sd* is a set of potential initial solutions, each solution being associated with the corresponding weight that was used to generate it; *Archive* is the archive of non-dominated solutions.

Algorithm 9 describes RA-TPLS in detail. In the initialization phase, an 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 *Sd*. In the next phase, the `while` loop (lines 5 to 10) is iterated until a stopping criterion is met. At each iteration, a pair of consecutive weights $(\lambda_{\text{sup}}, \lambda_{\text{inf}})$ is subtracted randomly from L_i and used to calculate the new weight $\lambda = (\lambda_{\text{sup}} + \lambda_{\text{inf}})/2$. Then, procedure ChooseSeed uses this weight λ to choose a solution from the set of initial solutions *Sd*. To do so, first ChooseSeed finds the two non-dominated solutions that were obtained from scalarizations using the weights closest to λ:

$$\begin{aligned} s_{\text{inf}} &= \left\{ s_i \mid \max_{(s_i,\lambda_i)\in S_d} \{\lambda_i : \lambda_i < \lambda\} \right\} \\ s_{\text{sup}} &= \left\{ s_i \mid \min_{(s_i,\lambda_i)\in S_d} \{\lambda_i : \lambda_i > \lambda\} \right\} \end{aligned} \tag{3.1}$$

Next, ChooseSeed calculates the scalar value of s_{sup} and s_{inf} according to the new weight λ, and returns the solution with the smaller scalar value. This solution is the initial solution for SLS_Σ, the SLS algorithm used to tackle the scalarizations. This algorithm produces a new solution s', which is added to both the global archive and, together with its corresponding weight, to the set of initial solutions *Sd*, from which any dominated solutions are removed. Finally, the set of weights for the next level L_{i+1} is extended with the new pairs $(\lambda_{\text{sup}}, \lambda)$ and $(\lambda, \lambda_{\text{inf}})$. This completes one iteration of the loop. If the current set of weights L_i is empty, a level of the search is complete, and the algorithm starts using pairs of weights from the next level L_{i+1}. In principle, this procedure may continue indefinitely, although larger number of scalarizations will lead to diminishing improvements in the approximation to the Pareto front.

3.3.2 Experimental Analysis

In this chapter, we perform a comprehensive study on a large set of instances for bPFSPs and the bi-objective TSP (bTSP). The different characteristics of these two problems will allow us to show that the shape of the Pareto front plays a fundamental role in the

Algorithm 9 RA-TPLS

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

performance of the RA-TPLS strategy. Sb All algorithms evaluated in this chapter were implemented in `C++`, compiled with gcc 4.4, 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.

Case Study: Bi-objective Traveling Salesman Problem

The single-objective TSP, presented in Section 2.2, can be easily extended to several objectives. Single-objective instances that have the same size can be considered for each objective. That is, for p objectives, p distance matrices are given. These distance matrices can have some intrinsic characteristics as in the single-objective case (Euclidean, symmetric...) but also can have mutual characteristics. For instance they can be correlated, to vary the correlation between the objectives.

Here we focus on the bi-objective TSP (bTSP). The focus on this thesis is on situations where the preferences of the decision maker are not known a priori. Hence, the goal is

to find a set of feasible solutions that "minimizes" the bTSP in the sense of Pareto optimality. The bTSP is a frequent benchmark for testing algorithms and comparing their performance (Ehrgott and Gandibleux, 2004; Paquete and Stützle, 2009b). Moreover, TPLS is a main component of the current state-of-the-art algorithm (Lust and Teghem, 2010b) for the bTSP.

Isometric and Anisometric Bi-objective Traveling Salesman Problem Instances

We created 10 Euclidean bTSP instances by generating for each instance two sets of 1000 points with integer coordinates uniformly distributed in a square of side-length 10^5. We call these instances *isometric* because both distance matrices have similar range.

In addition, we generated other bTSP instances, where the first distance matrix (corresponding to the first objective) is Euclidean whereas the second matrix (corresponding to the second objective) is randomly generated with distance values in the range $[1, \max_{dist}]$, with $\max_{dist} \in \{5, 10, 25, 100\}$. Given the different range of both distance matrices, we call these instances *anisometric*. We generated 10 instances of $1\,000$ nodes for each value of $\max_{dist}$, that is, 40 *anisometric* bTSP instances in total.

Experimental Setup for the Bi-objective Traveling Salesman

The underlying single-objective algorithm for the TSP is an iterated local search (ILS) algorithm based on a first-improvement 3-opt algorithm (Hoos and Stützle, 2005).[1] In order to speed up the algorithm, we compute a new distance matrix for each scalarization and we recompute the candidate sets used by the speed-up techniques of this ILS algorithm. For each scalarization, ILS runs for $1\,000$ ILS iterations (equal to the number of nodes in the instance). With our implementation and computing environment, $1\,000$ iterations require between 0.5 and 1 CPU seconds depending on the instances. Each of the two initial solutions is generated by running ILS for $2\,000$ iterations. Finally, each run of the multi-objective algorithms performs 30 scalarizations after generating the two initial solutions. The normalization of the objectives, necessary when solving a scalarization or calculating a weighted sum of the objectives, is performed by normalizing the two distance matrices to the same range $[1, 2]$.

[1] This algorithm is available online at `http://www.sls-book.net/`

We measure the quality of the results by means of the hypervolume unary measure (see Chapter 2, Section 2.6.1). We use $(2.1, 2.1)$ as the reference point for computing the hypervolume.

Experimental Evaluation of Regular Anytime Two-Phase Local Search on Bi-objective Traveling Salesman Instances

We first study how the different TPLS strategies satisfy the *anytime* property by examining the quality of the Pareto front as the number of scalarizations increases. For each TPLS strategy, we plot the hypervolume value after each scalarization averaged across 15 independent runs. Figure 3.1 shows four exemplary plots comparing RA-TPLS, *1to2* and D-TPLS on two isometric bTSP instances, and two anisometric bTSP instances. We do not show the strategy *2to1* for isometric instances because it performs almost identical to *1to2* w.r.t. the hypervolume; however, for anisometric instances we include *2to1* due to its rather different behavior when compared to *1to2*. These plots are representative of the general results on other instances; the complete results are given as supplementary material (Dubois-Lacoste *et al.*, 2010a).

For isometric bTSP instances, according to the top plots in Fig. 3.1, the three strategies reach similar final quality. However, there are strong differences in the development of the hypervolume during the execution of the algorithms. The hypervolume of the Pareto front approximations generated by RA-TPLS shows a quick initial increase, and for few scalarizations a much higher value than D-TPLS and *1to2*. Hence, if the algorithms are interrupted before completing the pre-defined number of scalarizations, RA-TPLS would clearly produce the best results.

For anisometric bTSP instances, where the first objective corresponds to a Euclidean distance matrix and the second objective corresponds to a randomly generated matrix with distance values in the range $[1, \max_{dist}]$, the bottom plots in Fig. 3.1 show a clear difference between strategies *1to2* and *2to1*. Moreover, the smaller the value of $\max_{dist}$, the larger is the difference. The value of $\max_{dist}$ also affects the anytime behavior and final performance of RA-TPLS. For small $\max_{dist}$, both *1to2* and D-TPLS seem to outperform RA-TPLS at various times. This can be explained as follows. Smaller values of $\max_{dist}$ result in a very large number of optimal solutions for the second objective. In such instances, the first scalarization of *2to1*, which uses a nonzero weight for the first

objective and a large weight for the second, generates a solution with a value of the second objective being still optimal and a good value of the first objective. This translates into a huge initial improvement of the hypervolume. The underlying reason is that the initial solution minimizing the second objective is weakly dominated by the solution returned for solving the first scalarization. Several subsequent scalarizations of *2to1* improve only slightly the value of the first objective, while keeping the optimal value of the second objective; therefore, the hypervolume improves very slowly. Only when the weight for the first objective grows large enough, a solution with a non-optimal value of the second objective is chosen, and the hypervolume starts improving in larger steps. On the other hand, when starting from the first objective in *1to2*, every scalarization finds non-dominated solutions closer to each other, and the hypervolume grows initially slower than what is observed for the first huge step in *2to1*. However, as soon as the weight of the second objective is large enough that only optimal values of the second objective solutions are accepted, the hypervolume quickly reaches its maximum. Finally, D-TPLS obtains better results because it progresses faster towards the second objective. All these behaviors show that equally distributing the computational effort in all directions does not pay off in these instances. It leads to a waste of scalarizations when being close to the optimum of the second objective, and very slow progress when being close to the optimum of the first objective. This effect will be even stronger in the case of the bPFSP.

Experimental Setup for the Bi-objective Permutation Flowshop Scheduling Problem

We use effective iterated greedy (IG) algorithms (Ruiz and Stützle, 2007) to tackle the scalarized problems withing each of the TPLS variants. More details on these IG algorithms will be given in Chapter 4 (in particular in Sec. 4.1, p. 92). Practically, each TPLS algorithm generates two initial solutions for each objective by running $1\,000$ iterations of the corresponding IG algorithm. Then, it performs 30 scalarizations, each scalarization running 500 iterations of the IG algorithm corresponding to the combination of objectives.

We generate 10 benchmark instances with $n = 50$ and $m = 20$ (`50x20`), and 10 instances with $n = 100$ and $m = 20$ (`100x20`), following the procedure described by Mi-

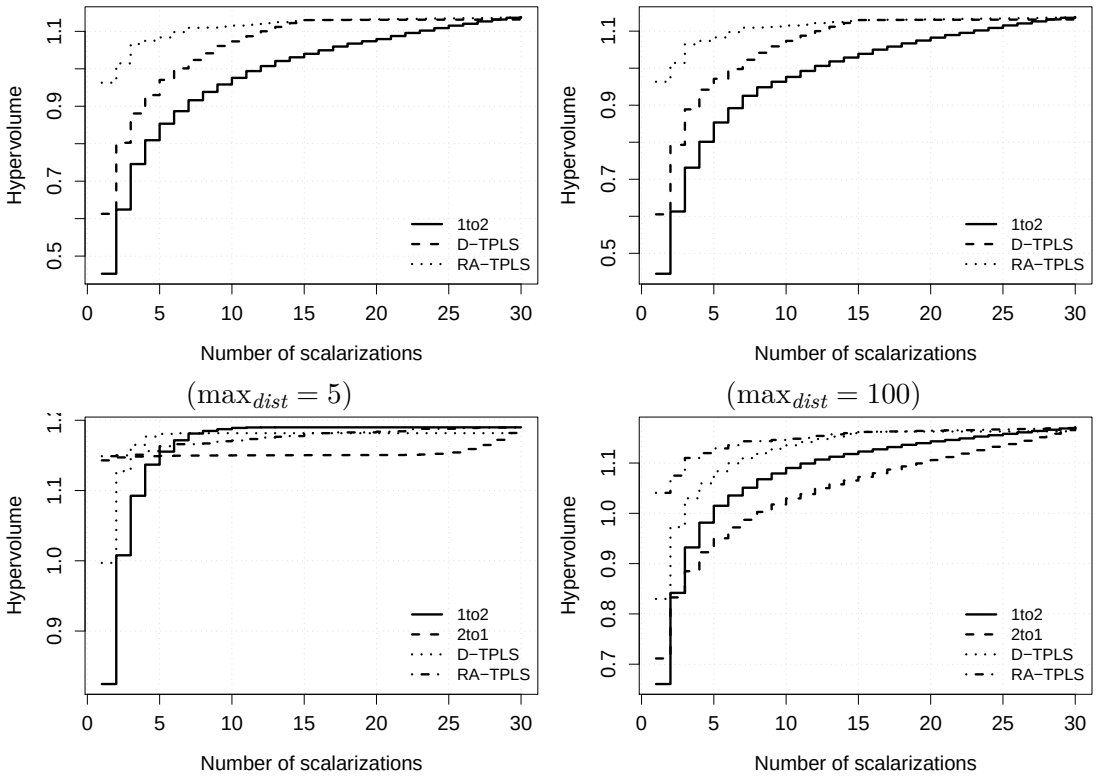

Figure 3.1: Development of the hypervolume over the number of scalarizations for *1to2*, D-TPLS and RA-TPLS for two isometric (top plots) and two anisometric (bottom plots) bTSP instances. For anisometric instances we also plot the results of *2to1*, since they differ strongly from *1to2*. For isometric instances, there is almost no difference between *1to2* and *2to1*. Anisometric instances with intermediate values of $\max_{dist}$ (equal to 10 or 25) show a compromise trend between the two extreme values 5 and 100 (see supplementary material in Dubois-Lacoste *et al.* (2010a)).

nella *et al.* (2008). These instances are available as supplementary material in Dubois-Lacoste *et al.* (2010a). Given the large discrepancies in the range of the various objectives, all objectives are dynamically normalized using the maximum and minimum values found during each run for each objective. We compute and plot the evolution of the hypervolume as we did earlier for the bTSP.

Experimental Evaluation of Regular Anytime Two-Phase Local Search on Bi-objective Permutation Flowshop Scheduling Problem

We examine the quality of the result of each TPLS variant, *1to2*, *2to1*, D-TPLS, and RA-TPLS during the run of the algorithm. Figure 3.2 shows the development of the hypervolume of each TPLS variant, averaged across 15 independent runs. The plots show that the hypervolume value of *1to2*, *2to1*, and D-TPLS is rather poor up to the point that the sequence of weights reaches the other objective. On the other hand, RA-TPLS quickly reaches a high hypervolume in very few scalarizations. In terms of final quality, however, D-TPLS clearly performs better than RA-TPLS as soon as the former reaches half of its scalarizations and starts performing scalarizations back from the second to the first one. This fact and the differences between *1to2* and *2to1* strongly indicate that, for the bPFSPs considered here, the starting objective plays a significant role on both the anytime behavior and the final solution quality.

3.4 Adaptive TPLS

The TPLS variants discussed so far generate a sequence of weights that is determined by the number of scalarizations, and aims to allocate the same computational effort to all regions of the Pareto front. This strategy, however, may not be adequate when the underlying single-objective algorithm shows a different performance for each objective, and the shape of the Pareto front is not regular in all search directions. Therefore, we proposed an adaptive TPLS variant that dynamically generates weights in order to adapt the search to the shape of the Pareto front. In this section, we explain this adaptive TPLS variant and discuss some possible improvements.

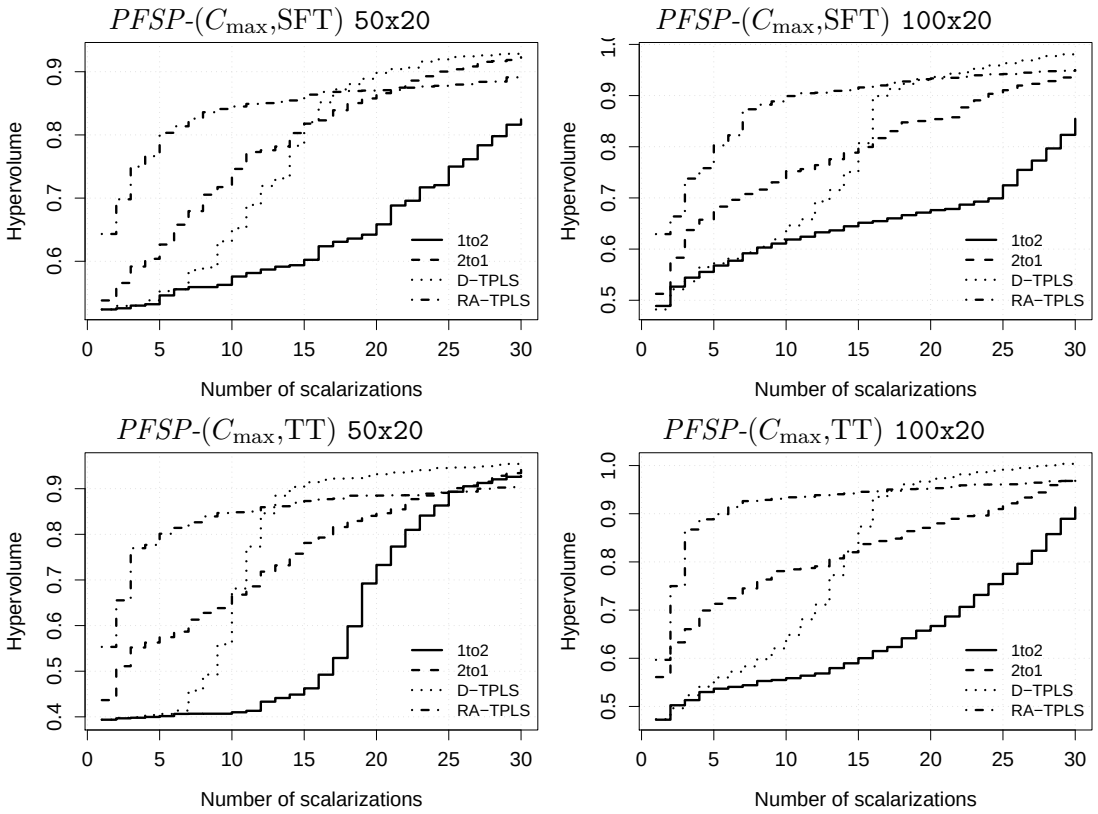

Figure 3.2: Development of the average hypervolume over the number of scalarizations for *1to2*, *2to1*, D-TPLS, and RA-TPLS for bPFSP. Results are given for one instance of size 50x20 (left column) and one instance of size 100x20 (right column). The problems are *PFSP*-($C_{\max}$,SFT) (top plots) and *PFSP*-($C_{\max}$,TT) (bottom plots).

3.4.1 Adaptive Anytime Strategy

Our *adaptive* TPLS is inspired by the *dichotomic* scheme proposed by Aneja and Nair (1979) for exact algorithms and used for the approximate case by Lust and Teghem (2010b). 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 segment (henceforth denoted by an overline) defined by s_1 and s_2 in the objective space, that is, assuming the range of the objectives are normalized or equal, we have

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

The *dichotomic* scheme 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, leading to a poor distribution of solutions and, hence, to poor anytime behavior. Moreover, Lust and Teghem (2010b) apply the *dichotomic* scheme as a *Restart* strategy that starts each scalarization from a newly generated initial solution. In the exact case, the algorithm of Aneja and Nair (1979) is deterministic, and, hence, applying the same weight results in the same output. Also, they did not consider the concept of seeding a scalarization. Our extension of the *dichotomic* strategy to the TPLS framework makes effective use of solutions found by previous scalarizations to seed later scalarizations and satisfies the *anytime* property (see Section 2.9). We describe this *adaptive* TPLS strategy as Algorithm 10.

The main additional data structure in Algorithm 10 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 $(s_{\text{sup}}, s_{\text{inf}}) \in S$ whose images define the "largest gap" in the objective space, using a given norm $\|(\vec{f}(s), \vec{f}(s')\|$ to compare every pair of solutions. The idea is to focus the search on the largest gap in the Pareto front in order to obtain a well-spread set of non-dominated solutions. This is different from the original *dichotomic* scheme, which explores segments recursively. In what follows we propose to use as norm the Euclidean distance in the normalized objective space. Later on, we propose and test a new alternative in Section 3.6. After choosing the pair of solutions

Algorithm 10 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: $(s_{\text{sup}}, s_{\text{inf}}) := \arg\max_{(s,s')\in S} \|(\vec{f}(s), \vec{f}(s')\|$
7: Calculate λ perpendicular to $\overline{\vec{f}(s_{\text{sup}})\vec{f}(s_{\text{inf}})}$ following Eq. 3.2
8: **if** `one_seed_case` **then**
9: $s := \mathsf{ChooseRandomly}(s_{\text{sup}}, s_{\text{inf}})$
10: $s' := \mathsf{SLS}_\Sigma(s, \lambda)$
11: Add s' to *Archive*
12: $\mathsf{Update}(S, s')$
13: **else**
14: $s'_{\text{sup}} := \mathsf{SLS}_\Sigma(s_{\text{sup}}, \lambda)$
15: $s'_{\text{inf}} := \mathsf{SLS}_\Sigma(s_{\text{inf}}, \lambda)$
16: Add s'_{sup} and s'_{inf} to *Archive*
17: $\mathsf{Update}(S, s'_{\text{sup}})$
18: $\mathsf{Update}(S, s'_{\text{inf}})$
19: **end if**
20: **end while**
21: $\mathsf{RemoveDominated}(\mathit{Archive})$
22: **Output:** *Archive*

$(s_{\text{sup}}, s_{\text{inf}})$ according to the norm (line 6), the algorithm calculates a new weight λ perpendicular to the segment $\overline{\vec{f}(s_{\text{sup}})\vec{f}(s_{\text{inf}})}$ in the objective space (line 7), following Eq. 3.2. Next, the underlying single-objective SLS algorithm, SLS_Σ, is run using the weight λ either once, starting from either s_{sup} and s_{inf} (lines 9 to 12), or twice, starting one time from solution s_{sup} and one time from solution s_{inf} (lines 14 to 18). Which of these two possibilities is chosen, depends on the parameter `one_seed_case`.

In the last step of an iteration, procedure Update updates the set of initial solutions S using the new solutions found. If s' is a new solution, any single solution in S dominated by s' is replaced by s', and any pair of solutions (weakly) dominated by s' is removed. The *dichotomic* scheme (Aneja and Nair, 1979; Lust and Teghem, 2010b) only accepts solutions for inclusion in S if they lie *within* the triangle defined in the objective space by the solutions s_{sup} and s_{inf}, and their local ideal point, which is the point $(f_1(s_{\text{sup}}), f_2(s_{\text{inf}}))$ (see Figure 3.3). Heuristic algorithms may, however, generate so-

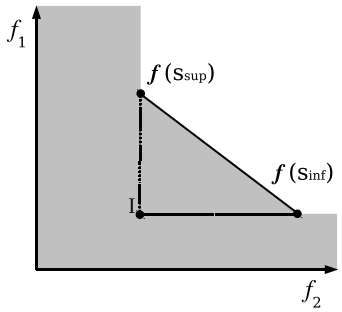

Figure 3.3: Only solutions in the gray area are accepted as initial solutions for further scalarizations (See the text for details).

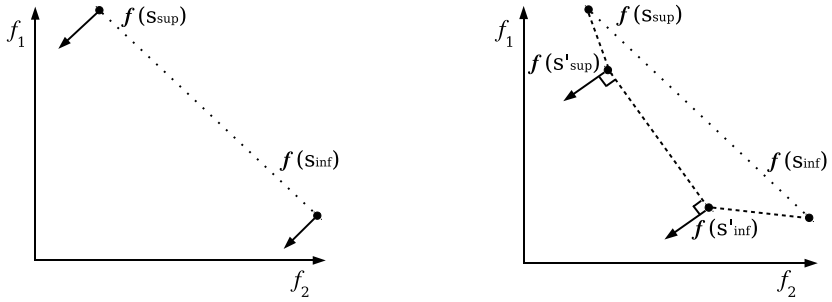

Figure 3.4: A single iteration of the Adaptive "Anytime" TPLS strategy with two initial solutions (AN-TPLS-2seed). On the left the state before the iteration and on the right after S has been updated. The next segment that will be considered is $(s'_{\text{sup}}, s'_{\text{inf}})$ because of its larger distance.

lutions that are in the gray area *outside* the triangle. Solutions outside the gray area are either dominated or not supported (that is, non-dominated but not optimal for any scalarization); 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 pair $(s_1, s_2) \in S$ with $f_1(s_1) < f_1(s') < f_1(s_2)$ is removed, and two new pairs (s_1, s') and (s', s_2) are added to S. Since each iteration produces at most two new solutions (s'_{sup} and s'_{inf}, or simply s'_1 in the one-seed variant), a maximum of three new pairs are added to S every iteration. Figure 3.4 shows an example of the update of S after one iteration of the *adaptive* algorithm. We call the algorithm that uses two initial solutions AN-TPLS-2seed in what follows (for adaptive normal TPLS), and we call AN-TPLS-1seed its variant using only one initial solution (in the outline of Algorithm 10, this corresponds to `one_seed_case=true`).

3.4.2 Experimental Evaluation of Adaptive Anytime TPLS on Bi-objective Traveling Salesman Problem Instances

To evaluate the performance of AN-TPLS-2seed and its variant AN-TPLS-1seed, we use the same experimental setup described in Section 3.3. We present the average hypervolume evolution of AN-TPLS-2seed and AN-TPLS-1seed in Fig. 3.5, comparing it to D-TPLS and RA-TPLS.

For the two isometric instances, AN-TPLS-1seed appears to be better than AN-TPLS-2seed. By checking carefully the output, we noticed that the underlying ILS algorithm usually finds two solutions whose images are very close to each other in the objective space or possibly even the same. Hence, using two initial solutions gives a negligible improvement with respect to the hypervolume in comparison to using a single one.

For the two anisometric instances, AN-TPLS-1seed is again the best strategy. Interestingly, one can see that the higher the value of $\max_{dist}$, the closer is the performance of RA-TPLS to AN-TPLS-1seed. This is due to the fact that by increasing $\max_{dist}$, instances are "smoother" in the sense they resemble more the isometric ones and, therefore, there is less need to adapt the weights to the particular shape of the Pareto front.

3.4.3 Experimental Evaluation of Adaptive Anytime TPLS on Bi-objective Permutation Flowshop Scheduling Problem

To test on a problem that has a front resulting from objectives with different properties, we use again the bPFSPs that were already described in Section 3.1, page 52. Results are presented in Fig. 3.6. In contrast with the results on the bTSP, AN-TPLS-2seed clearly outperforms RA-TPLS and it is also significantly better than AN-TPLS-1seed. Still, D-TPLS shows on several instances better final performance than AN-TPLS-2seed according to the hypervolume indicator.

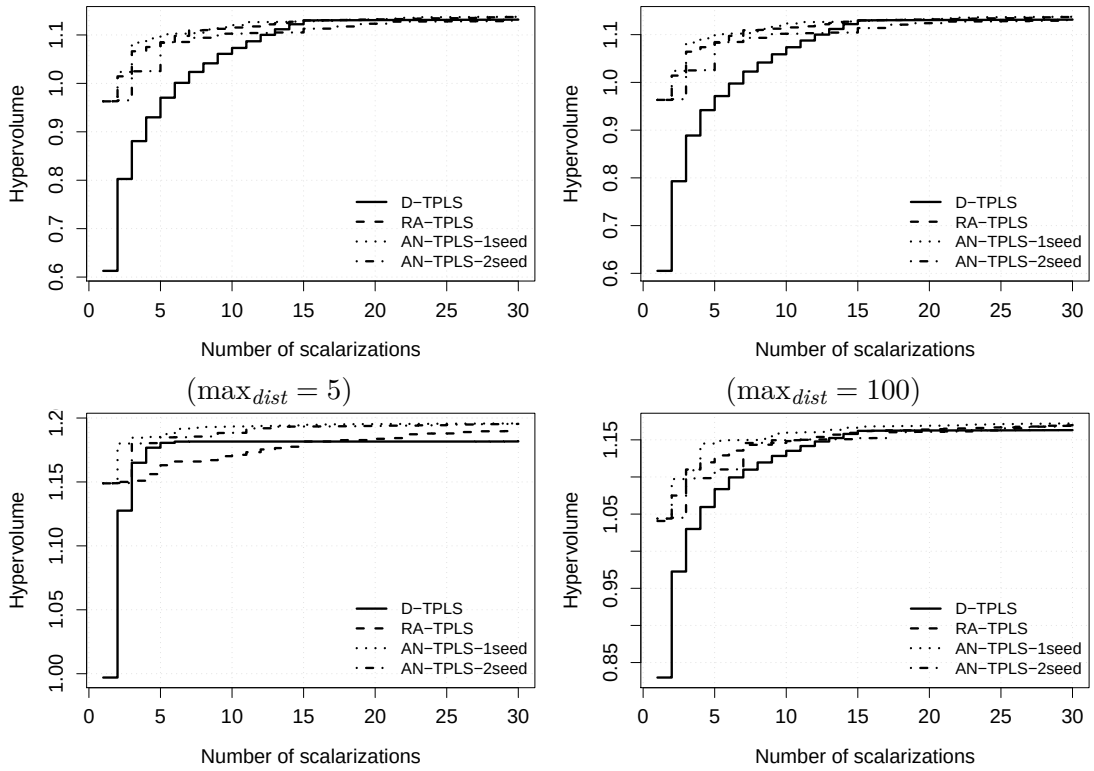

Figure 3.5: Development of the hypervolume over the number of scalarizations for D-TPLS, RA-TPLS, AN-TPLS-2seed and AN-TPLS-1seed for two isometric TSP instances and two anisometric TSP instances. Anisometric instances with intermediate value of $\max_{dist}$ show a compromise trend between the two extremes shown here (see supplementary material in Dubois-Lacoste *et al.* (2010a)).

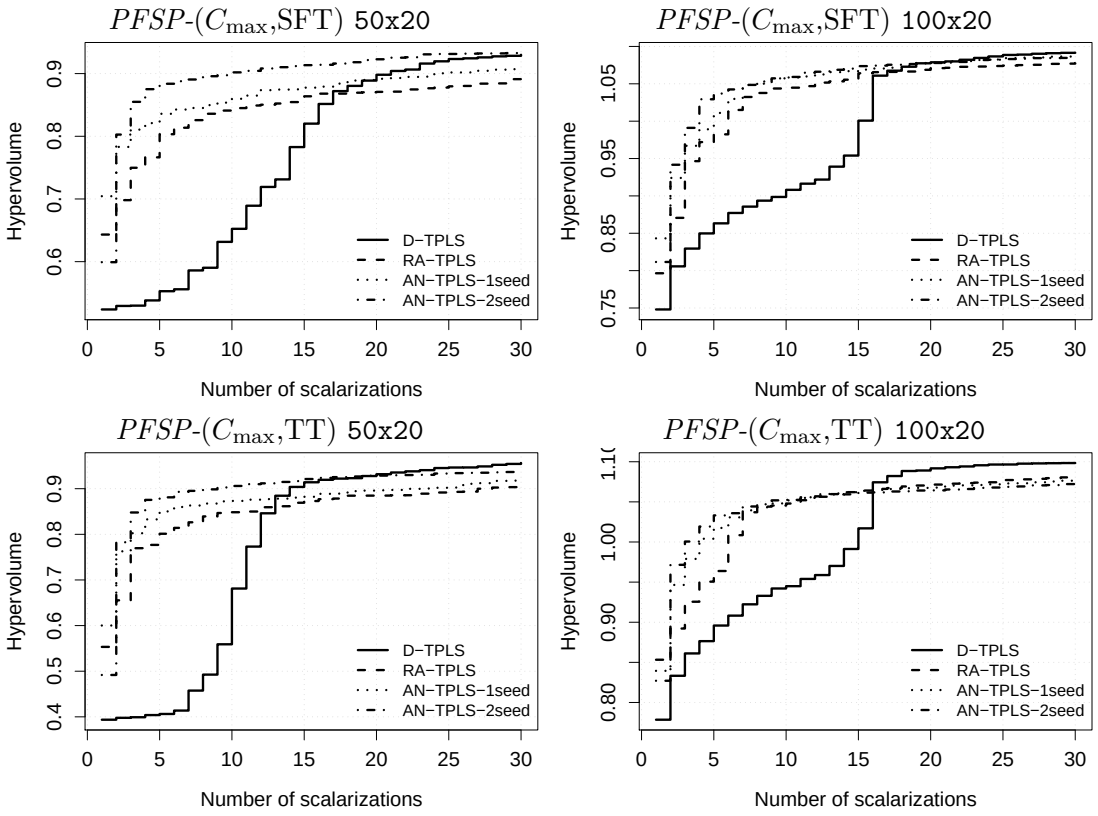

Figure 3.6: Development of the hypervolume over the number of scalarizations for D-TPLS, RA-TPLS, AN-TPLS-2seed and AN-TPLS-1seed for bPFSP. Results are given for one instance of size 50x20 (left column) and one instance of size 100x20 (right column). The problems are *PFSP*-($C_{\max}$,SFT) (top plots) and *PFSP*-($C_{\max}$,TT) (bottom plots).

3.4.4 Further Analysis of AN-TPLS-1seed and AN-TPLS-2seed

The results of the evaluation of Adaptive TPLS show that AN-TPLS-1seed is the best strategy for bTSPs while AN-TPLS-2seed is the best strategy for bPFSPs. In this section, we illustrate the fact that the different behavior of the underlying single-objective algorithms for each problem leads to such results. To do so, we examine in detail the scalarizations performed by AN-TPLS-2seed for the first three weights and each of the two seeds. We perform 15 independent runs of each scalarization, and we plot the objective vectors of the solutions obtained. Figure 3.7 presents the results for an isometric bTSP instance. The left plot shows the objective vectors of the solutions obtained with the first weight, using each of the two initial solutions as seed. The line indicates the direction of the search (the weight vector). The objective vector obtained by each of the 15 independent runs is represented by the same symbol as the seed used to initialize the corresponding run. In this case, all points overlap, that is, the resulting objective vectors fall into a tiny area of the objective space, independently of which initial solution was used as seed. The middle and right plots of Fig. 3.7 show the results of the same experiment for the second and the third weight, respectively. Hence, there is no significant advantage in using two different initial solutions with the same weight with respect to increasing the hypervolume. We observed the same behavior on the anisometric bTSP instances.

We repeat the experiment for the bPFSP (using IG as the underlying single-objective algorithm) and present the results in Fig. 3.8. The situation is very different now. As the plots show, there is a larger variability on the location of the resulting objective vectors, even for those solutions obtained from the same seed. More importantly, the vectors resulting from the two different seeds are located in two clearly distinct clusters, despite all being solutions for the same scalarized problem. Hence, performing a scalarization from two different seeds is advantageous in the case of the bPFSP in order to obtain a better distribution of solutions along the Pareto front and, hence, a higher value of the hypervolume indicator.

This insight might be used to define when it is better to use one or two initial solutions. A simple way to choose automatically AN-TPLS-1seed or AN-TPLS-2seed would be to use AN-TPLS-2seed for the first weight, and consider the Euclidean distance between the objective vectors of the two solutions obtained. If the distance is large enough,

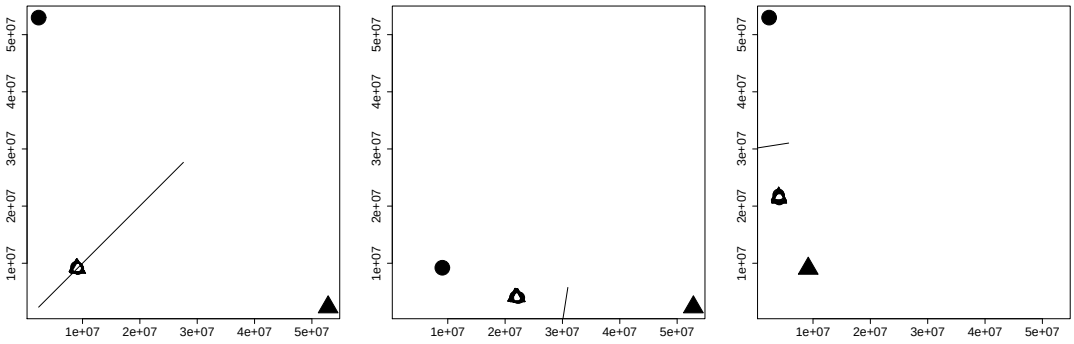

Figure 3.7: Distribution of objective vectors in the objective space after the first (left plot), the second (middle plot) and third (right plot) weights, for an isometric bTSP instance. The two vectors plotted with filled, black symbols (circle or triangle) are the initial ones, used to define the weight (the direction is represented with a line) and used as seeds. The symbol of each point denotes the seed from which they have been obtained.

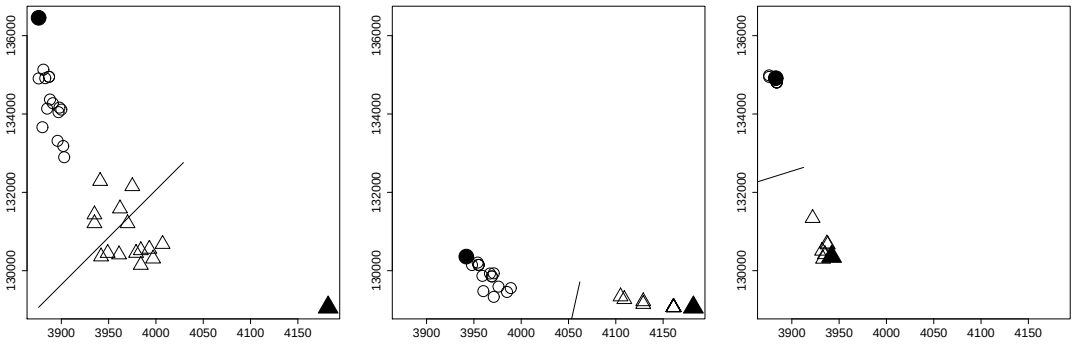

Figure 3.8: Distribution of objective vectors in the objective space after the first (left plot), the second (middle plot) and third (right plot) weights, for a *PFSP*-($C_{\max}$,SFT) instance. The two vectors plotted with filled, black symbols (circle or triangle) are the initial ones, used to define the weight (the direction is represented with a line) and used as seeds. The symbol of each point denotes the seed from which they have been obtained.

it might be advantageous to continue with AN-TPLS-2seed; while if points are very close, it might be better to switch to AN-TPLS-1seed. Another possibility to choose the best strategy would be to launch both strategies, allowing two scalarizations for each. The search would then continue with the strategy that leads to the larger hypervolume. We leave the evaluation of these ideas for future investigation.

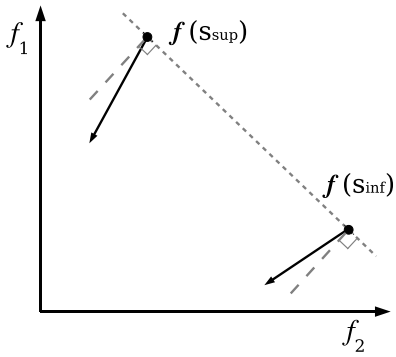

Figure 3.9: AF-TPLS strategy: the two weights are "focused" to the center of the segment.

3.4.5 Adaptive Focus TPLS

In AN-TPLS-2seed, when two adjacent segments in S are almost parallel, two scalarizations are solved using the same initial solution (the solution shared by the two segments) and very similar weights (because the two vectors perpendicular to the segments will again be almost parallel). A careful analysis of AN-TPLS-2seed showed that such situation actually occurs, and may result in negligible progress of the search. In order to avoid this problem, one can focus the search direction of each scalarization towards the center of each segment and further improve the results of AN-TPLS-2seed. We call this variant *adaptive focus* TPLS (AF-TPLS).

Given a segment $\overline{\vec{f}(s_1)\vec{f}(s_2)}$, with $f_1(s_1) < f_1(s_2)$, AF-TPLS generates 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 computed by Eq. 3.2, and θ is a parameter that modifies λ towards the center of the segment (see Fig. 3.9).

These two new weights replace the weight λ in Algorithm 10, that is, the run of the SLS algorithm that uses s_1 as initial solution solves a scalarization according to weight λ_1, while the run starting from s_2 uses the weight λ_2. A value of $\theta = 0$ would reproduce the AN-TPLS-2seed strategy.

3.4.6 Experimental Evaluation of Adaptive Focus on Bi-objective Permutation Flowshop Scheduling Problem

We test AF-TPLS on the bPFSP using different values of $\theta = \{0.05, 0.15, 0.25, 0.5\}$. Plots that present a comparison of AF-TPLS using these values are provided as supplementary material (Dubois-Lacoste *et al.*, 2010a). The values 0.25 and 0.15 show a similar performance, indicating a relative robustness of AF-TPLS with respect to the setting of θ. For the following comparisons, we consider only AF-TPLS using 0.25. In Fig. 3.10, we present a comparison of AF-TPLS, AN-TPLS-2seed and D-TPLS. AF-TPLS is at least as good as AN-TPLS-2seed, and it is often able to outperform it. Furthermore, AF-TPLS reaches a final quality often equal to or better than the one reached by D-TPLS. The instance `100x20_1` of problem *PFSP*-($C_{\max}$,TT) (bottom-right in Fig. 3.10) is actually the only instance of the 20 we tested where D-TPLS performs clearly better than AF-TPLS.

3.5 Statistical Analysis

We have examined so far the results of the different approaches by comparing their performance in each instance. In order to assess the performance over the whole set of instances, we perform the following statistical analysis on each problem. The analysis is based on the Friedman test for analyzing non-parametric unreplicated complete block designs, and its associated post-test for multiple comparisons (Conover, 1999). First, we calculate the mean hypervolume of the 15 runs of each algorithm for each instance. Then, we perform the Friedman test using the ten instances as the blocking factor, and the different strategies as the treatment factor. In most cases, the Friedman test rejects the null hypothesis with a p-value lower than 0.05. Then, we rank the strategies per instance according to the mean hypervolume, the lower rank the better. From this ranking we calculate the difference (ΔR) between the sum of ranks of each strategy and the best ranked one (with the lowest sum of ranks). Finally, we 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 indicate in bold face the best strategy (the one having the lowest sum of ranks) and those that are not significantly different from the best one.

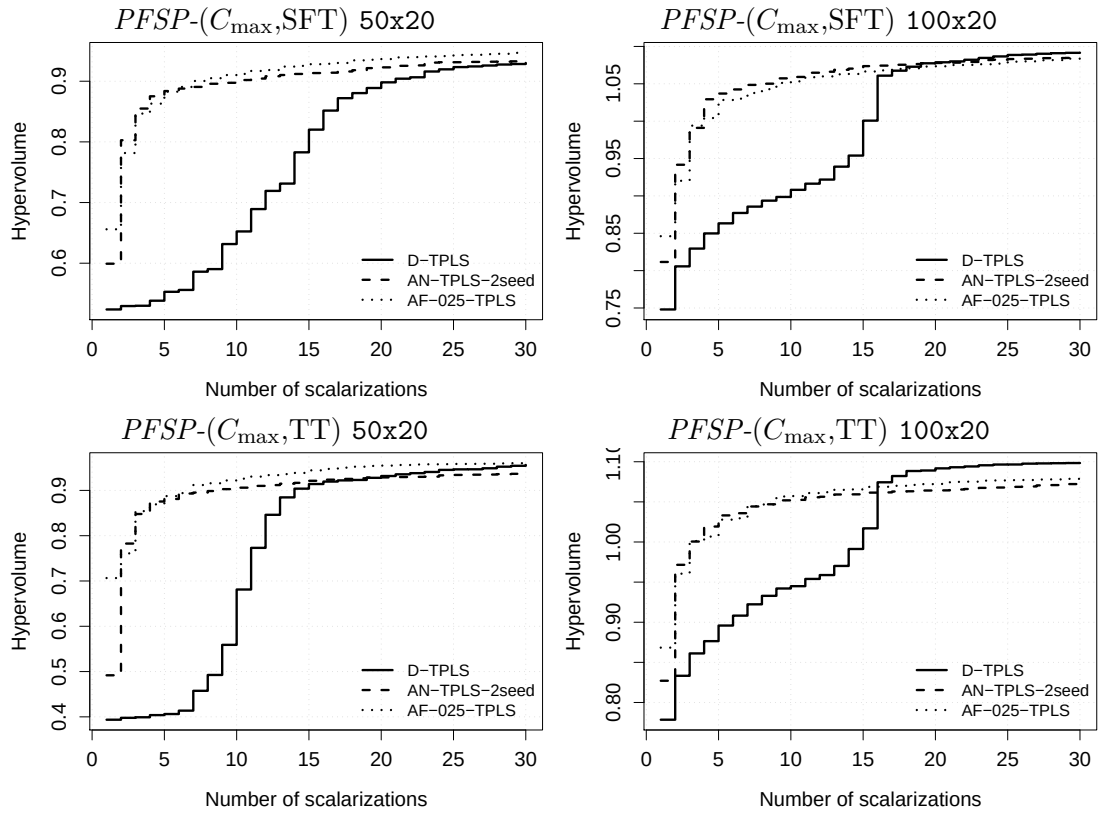

Figure 3.10: Development of the hypervolume over the number of scalarizations for D-TPLS, AN-TPLS-2seed and AF-TPLS. Results are given for one instance of size 50x20 (left column) and one instance of size 100x20 (right column). The problems are *PFSP*-($C_{\max}$,SFT) (top plots) and *PFSP*-($C_{\max}$,TT) (bottom plots).

Table 3.1: Statistical analysis for the isometric bTSP. 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. The strategy that is significantly better than the other ones is indicated in bold face. For isometric bTSP the ordering of all strategies is the same on all instances and, hence, $\Delta R_\alpha = 0$.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
10	0	**AN-TPLS-1seed**, RA-TPLS (10), AN-TPLS-2seed (20), D-TPLS (30), *1to2*(40)
20	0	**AN-TPLS-1seed**, RA-TPLS (10), D-TPLS (20), AN-TPLS-2seed (30), *1to2*(40)
30	0	***1to2***, RA-TPLS (10), AN-TPLS-1seed (20), D-TPLS (30), AN-TPLS-2seed (40)

3.5.1 Results on the Bi-objective Traveling Salesman Problem

We perform the statistical tests after the algorithms have performed 10, 20 and 30 scalarizations. We compare the strategies *1to2*, D-TPLS, RA-TPLS, AN-TPLS-2seed and AN-TPLS-1seed. We do not consider AF-TPLS since it leads to poor performance for bTSP instances (see the supplementary material in Dubois-Lacoste *et al.* (2010a)).

Results are given in Table 3.1 for isometric instances and in Table 3.2 for anisometric ones. When the value of the critical difference (ΔR_α) is equal to 0, the strategies have the same ranking over all instances. The numbers in parenthesis are the difference of ranks relative to the best strategy. For isometric instances, AN-TPLS-1seed is the best strategy before completion. However, when algorithms run until completion, *1to2* is significantly better than the other ones. For anisometric instances, we performed independent tests for each value of $\max_{dist}$ and we found that the results are consistent across the different values of $\max_{dist}$. AN-TPLS-1seed is always significantly better than all the other strategies. Hence, it is the strategy that should be used for anisometric instances, no matter the value of $\max_{dist}$.

Table 3.2: Statistical analysis for the anisometric bTSP. 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. The strategy that is significantly better than the other ones is indicated in bold face.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
$\max_{dist} = 5$		
10	3.31	**AN-TPLS-1seed**, AN-TPLS-2seed (14), *1to2*(16), D-TPLS (30), RA-TPLS (40)
20	2.03	**AN-TPLS-1seed**, AN-TPLS-2seed (10), *1to2*(20), RA-TPLS (31), D-TPLS (39),
30	3.31	**AN-TPLS-1seed**, AN-TPLS-2seed (10), *1to2*(24), RA-TPLS (26), D-TPLS (40)
$\max_{dist} = 10$		
10	0	**AN-TPLS-1seed**, D-TPLS (10), AN-TPLS-2seed (20), RA-TPLS (30), *1to2*(40)
20	2.03	**AN-TPLS-1seed**, AN-TPLS-2seed (11), *1to2*(19), RA-TPLS (30), D-TPLS (40)
30	0	**AN-TPLS-1seed**, AN-TPLS-2seed (10), *1to2*(20), RA-TPLS (30), D-TPLS (40)
$\max_{dist} = 25$		
10	0	**AN-TPLS-1seed**, AN-TPLS-2seed (10), RA-TPLS (20), D-TPLS (30), *1to2*(40)
20	3.31	**AN-TPLS-1seed**, AN-TPLS-2seed (10), RA-TPLS (24), D-TPLS (26), *1to2*(40)
30	4.11	**AN-TPLS-1seed**, AN-TPLS-2seed (14), *1to2*(17), RA-TPLS (29), D-TPLS (40)
$\max_{dist} = 100$		
10	0	**AN-TPLS-1seed**, AN-TPLS-2seed (10), RA-TPLS (30), D-TPLS (20), *1to2*(40)
20	3.31	**AN-TPLS-1seed**, AN-TPLS-2seed (10), RA-TPLS (24), D-TPLS (26), *1to2*(40)
30	4.11	**AN-TPLS-1seed**, AN-TPLS-2seed (14), *1to2*(17), RA-TPLS (29), D-TPLS (40)

3.5.2 Results on the Bi-objective Permutation Flowshop Scheduling Problem

For the bPFSP, we perform the same procedure separately for each combination of objectives, each instance size `50x20` and `100x20`, and we measure the hypervolume after 10, 20 and 30 scalarizations. We compared D-TPLS, RA-TPLS, AN-TPLS-2seed, AN-TPLS-1seed and AF-TPLS (with $\theta = 0.25$). The results are given in Table 3.3.

For a low number of scalarizations, the *adaptive* strategies (AN-TPLS-2seed 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 scalariza-

Table 3.3: Statistical analysis for the bPFSP. 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 that are not significantly different from the best one are indicated in bold face. See the text for details.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
($C_{\max}$, $\sum C_i$) 50x20		
10	5.41	**AF-TPLS**, AN-TPLS-2seed (6), AN-TPLS-1seed (12), RA-TPLS (26), D-TPLS (36)
20	7.65	**AN-TPLS-2seed, AF-TPLS (2)**, AN-TPLS-1seed (20), D-TPLS (21), RA-TPLS (32)
30	9.63	**AN-TPLS-2seed, AF-TPLS (3), D-TPLS (4)**, AN-TPLS-1seed (13), RA-TPLS (30)
($C_{\max}$, $\sum C_i$) 100x20		
10	6.51	**AF-TPLS, AN-TPLS-2seed (4), AN-TPLS-1seed (5)**, RA-TPLS (23), D-TPLS (33)
20	9.91	**AF-TPLS, AN-TPLS-2seed (2)**, AN-TPLS-1seed (11), D-TPLS (18), RA-TPLS (29)
30	9.44	**D-TPLS, AF-TPLS (8)**, AN-TPLS-2seed (16), AN-TPLS-1seed (27), RA-TPLS (29)
($C_{\max}$, $\sum T_i$) 50x20		
10	3.88	**AF-TPLS**, AN-TPLS-2seed (5), AN-TPLS-1seed (16), RA-TPLS (27), D-TPLS (37)
20	5.36	**AF-TPLS**, AN-TPLS-2seed (13), D-TPLS (23), AN-TPLS-1seed (24), RA-TPLS (40)
30	5.76	**AF-TPLS, D-TPLS (1)**, AN-TPLS-2seed (11), AN-TPLS-1seed (25), RA-TPLS (33)
($C_{\max}$, $\sum T_i$) 100x20		
10	4.97	**AF-TPLS**, AN-TPLS-2seed (14), AN-TPLS-1seed (14), RA-TPLS (28), D-TPLS (39)
20	10.36	**AF-TPLS**, AN-TPLS-2seed (13), D-TPLS (19), AN-TPLS-1seed (22), RA-TPLS (31)
30	8.42	**D-TPLS, AF-TPLS (2)**, AN-TPLS-2seed (21), RA-TPLS (23), AN-TPLS-1seed (29)

tions), while the opposite is true two times. In conclusion, AF-TPLS would be the strategy of choice for bPFSP problems.

3.6 Optimistic Hypervolume Contribution as Selection Criterion

The main idea of the adaptive anytime strategies is to focus the search on the most promising region of the objective space for improving the quality of the Pareto front approximation. In this sense, the algorithm aims at filling the "largest gaps" in the Pareto front approximation. In order to measure the "size of the gap", we use a norm

as described in line 6 of Algorithm 10 (Section 3.4), where the pair of solutions that maximizes it are selected as seeds for the next scalarization.

For all experiments presented so far, we have used as norm the Euclidean distance on the normalized objective space. Although this distance leads to a good "visual" distribution of solutions, it may not lead to the selection of the seeds with the maximum potential of improving quality. A measure of the quality of the current Pareto front approximation is the hypervolume, and therefore, we could select the pair of seeds that may lead to the largest improvement of the hypervolume. Assuming that the new solution found is within the rectangle defined in the objective space by the two seeds, the maximum improvement in terms of hypervolume is proportional to the area of this rectangle. Hence, using normalized objective values, we compute this norm as follows:

$$\mathsf{ohvc}(s, s') = |((f_1(s) - f_1(s')) \cdot ((f_2(s) - f_2(s'))| \tag{3.4}$$

This optimistic hypervolume contribution is different from measuring the actual contribution to the hypervolume of each solution that are already in the current archive for the purposes of selecting or discarding some solutions (Knowles and Corne, 2003b; Beume *et al.*, 2007).

We compare this optimistic hypervolume contribution with the Euclidean distance as the selection criterion in AN-TPLS-1seed, which is the best adaptive TPLS strategy for the isometric and the anisometric bTSP, and AF-TPLS, which is the best adaptive TPLS strategy for the bPFSP.

Figure 3.11 shows the development of the hypervolume of the resulting adaptive TPLS variants. The version of AN-TPLS-1seed that uses the ohvc norm is slightly but consistently better than the one that uses the Euclidean distance. For AF-TPLS on the bPFSP, results are not as consistent as for the bTSP (additional plots are available as supplementary material).

To assess the statistical significance of the differences between the two selection criteria over all instances, we perform the same statistical analysis as in the previous section. For the isometric bTSP, we compare in Table 3.4 the quality of AN-TPLS-1seed, its variant that uses the ohvc norm (AN-TPLS-1seed$_{\mathcal{HV}}$), and *1to2*, which outperformed all other strategies in terms of final quality. We compare in Table 3.5 AN-TPLS-1seed, AN-

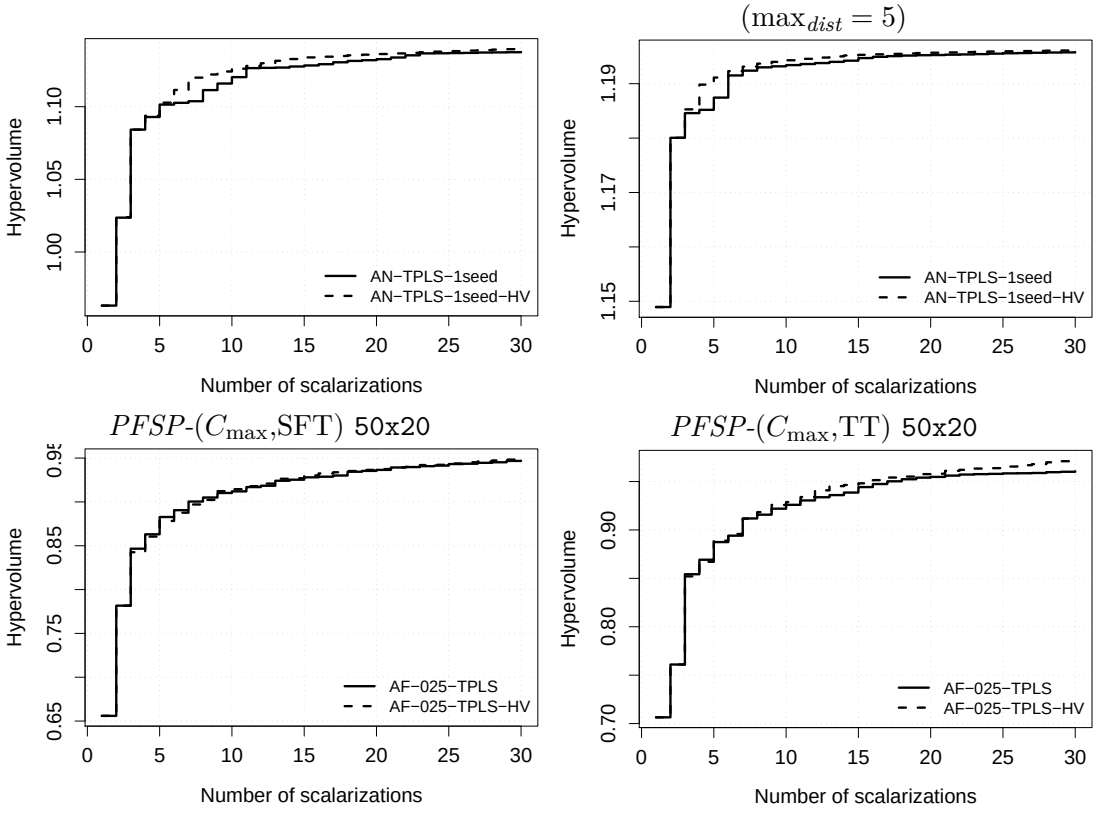

Figure 3.11: Development of the hypervolume over the number of scalarizations for AN-TPLS-1seed using Euclidean distance and ohvc for one isometric TSP instance (top-left), one anisometric TSP instance (top-right), and one instance of bPFSP with the two different combinations of objectives.

Table 3.4: Statistical analysis for the isometric bTSP. For each number of scalarizations, strategies are ordered according to the rank obtained.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
10	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), *1to2*(20)
20	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), *1to2*(20)
30	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, *1to2*(10), AN-TPLS-1seed (20)

TPLS-1seed$_{\mathcal{HV}}$, and D-TPLS, for the anisometric bTSP. The results are consistent for the two types of instances, and all values of max$_{dist}$. AN-TPLS-1seed$_{\mathcal{HV}}$ is the best-ranked strategy and it is always significantly better than all the other ones, including *1to2*.

In the case of the bPFSP, Table 3.6 compares the two adaptive strategies AN-TPLS-2seed and AF-TPLS, their variants using the ohvc norm, and D-TPLS. The improvement is not as consistent as for the bTSP. However, AF-TPLS$_{\mathcal{HV}}$ is most often the best-ranked strategy and never significantly worse than the best ranked one.

3.7 Graphical Analysis Based on the Difference of Empirical Attainment Functions

We further explore the differences between RA-TPLS, AF-TPLS and D-TPLS by examining the empirical attainment functions (EAF) of the final results after 30 scalarizations. As mentioned in section 2.6.2, page 27, the EAF of an algorithm provides the probability, estimated from several runs, of an arbitrary point in the objective space being attained by (dominated by or equal to) a solution obtained by a single run of the algorithm. Examining the differences between the EAFs of two algorithms allows us to identify regions of the objective space where one algorithm performs better than another. Given a pair of algorithms, the differences in favor of each algorithm are plotted side-by-side and the magnitude of the difference is encoded in gray levels. For more details, we refer to Chapter 2, Section 2.6.2.

Table 3.5: Statistical analysis for the anisometric bTSP. 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. The strategy that is significantly better than the other ones is indicated in bold face.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
$\max_{dist} = 5$		
10	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
20	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
30	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
$\max_{dist} = 10$		
10	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
20	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
30	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
$\max_{dist} = 25$		
10	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
20	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
30	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
$\max_{dist} = 100$		
10	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
20	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)
30	0	**AN-TPLS-1seed$_{\mathcal{HV}}$**, AN-TPLS-1seed (10), D-TPLS (20)

Table 3.6: Statistical analysis for the bPFSP. 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 that are not significantly different to the best one are indicated in bold face. See the text for details.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
$(C_{\max}, \sum C_i)$ 50x20		
10	3.87	**AF-TPLS, AF-TPLS$_{\mathcal{HV}}$ (3)**, D-TPLS (16.5)
20	6.80	**AF-TPLS$_{\mathcal{HV}}$, AF-TPLS (3)**, D-TPLS (13.5)
30	8.23	**AF-TPLS$_{\mathcal{HV}}$, AF-TPLS (4)**, D-TPLS (11)
$(C_{\max}, \sum C_i)$ 100x20		
10	3.96	**AF-TPLS$_{\mathcal{HV}}$**, AF-TPLS (4), D-TPLS (17)
20	7.07	**AF-TPLS$_{\mathcal{HV}}$, AF-TPLS (2)**, D-TPLS (13)
30	pval>0.05	**D-TPLS, AF-TPLS$_{\mathcal{HV}}$, AF-TPLS**
$(C_{\max}, \sum T_i)$ 50x20		
10	4.54	**AF-TPLS$_{\mathcal{HV}}$**, AF-TPLS (4), D-TPLS (17)
20	4.54	**AF-TPLS$_{\mathcal{HV}}$, AF-TPLS (4)**, D-TPLS (17)
30	pval>0.05	**AF-TPLS$_{\mathcal{HV}}$, AF-TPLS, D-TPLS**
$(C_{\max}, \sum T_i)$ 100x20		
10	3.96	**AF-TPLS, AF-TPLS$_{\mathcal{HV}}$ (6)**, D-TPLS (18)
20	6.86	**AF-TPLS, AF-TPLS$_{\mathcal{HV}}$ (10)**, D-TPLS (14)
30	pval>0.05	**AF-TPLS, D-TPLS, AF-TPLS$_{\mathcal{HV}}$**

3.7.1 Graphical Analysis on Bi-objective Traveling Salesman Problem

For the isometric bTSP, we compare the best strategy AN-TPLS-1seed, with the second best, *1to2*. Figure 3.12 shows the differences in the EAFs of these two strategies for one isometric instance. In the top plot, we show the differences after 15 scalarizations out of 30, whereas the bottom plot compares the final quality. The EAF differences after 15 scalarizations show that *1to2* simply does not cover a significant part of the objective space, whereas AN-TPLS-1seed covers the front equally in all directions. Here we color in black the region of the objective space attained by more than 80% of the runs of each algorithm, to help to visualize this behavior. This plot and the ones for

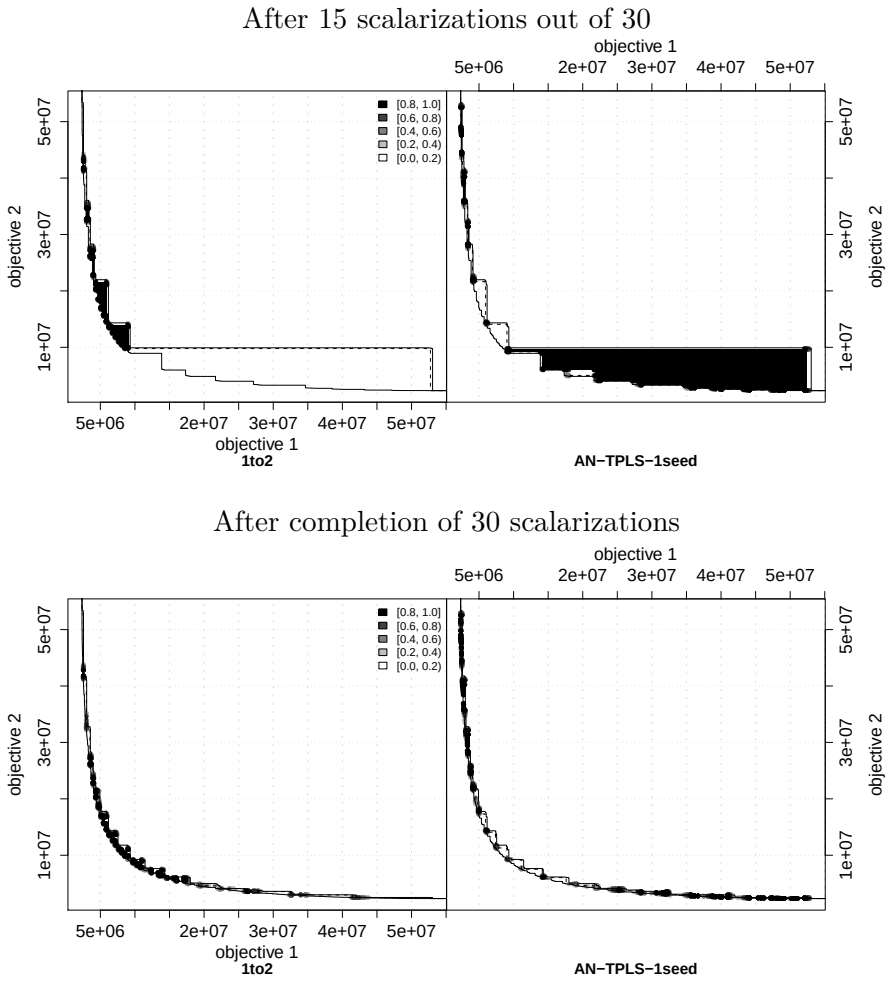

Figure 3.12: EAF differences for one isometric TSP instance, after 15 scalarizations (top) and after 30 scalarizations (bottom). Strategies are *1to2* (left) and AN-TPLS-1seed (right). Plots for other instances are available in Dubois-Lacoste *et al.* (2010a).

other instances show very clearly the lack of the anytime property in *1to2*, and the much better anytime behavior of AN-TPLS-1seed. The EAF differences after completion of the 30 scalarizations show differences in favor of both algorithms along the whole Pareto front. In this case, *1to2* appears to be better in the center of the Pareto front, whereas the adaptive TPLS finds better solutions along the extremes. Similar results are observed for other instances (see Dubois-Lacoste *et al.* (2010a)).

For anisometric instances, we first give in Fig. 3.13 plots that show the EAF differences between AN-TPLS-2seed and AN-TPLS-1seed. These plots support the conclusion from the statistical test, namely that AN-TPLS-1seed is better than AN-TPLS-2seed for this problem, which is true also when varying $\max_{dist}$. In comparison with *1to2* (the best among the classical TPLS strategies for this problem), AN-TPLS-1seed is clearly better for a small values of $\max_{dist}$ (top plot of Fig. 3.14), whereas for large values of $\max_{dist} = 100$ (bottom plot of Fig. 3.14), the results are similar to the ones obtained in the isometric bTSP instances.

3.7.2 Graphical Analysis on Bi-objective Quadratic Assignment Problem

Figure 3.15 illustrates the EAF differences between AF-TPLS and *1to2* on one instance for *PFSP*-($C_{\max}$,SFT), after 15 scalarizations out of 30 and after completing the 30 scalarizations. In both cases, there are strong differences in favor of AF-TPLS.

3.8 Summary

TPLS is a key component of effective bi-objective optimization algorithms (Dubois-Lacoste *et al.*, 2009; Lust and Teghem, 2010b). However, the originally proposed TPLS framework has an important drawback: it requires to know in advance the available computation time to distribute appropriately the computational effort and to reach high quality results. Stopping the algorithm earlier than scheduled would lead to poor performance, as we have shown in this chapter. Therefore, the original TPLS framework has poor *anytime* behavior.

In this chapter, we have addressed this weakness. We have proposed new ways to define the weights used to start new scalarizations and the order in which these weights are considered.

Our first proposal, RA-TPLS, improves strongly the anytime behavior of classical TPLS strategies and, thus, outperforms these if they are stopped before completion. However, the final quality of the Pareto front approximations obtained by the classi-

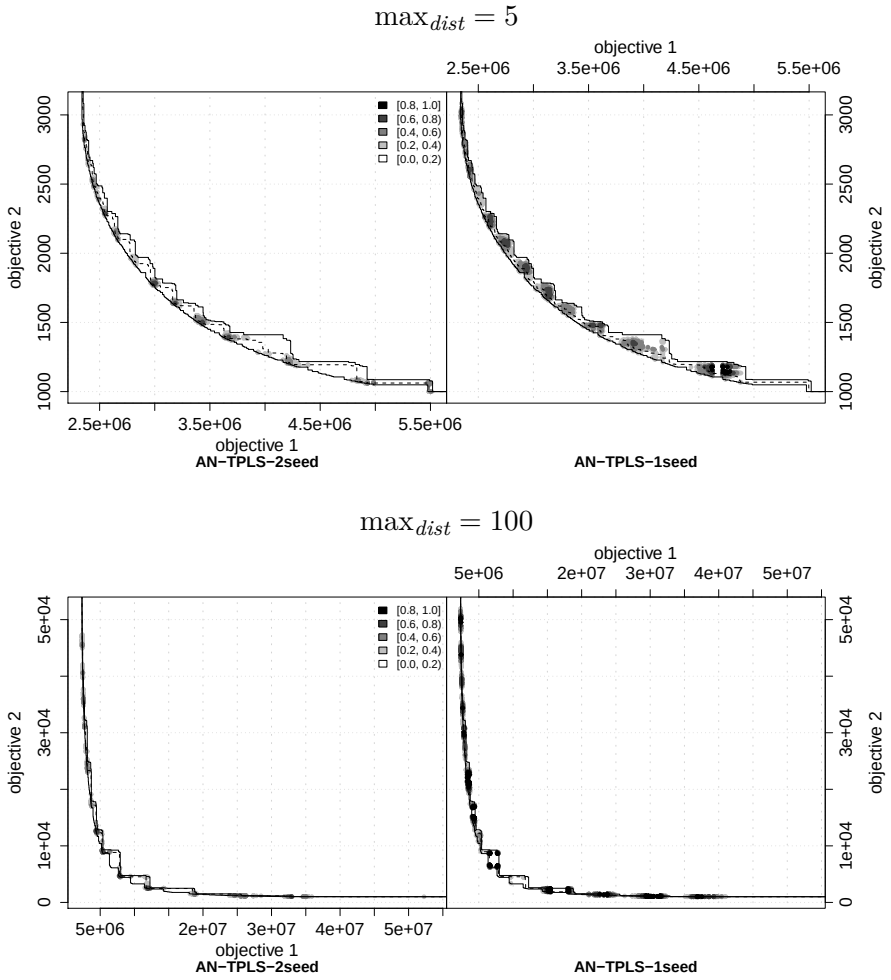

Figure 3.13: EAF differences for two anisometric TSP instances ($\max_{dist} = 5$, top, and $\max_{dist} = 100$, bottom), after completing 30 scalarizations. Strategies are AN-TPLS-2seed (left) and AN-TPLS-1seed (right). Other values of $\max_{dist}$ show similar trends. Plots for other instances are available in Dubois-Lacoste *et al.* (2010a).

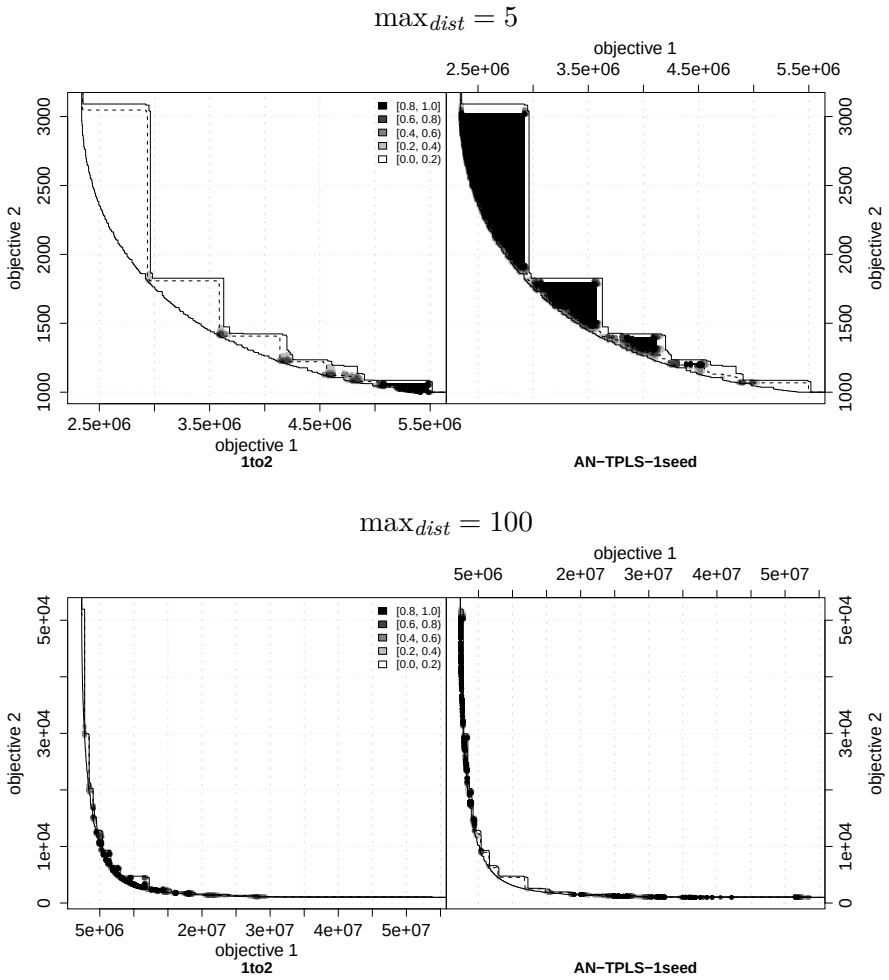

Figure 3.14: EAF differences for two anisometric TSP instances ($\max_{dist} = 5$, top, and $\max_{dist} = 100$, bottom), after completing 30 scalarizations. Strategies are *1to2* (left) and AN-TPLS-1seed (right). Plots for other instances are available in Dubois-Lacoste *et al.* (2010a).

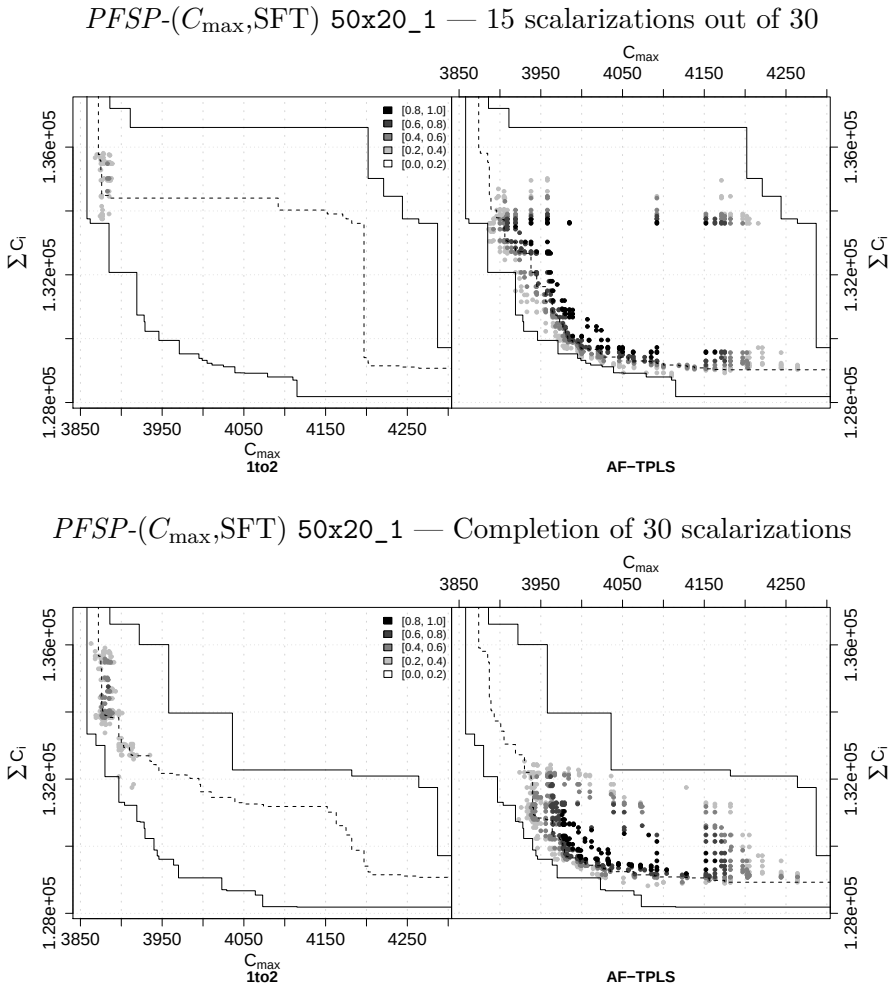

Figure 3.15: EAF differences for one bPFSP instance, after 15 scalarizations out of 30 (top plot) and after completing the 30 scalarizations (bottom plot). Strategies are *1to2* (left) and AF-TPLS (right). The instance shown is `50x20_1` and the combination of objectives is *PFSP*-($C_{\max}$,SFT). Plots for other instances are available in Dubois-Lacoste *et al.* (2010a).

cal TPLS strategies, for example D-TPLS, is better than that of RA-TPLS. Therefore, we have proposed adaptive TPLS variants that define the scalarizations adaptively in dependence of the solutions obtained in previous scalarizations.

Our adaptive TPLS variants are inspired by the dichotomic scheme proposed for exact algorithms (Aneja and Nair, 1979). Yet, exact algorithms require prohibitively high computation time for the problems considered here. We studied various variants of adaptive TPLS algorithms that differ in the number of initial solutions (one or two) that are used per weight, variants for focusing the search towards the center of a segment, and different ways of choosing the region of the objective space where to intensify the search. Our experimental results have unambiguously shown that (*i*) the adaptive TPLS variants have better anytime behavior than the non-adaptive anytime TPLS variant and (*ii*) the best adaptive TPLS variants typically also improve over the final quality of the approximations to the Pareto front reached by the best classical TPLS variants. Hence, our results suggest that the new adaptive TPLS variants should replace the classical variants in future TPLS applications.

In the next chapter, we combine this improved TPLS algorithm with Pareto Local Search, and study the performance of the resulting hybrid algorithm.

Chapter 4

Combination of Scalarization-Based and Dominance-Based Paradigms

In this chapter, we combine the scalarization-based and the dominance-based paradigms in an hybrid algorithm. We then evaluate the quality of results that can be attained by doing so. We carry out this empirical study on the flowshop scheduling problem (described previously in Section 3.1), where this type of approach has never been considered so far despite its potential. Here we use five bi-objective variants of this problem, that are different combinations of the most common objectives.

This chapter is organized as follows. In Section 4.1, we focus on algorithm development for single-objective versions of this problem that can be used to tackle scalarizations, and as a side contribution we propose new highly effective algorithms for some objectives. Next, Section 4.2 presents the design of all multi-objective aspects of the hybrid algorithm that combines the anytime version of Two-Phase Local Search (AA-TPLS, see Chapter 3) and Pareto Local Search (PLS). In Section 4.3, we present an empirical evaluation of this algorithm, that we compare with algorithms from the literature that were known to reach state-of-the-art performance. Finally, in Section 4.4 we summarize the contributions of the chapter and we highlight some possible follow-up for future research.

4.1 Single-Objective Stochastic Local Search Algorithms

The aim of the TPLS framework, and of AA-TPLS developed in Chapter 3 in particular, is to extend the efficiency of single-objective algorithms to the multi-objective context. Therefore the performance of the underlying single-objective algorithms used by AA-TPLS is crucial. In fact, they should be ideally state-of-the-art algorithms for each single-objective problem, as well as 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 Permutation Flowshop Scheduling Problem (PFSP). For a formal description of the PFSP we refer to Section 3.1. All single-objective algorithms developed are based on the iterated greedy (IG) principle (previously used in Chapter 3).

4.1.1 Stochastic Local Search Algorithm for *PFSP-*$C_{\max}$

For the *PFSP-*$C_{\max}$, we re-implemented the iterated greedy (IG) algorithm (*IG-*$C_{\max}$) by Ruiz and Stützle (2007). IG algorithms has shown to be very competitive when compared to more complex Stochastic Local Search methods, and it requires only very few parameters to be set. Algorithm 11 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 (Nawaz *et al.*, 1983) to construct the initial solution and to reconstruct a full solution from a partial one in the main IG loop. NEH sorts the jobs in descending order of their sum of processing times and inserts them following this order in the partial solution at the best position according to the objective value. When using *IG-*$C_{\max}$, this ordering is only used when NEH creates the initial solution. In the main loop of IG, the algorithm reconstructs a complete solution by reinserting previously removed jobs in random order. After reconstruction, the

Algorithm 11 Iterated Greedy

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

solution is improved by a first-improvement local search based on the *insert* neighborhood. This neighborhood is defined such that π' is a neighbor of π if π' can be obtained from π by removing a job π_i and inserting it at a different position j. The local search scans the neighborhood job by job. For a job π_i, it determines the best position where it can be inserted. If this best move improves the objective value, it is immediately applied. These steps are then repeated with the next job until a local optimum is found. A speed-up proposed by Taillard (1990) 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 the 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 (2007) 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 Stochastic Local Search Algorithm for Sum of Flowtimes Minimization

Given the very good performance of IG for makespan minimization, we decided to adapt the IG algorithm to tackle the PFSP with sum of flowtimes (*PFSP-SFT*) minimization. Although the main outline of IG (Algorithm 11) remains the same, several modifications are necessary to reach a high performance for the *PFSP-SFT*. In particular, the speed-up proposed by Taillard for exploring the insertion neighborhood is only valid for makespan minimization. Without this technique, the complexity of exploring the full insertion neighborhood becomes $O(mn^3)$, and there is no clear a priori advantage of using this neighborhood operator over pairwise exchanges of jobs. Therefore, we implement and test three neighborhood operators based on the following moves: *insertion*, which is the same as for makespan minimization but without the speed-up; *exchange*, which exchanges the positions of any pair of jobs; and *swap*, which considers only swaps of the positions of adjacent jobs. Our implementation takes advantage of the following observation: when a job in a (partial or complete) solution is moved to an earlier or later position, this move does not affect the completion times of jobs that precede the affected positions in the schedule. Therefore it is not required to recompute the completion times of unaffected jobs, which effectively halves the time of the neighborhood search. Experimental tests (Dubois-Lacoste, 2009b; Dubois-Lacoste *et al.*, 2009), which are not reported here, showed that the neighborhood operator based on swap moves leads to the best results, and therefore we used this operator to tackle the *PFSP-SFT*. More precisely, our local search sequentially examines all possible swap moves, and if it finds an improvement, it performs the move (first improvement) and continues the evaluation of the remaining moves. Then, if the objective value has been improved, a new sequential evaluation can be performed from the current solution in order to reach a local optimum.

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

For the initial solution, we use the same NEH algorithm as for *PFSP-*$C_{\max}$ since it was shown to provide good quality solutions for *PFSP-SFT* as well (Woo and Yim, 1998).

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 (Dubois-Lacoste *et al.*, 2009) that good results are produced by using the formula:

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

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

4.1.3 Stochastic Local Search Algorithm for Total and Weighted Tardiness Minimization

We also adapted IG to tackle the total tardiness (*PFSP-TT*) and the weighted tardiness (*PFSP-WT*) criteria. 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 (Vallada *et al.*, 2008) to provide acceptable solutions. We extended SLACK to take into account the job weights, and we call this variant the WSLACK heuristic. Our evaluation of WSLACK and other heuristics, reported in Dubois-Lacoste (2009b), has shown that using WSLACK to provide the initial order for the NEH algorithm produces the best results. A further modification of the NEH algorithm is necessary for the case when there are several positions for inserting a job that minimizes the objective value of a (possibly partial) solution. This situation specially arises when the solution is partial and each job can be processed before its due date, and, hence, several insertion positions result in a tardiness value of 0. Our implementation of the NEH algorithm inserts jobs in the earliest position from such a set of equally good positions.

Because of the due dates assigned to the jobs, objective values can differ very much between instances, and we could not find an effective setting based on the input data for the formula of the temperature (Eq. 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 depends on the relative difference between the objective value of the current and the new solution and a parameter T_c:

$$p = \exp\left(-100 \cdot ((f(\pi') - f(\pi))/f(\pi))/T_c\right). \tag{4.3}$$

4.1.4 Stochastic Local Search Algorithms for the Scalarized Problems - Combinations of Two Objectives

Given the very good performance of our IG algorithms to minimize each objective (Dubois-Lacoste, 2009b; Dubois-Lacoste *et al.*, 2009), we also use IG to solve each scalarized problem. To define the acceptance probability in the procedure AcceptanceCriterion (Algorithm 11, line 6), we use the same formula as for the tardiness objective (Eq. 4.3). Another important adaptation of IG for solving the scalarized problems resulting from combinations of objectives 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 ade-

quate 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 of Single-Objective Algorithms

We fine-tuned the parameters d, T_c and N_{LS} of each IG variant for *PFSP-SFT*, *PFSP-TT* and *PFSP-WT*, and for the five scalarized problems. The range of possible values for d was $[2, 12]$, for T_c it was $(0, 100]$ (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 minimization problem (*PFSP-*C_{max}), because Ruiz and Stützle (2007) have already proposed good parameter settings (see Section 4.1.1).

We used `irace` (presented in appendix to this thesis) for the automatic tuning. (In particular we use an implementation that is a side contribution of this thesis, and that is presented in the appendix). For this purpose, we generated 100 new instances for each number of jobs in {`20,50,100,200`}, following the procedure described by Minella *et al.* (2008). These instances all have `20` machines, since they are the hardest considered in this test set. All code is implemented in `C++` and compiled with gcc version 3.4.6 using the `-O3` flag. Experiments presented in this chapter 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. 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.* (2008) divided by 30. (In fact, for the time limits to be used by the final multi-objective algorithm we heuristically adopted those used by Minella et al., since they have run the experiments on a very similar hardware as ours.) The choice of the time limits is also motivated by the assumption that the AA-TPLS phase of the final algorithm will be allocated half of the total time and that we use 15 scalarizations overall.

Table 4.1: IG parameter settings. The settings for *PFSP*-$C_{\max}$ are taken from Ruiz and Stützle (2007), in particular the neighborhood they use is based on best insertion moves and is stopped when reaching a local optimum. The other settings were found by means of automatic tuning (Section 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

To assess the importance of the parameter tuning on the solution quality, we compared two versions of our algorithm. In the first one, we consider parameter settings that are specific to each number of jobs, which is done by executing multiple tuning runs one for each number of jobs. In the second one, we use the same parameter settings for all instance sizes of a problem, which is done by executing the tuning runs using a mix of instances with different sizes. The size-independent parameter settings produce only slightly worse results (Dubois-Lacoste *et al.*, 2010b) and they are arguably more robust when applied to instances of an intermediate size, which is not considered in the tuning. Table 4.1 describes the parameter settings of IG we used to produce all results given later in this chapter. We provide as supplementary material (Dubois-Lacoste *et al.*, 2010b) results obtained using size-specific parameters.

4.2 Multi-objective Algorithms Design

In this section, we turn our attention to the bi-objective problems in terms of Pareto optimality, combining the two multi-objective frameworks, AA-TPLS and PLS. The results

of this analysis lead us to propose a hybrid multi-objective algorithm that combines both of them together.

4.2.1 Analysis of Pareto Local Search Components

In the following paragraphs, we study the two main aspects of the original PLS algorithm (see Section 2.7.5) that we use in this chapter. There are the initial set of solutions given as input to PLS (the seed of PLS); and the neighborhood operator used for generating new solutions.

Seeding

We analyze the computation time required by PLS and the final quality of its output when seeding PLS with solutions of different quality. We test seeding PLS with (*i*) one randomly generated solution, (*ii*) two solutions, one for each single objective, obtained by the appropriate version of the NEH heuristic for each objective (Section 4.1), and (*iii*) two solutions obtained by IG (Section 4.1) for each objective after $10\,000$ iterations. The neighborhood used for PLS is a combination of exchange and insertion (for details, see the next paragraph on the neighborhood operator).

Figure 4.1 gives representative examples of non-dominated sets obtained by PLS for each kind of seed along with the initial seed solutions of NEH and IG. Generally, seeding PLS with very good initial solutions, as obtained by IG runs, produces better non-dominated sets in terms of a wider range of the Pareto front and better quality. This result is strongest for the *PFSP*-($C_{\max}$,SFT). We use the empirical attainment function (EAF, see Section 2.6.2) to assess graphically the difference of the resulting non-dominated sets. As shown on the supplementary material page (Dubois-Lacoste *et al.*, 2010b), the differences between the EAFs obtained for each kind of seed across 10 runs confirm this result for the instance of Figure 4.1 and also other instances. The computation time required by PLS in dependence of the initial seed is given in Table 4.2. The results show that seeding PLS with solutions of higher quality does not strongly affect the computation time required by PLS. From these results, we also expect that seeding

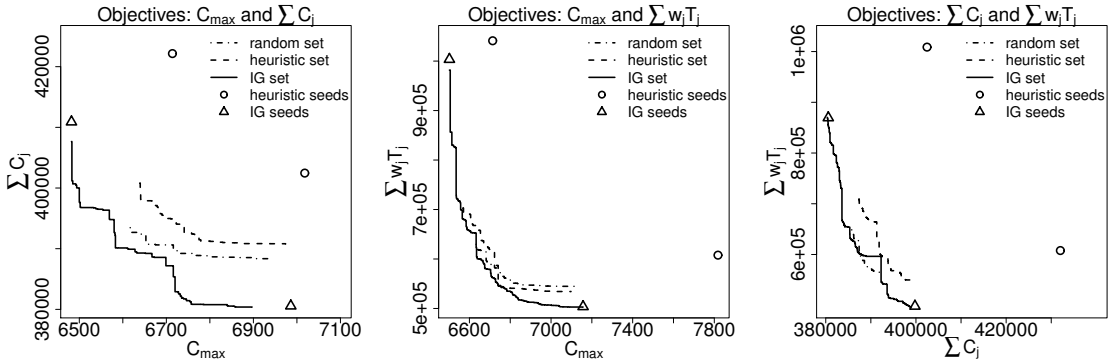

Figure 4.1: Each plot shows one non-dominated set obtained by PLS using different quality of seeds for instance `100x20_3`. The randomly generated solutions are outside the shown range.

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, 839.0]	[831.7, 958.8]

PLS with solutions obtained by AA-TPLS will further enhance the quality of the results without an excessive computation time overhead.

Neighborhood operator

Starting from two solutions obtained by IG (Section 4.1) for each objective after 10 000 iterations, we test variants of PLS based on three different neighborhoods: (*i*) insertion, (*ii*) exchange, and (*iii*) the combination of exchange and insertion. The latter one sim-

Table 4.3: Confidence intervals (with level=0.95) of the computation time (seconds) of PLS using different neighborhood 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]

ply checks for all moves in the exchange and insertion neighborhoods. We measure the computation time of PLS with each neighborhood operator for different combinations of objectives in Table 4.3. The computation time of the combined exchange and insertion neighborhood is slightly more than the sum of the computation times for the exchange and the insertion neighborhoods separately. For comparing the quality of the results, we examine the EAF differences of 10 independent runs. Figure 4.2 gives two representative examples. Typically, the exchange and insertion neighborhoods produce better results in different regions of the Pareto front (top plot), and obviously both of them are consistently outperformed by the combined exchange and insertion neighborhood (bottom plot). Given the complementarity of exchange and insertion to perform well in different regions, we decided to use the combined neighborhood in our hybrid approach.

Continuous Improvement to Use All Available Computation Time

The original PLS stops when no unexplored non-dominated solutions remain in the archive. When using a limit for the computation time, for instance to compare with other algorithms, PLS may naturally finish before the available time is consumed, and, in this case, the remaining time would be wasted. We therefore modify PLS to continue exploring the search space up to the time limit by extending the neighborhood to those

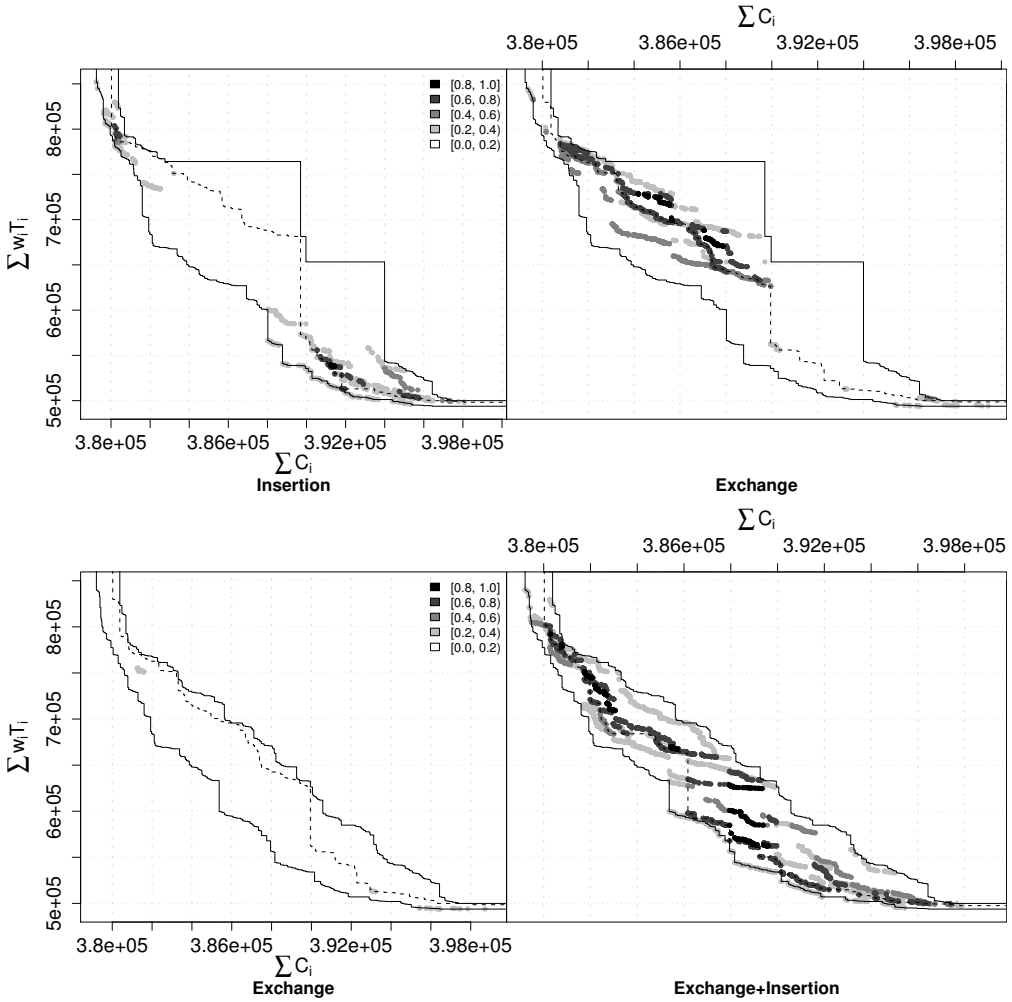

Figure 4.2: EAF differences for (*top*) insertion vs. exchange and (*bottom*) exchange vs. exchange and insertion, for *PFSP*-(*SFT*, *WT*). Dashed lines are the median attainment surfaces of each algorithm. Black lines correspond to the overall best and overall worst attainment surfaces of both algorithms.

solutions that can be reached by applying to each non-dominated solution the exchange or the insert neighborhood operator twice.

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

4.2.2 Analysis of Adaptive Anytime Two-Phase Local Search components

In this section, we examine several components of the Adaptive Anytime Two-Phase Local Search framework (AA-TPLS). In Chapter 3, we presented an in-depth study of the most important one: the weight setting strategy, which defines the sequence of weights used by consecutive scalarizations. Here we use the same algorithm, and we examine an aspect that is problem-dependent: whether AA-TPLS performs better than a *restart* strategy where the initial solution for each scalarization is generated from a new seed, independently of previously found solutions. Finally, we discuss appropriate settings for the number of scalarizations.

Chaining 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 chapter, we have assumed that AA-TPLS is superior to independent restarts for the bi-objective PFSPs problems tackled here. To confirm this hypothesis, we performed experiments comparing both strategies.

We implemented a *Restart* strategy derived from AA-TPLS. In this *Restart* strategy, the initial solution of each scalarization is generated by variants of the NEH heuristic

for the scalarized problems.[1] The *Restart* strategy solves only one scalarization for each pair of solutions (since it does not involve the two different seeds), however, in our experiments it still executes the same number of scalarizations as AA-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 AA-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 (Dubois-Lacoste *et al.*, 2010b). Therefore, we conclude that the AA-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-like strategies 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 AA-TPLS algorithm with the number of scalarizations $N_{\text{scalar}} \in \{10, 20, 40, 80\}$. We used the hypervolume indicator (see Section 2.6.1) to compare the

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

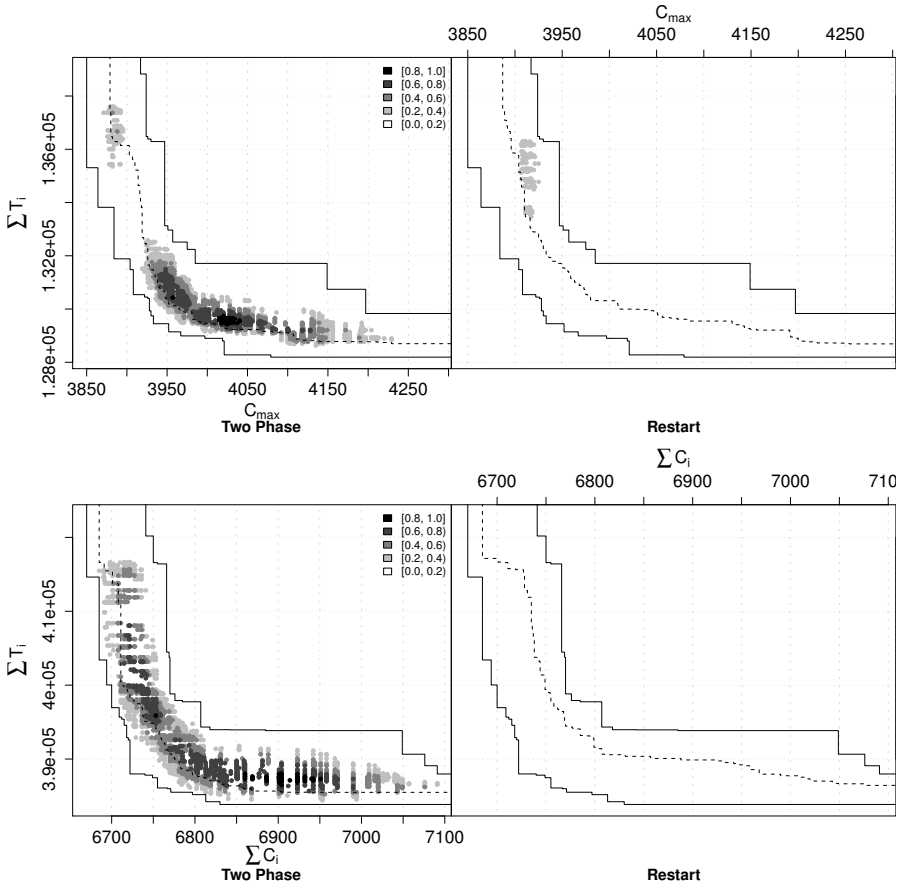

Figure 4.3: AA-TPLS on the left and *Restart* on the right. Results are shown for objectives $C_{\max}$ and $\sum C_i$, and instances `50x20_1` (top) and `100x20_1` (bottom).

quality obtained for the four values for N_{scalar}; the objective values are normalized to the range $[1, 2)$ such that 2 corresponds to the worst value of the corresponding objective plus one. Then we computed the hypervolume of these normalized non-dominated sets, using $(2, 2)$ as the reference point. We used five instances of size `50x20_1` and `100x20_1` and 25 independent runs of AA-TPLS per instance. We performed an analysis of variance (ANOVA) in order to determine if there are significant differences. The difference in the results quality appeared to be rather small, showing that AA-TPLS is rather robust to the change of this parameter. However, significant differences are never in disfavor of 10 and 20 scalarizations, and therefore we focus on a rather small number of scalarizations in our final algorithm.

4.2.3 Adaptive Anytime TPLS + Component-Wise Step, Adaptive Anytime TPLS + PLS

As a final step of our algorithm engineering process, we compare the performance trade-offs incurred by post-processing AA-TPLS results by either PLS or the component-wise step (CW-step, a restricted version of PLS, see Section 2.7.7), both using the combination of the insertion and exchange neighborhood. For all instances, we generated 10 initial sets of solutions by running AA-TPLS for 30 scalarizations each of 1000 iterations of IG and, in order to reduce variance, we apply CW-step and PLS once to each of these sets.

Table 4.4 gives the computation time that is incurred by PLS and the CW-step after AA-TPLS has finished. The CW-step incurs only a very minor overhead with respect to AA-TPLS, while PLS requires considerably longer times, especially on instances with 100 jobs. However, the times required for PLS to finish are much lower than when seeding it with only two very good solutions (compare with Table 4.2 on p. 100). With respect to solution quality, Figure 4.4 compares AA-TPLS versus AA-TPLS +CW-step (top), and AA-TPLS +CW-step versus TP+PLS (bottom). As expected, the CW-step is able to slightly improve the results of AA-TPLS, while PLS produces much better results. In summary, if the computation time is very limited, the CW-step provides significantly better results at almost no computational cost; if enough time is available, a full execution of PLS gives a further substantial improvement. These conclusions lead

Table 4.4: Confidence intervals (with level=0.95) of the computation time (seconds) for CW-step and PLS seeding with the output of AA-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]

us to propose a hybrid TP+PLS algorithm, where a time-bounded PLS is applied to the solutions obtained by AA-TPLS.

Hybrid TP+PLS Algorithm

We design a final hybrid algorithm that uses AA-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 use AA-TPLS to perform a series of scalarizations and to produce a set of high-quality, non-dominated solutions. This set is then further improved by a time-bounded PLS that uses a combined insertion plus exchange neighborhood operator. The result is a hybrid TP+PLS algorithm for each of the five bi-objective PFSPs. The parameters of TP+PLS are the time given to the initial IG algorithms for each single objective, the number of scalarizations of AA-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.

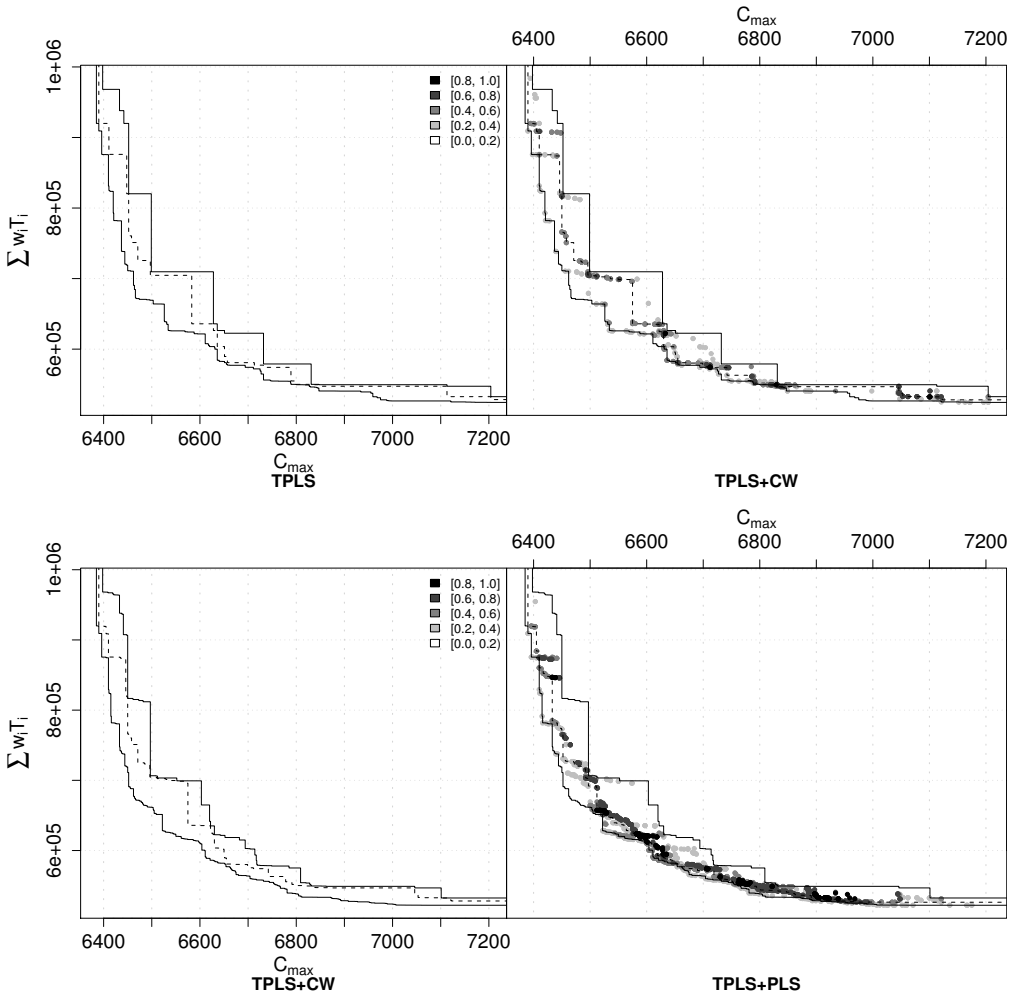

Figure 4.4: EAF differences between (top) simple AA-TPLS vs. AA-TPLS + CW-step, and (bottom) AA-TPLS + CW-step vs. AA-TPLS + PLS. Objectives are $C_{\max}$ and $\sum w_i T_i$.

4.3 Performance Evaluation of the Hybrid TP+PLS Algorithm

In this section, we compare the hybrid TP+PLS algorithm previously designed with algorithms that are known to be state-of-the-art for the bi-objective PFSPs.

Table 4.5: Average number of iterations performed in 10 seconds by IG algorithms, for each size instance size. These numbers are similar when considering the weighted tardiness.

Size	$C_{\max}$	SFT	TT	($C_{\max}$,SFT)	($C_{\max}$,TT)	(SFT, TT)
20x5	68000	78000	60000	70500	65000	66000
20x10	30200	48000	42300	42400	41500	45000
20x20	14800	29000	26000	25000	24600	28000
50x5	15800	18600	14300	16000	14600	15200
50x10	4500	11100	9600	9400	9000	9900
50x20	2100	5900	5400	5000	4800	5500
100x5	4800	6200	4750	5100	4400	4700
100x10	1330	3350	2900	2700	2500	2900
100x20	500	1650	1550	1350	1300	1550
200x10	460	900	800	730	680	790
200x20	120	450	420	360	350	420

4.3.1 Experimental Setup

For the experimental analysis of TP+PLS, we use the same benchmark instances as Minella *et al.* (2008). 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 (1993) and augmented with due dates by Minella et al. Recall that these instances are different from the ones we used for the tuning of IG and the design of the TP+PLS algorithm. In other words, we have a clear separation between training instances and test instances.

Table 4.5 shows, for a given CPU time, how many iterations can be performed by our implementation, on the hardware environment we used (see Section 4.2.1). 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.* (2008). 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 assigned to $IG_{\{1,2\}}$ being 1.5 times the time assigned to IG_Λ. Once all scalarizations are finished, the remaining time is spent on PLS.

Table 4.6 gives the value of these parameters for each instance size. We set these values based on the following considerations. First, we focus on the time settings for instances with 20 machines, and obtain the time settings for instances with 5 and 10 machines by dividing it by 4 and 2, respectively. We assign 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.

Note that the tuning of the IG algorithms in Section 4.1.5 was done using slightly different computation time limits for each of the IG runs. In fact, we did not repeat the tuning of IG for these slightly different computation time limits, since we anyway hope the final IG algorithm to be relatively robust with respect to the parameter settings so that a re-tuning would not result in very strong gains in solution quality. The very high performance of the hybrid algorithm, as shown in the following, confirms this assumption.

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.* (2008). These reference sets correspond to the non-dominated points from all outcomes of 10 independent runs of 23 heuristics and metaheuristics, including algorithms for specific PFSP variants or adaptations of algorithms originally proposed for other problems. Each of those runs was stopped after the same time limit as our TP+PLS. These reference sets were obtained on an Intel Dual Core E6600 CPU running at 2.4Ghz, which is similar in speed to the CPU 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 TP+PLS typically matches and is often better than the reference set. That is, in at least half of the runs, TP+PLS ob-

Table 4.6: 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 (it does not include $IG_{\{1,2\}}$). PLS is run until the overall computation time is reached. Computation times are given in seconds. N_{scalar} does not include the runs of IG for the two initial solutions.

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	3.0	12	50
50x20	9.0	6.0	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

tains 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 to State-of-the-art Algorithms

Given the good quality of TP+PLS suggested by the comparison with reference sets, we next compare the results of TP+PLS with Multi-objective Simulated Annealing (Varadharajan and Rajendran, 2005) and Multi-objective Genetic Local Search (Arroyo and Armentano, 2005), two algorithms that have recently been shown to be state of the art for the bi-objective PFSPs (Minella *et al.*, 2008). To make this comparison more fair and account for possible differences in implementations and computing environment, we

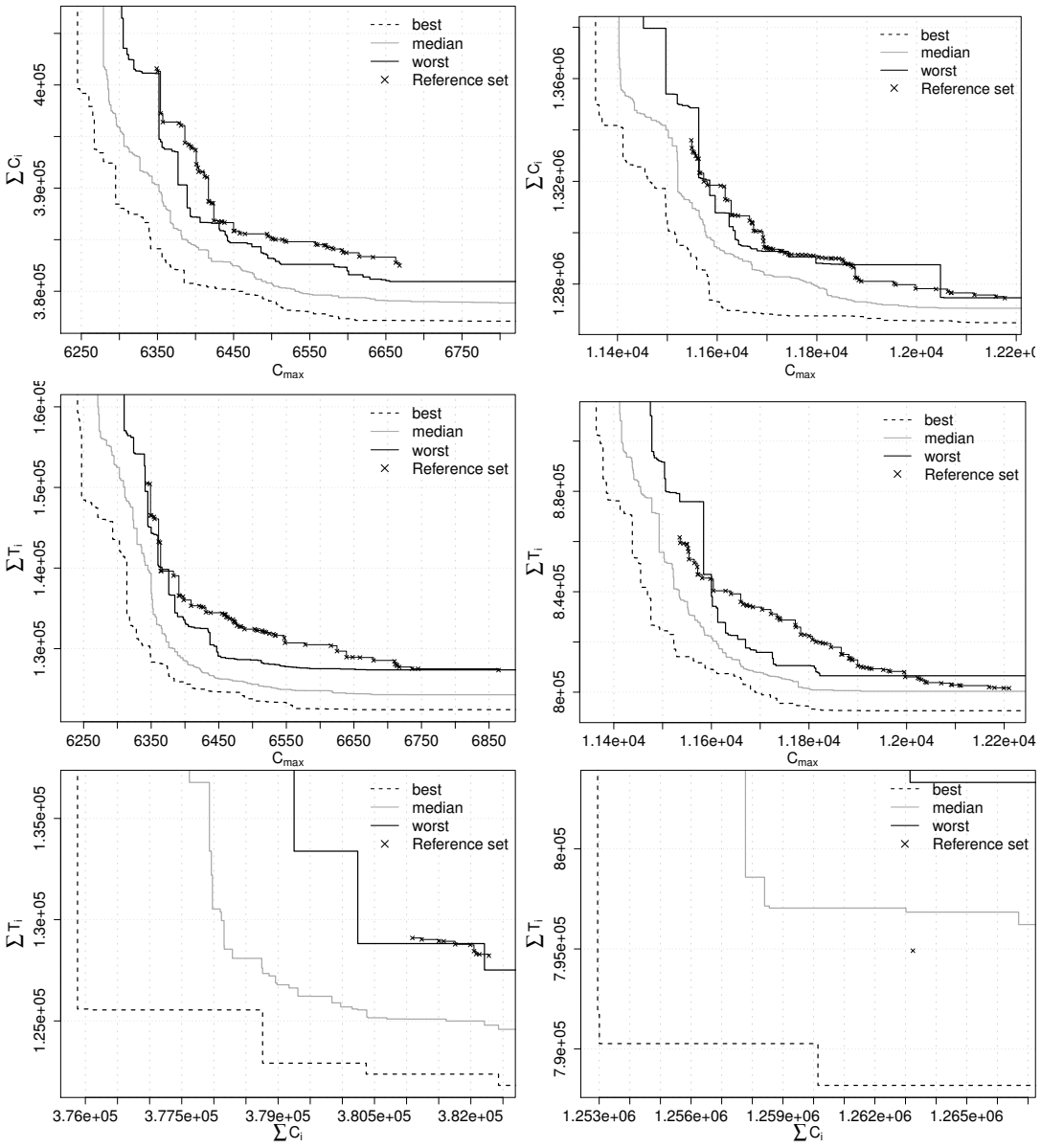

Figure 4.5: Attainment surfaces of TP+PLS against the reference set for instances `DD_Ta082` (100x20) on the left and `DD_Ta102` (200x20) on the right, for top: *PFSP*-($C_{\max}$,SFT), middle: *PFSP*-($C_{\max}$,TT) and bottom: *PFSP*-(*SFT*, *TT*).

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

Multi-objective Simulated Annealing

Varadharajan and Rajendran (2005) designed Multi-objective Simulated Annealing (MOSA) for the bi-objective *PFSP*-($C_{\max}$,SFT) (minimisation of the makespan and sum of flowtimes). Recently, Minella *et al.* (2008) 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 (2005) proposed two combinations of parameters for MOSA, one for shorter and another for longer runs. We use here the parameters for longer runs, with a higher value for the epoch length and a lower temperature threshold.

The core of MOSA is a standard classical simulated annealing algorithm, henceforth denoted 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 one objective over the other is kept constant until the temperature reaches a certain value. Then single-SA restarts by setting the temperature again to its initial value, but the probability to choose an objective over the other one is slightly changed.

From a high-level point of view, MOSA consists of two main phases. The first phase starts from a solution provided by the NEH heuristic to minimize the makespan, which is subsequently improved by three improvement schemes (named JIBIS, OSSBIS and JIBSS) that evaluate a sequence of *insertion* or *exchange* moves by considering the job following either their index or their position, and apply the improving moves. Then, single-SA is run four times with different probabilities to consider the makespan instead of the sum of flowtimes when a new solution has to be considered. These probabilities for the four runs are $(1, 0.83, 0.66, 0.5)$. Each run of single-SA starts from the previous solution found and stops when the temperature threshold is reached. The second phase of MOSA starts from a solution provided by Rajendran's heuristic (Rajendran, 1993), 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 flow-

times 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 96.

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 computation time limit. For this purpose, we have considered two alternatives. The first alternative uses a modified cooling rate of the temperature that *approximately* reaches the temperature threshold when the computation time reaches the limit. The second alternative keeps the original cooling rate, and sets again the temperature to its initial value when it reaches the threshold (but keeping the current solution). This latter possibility allows to run the algorithm for a precise computation time, which is exactly divided among the eight runs of single-SA. We carried out some preliminary experiments to compare the quality of the outputs provided by each variant of MOSA. The quality of the non-dominated sets was roughly equivalent, and we decided to use the second variant for our comparison. The other parameter settings of MOSA are taken directly from the original publication.

Multi-objective Genetic Local Search

Multi-objective Genetic Local Search (MOGLS), proposed by Arroyo and Armentano (2005), was the second-best algorithm for the bi-objective PFSPs studied in the review of Minella *et al.* (2008). 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 (Arroyo and Armentano, 2004) 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 (Glover, 1998) generates the remaining solutions. The original MOGLS—and, as far as we know, the implementation of Minella *et al.* (2008)— uses the version of non-dominated sorting proposed by Deb *et al.* (2002) 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 (2003). After a given number of generations, a multi-objective local search is performed on a subset of the current popu-

lation 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 (Morse, 1980). 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 remaining parameters of MOGLS are set to the same values as in the original publication.

Comparison of TP+PLS with Multi-objective Simulated Annealing and Multi-objective Genetic Local Search

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.* (2008). 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 in Dubois-Lacoste *et al.* (2010b)). Since the differences in implementation language and computation environment with respect to Minella *et al.* (2008) 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 4.7 the percentage of runs (computed for each instance over the 625 pairwise comparisons of the 25 runs, and averaged over the 10 instances of each size) that the output set of our TP+PLS algorithm is better in the Pareto sense (in the sense of

"$\lhd$", see Def. 8, page 24 of Section 2.5.2) than the output set obtained by a run of MOSA, and, conversely, the average percentage of runs that the output set of MOSA is better than TP+PLS. The same comparison is done in Table 4.8 between TP+PLS and MOGLS. Detailed tables with percentage values for each instance are available as supplementary material (Dubois-Lacoste *et al.*, 2010b). 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 4.7 show that for small instances of 20 jobs, MOSA and our TP+PLS algorithm are difficult to compare. The low percentages are explained by the fact that both algorithms often find the same non-dominated set, which is probably the optimal Pareto front. For these small instances, differences are not consistent across instances and combinations of objectives, and it cannot be said that any algorithm is clearly better than the other. Nevertheless, for all the remaining instances, Tables 4.7 and 4.8 show the excellent results of our TP+PLS algorithm, with very high percentages in its favor, whereas the percentages in favor of MOSA and MOGLS are negligible.

Beyond the fact that TP+PLS often dominates MOSA and MOGLS (Tables 4.7 and 4.8), one may wonder how important is the difference between the sets. To answer this question, we also examine the EAF differences between the algorithms (Section 2.6.2). Plots in Figures 4.6, 4.7 and 4.8 show some examples of these differences for three different instances. These plots reveal strong differences and a large gap along the whole Pareto frontier between the region typically attained by MOSA and MOGLS and the region typically attained by TP+PLS. Hence, we can conclude that the difference between the non-dominated sets is not only very often in favor of our algorithm, but that these differences are also very strong.

All the additional plots and detailed tables (including the ones with the version of our hybrid algorithm using size-specific parameters), together with new reference sets obtained from our results are available as supplementary material (Dubois-Lacoste *et al.*, 2010b).

Given such clear results, the usage of unary or binary performance indicators, which assess the quality of non-dominated sets that are not comparable in the Pareto sense, is

superfluous. Our conclusion from this assessment is that TP+PLS is the new state-of-art for the bi-objective permutation flow-shop scheduling problem, for all combinations of objectives we studied.

It should be noted that at the same time that our results were published (Dubois-Lacoste *et al.*, 2011b), another algorithm for the bi-objective flowshop was published that also shows high-quality results (Minella *et al.*, 2011). The results shown by the algorithms in these two articles look roughly of the same quality. A detailed comparison between them would be interesting in the future.

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

In this chapter, we have detailed the steps followed in the engineering process of a multi-objective Stochastic Local Search (SLS) algorithm for five bi-objective permutation flow-shop problems that combines the scalarization-based and the dominance-based paradigms and, in particular, AA-TPLS and PLS.

We have followed a bottom-up SLS algorithm engineering process that first engineered effective SLS algorithms for each of the single objective problems that underlie the bi-objective ones, and for the weighted sum scalarization of pairs of objectives. In fact, high performing SLS algorithms for these single objective problems are of crucial importance for the final performance of the AA-TPLS algorithm. In a second step, we examined the main aspects of the multi-objective part itself. In the study of the PLS algorithm components, we could show that PLS strongly profits from seeding it with good initial solutions. Concerning the neighborhood to be used in PLS, we found that a combination of the exchange and the insert neighborhoods was beneficial. In a final step, we examined the usefulness of combining AA-TPLS with either a limited component-wise step (Paquete and Stützle, 2003, 2009b) or a time-bounded run of PLS, the latter resulting in clearly superior solution quality.

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

Our results, previous experimental analysis (Paquete and Stützle, 2009b) and other similar success stories recently reported in the literature (Lust and Teghem, 2010b) indicate the large potential of hybrid algorithms combining the TPLS (and in particular the new variant AA-TPLS) and PLS frameworks. Moreover, the general engineering methodology that we followed in this chapter is applicable to other combinatorial bi-objective problems.

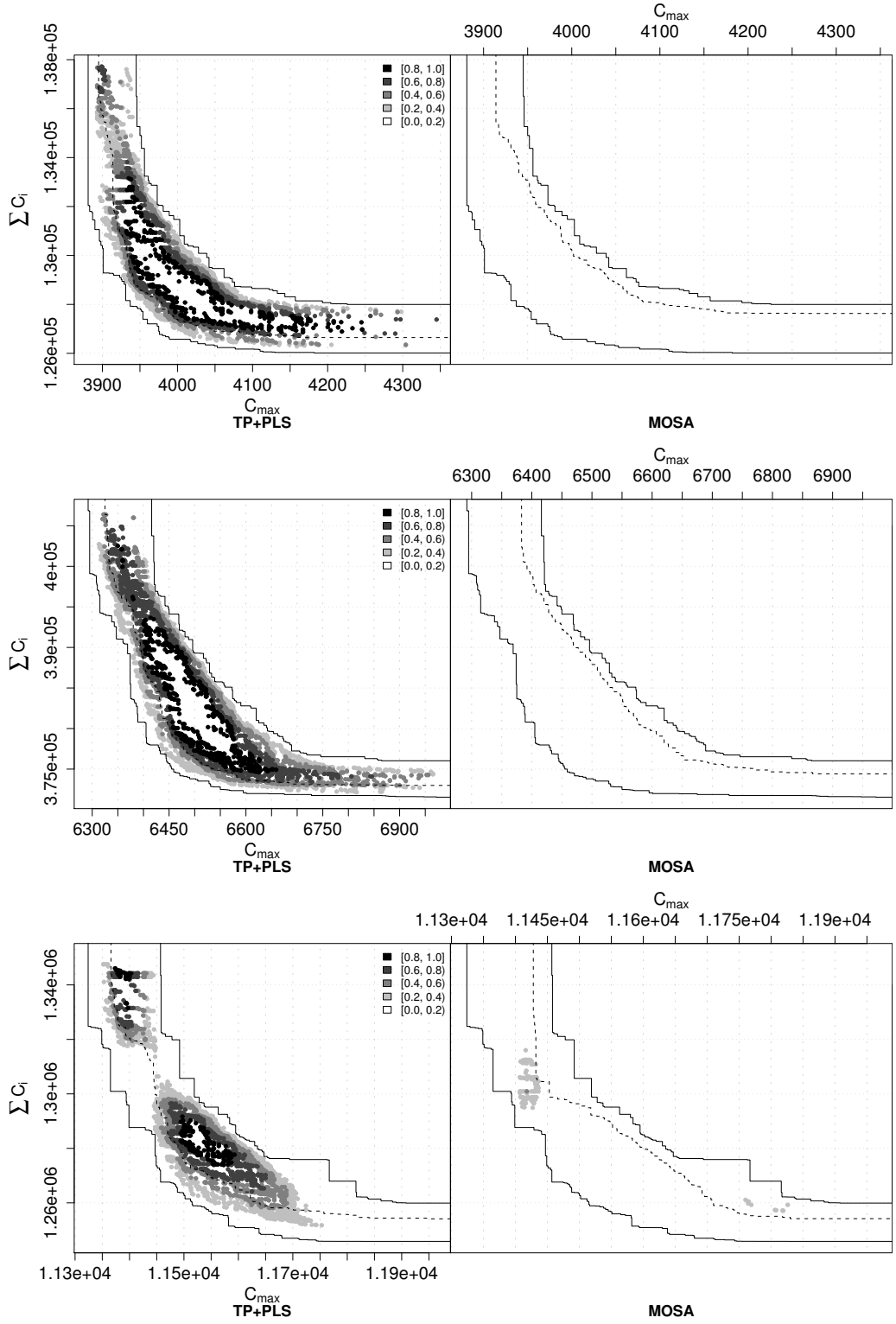

Figure 4.6: EAF difference for *PFSP*-($C_{\max}$,SFT) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

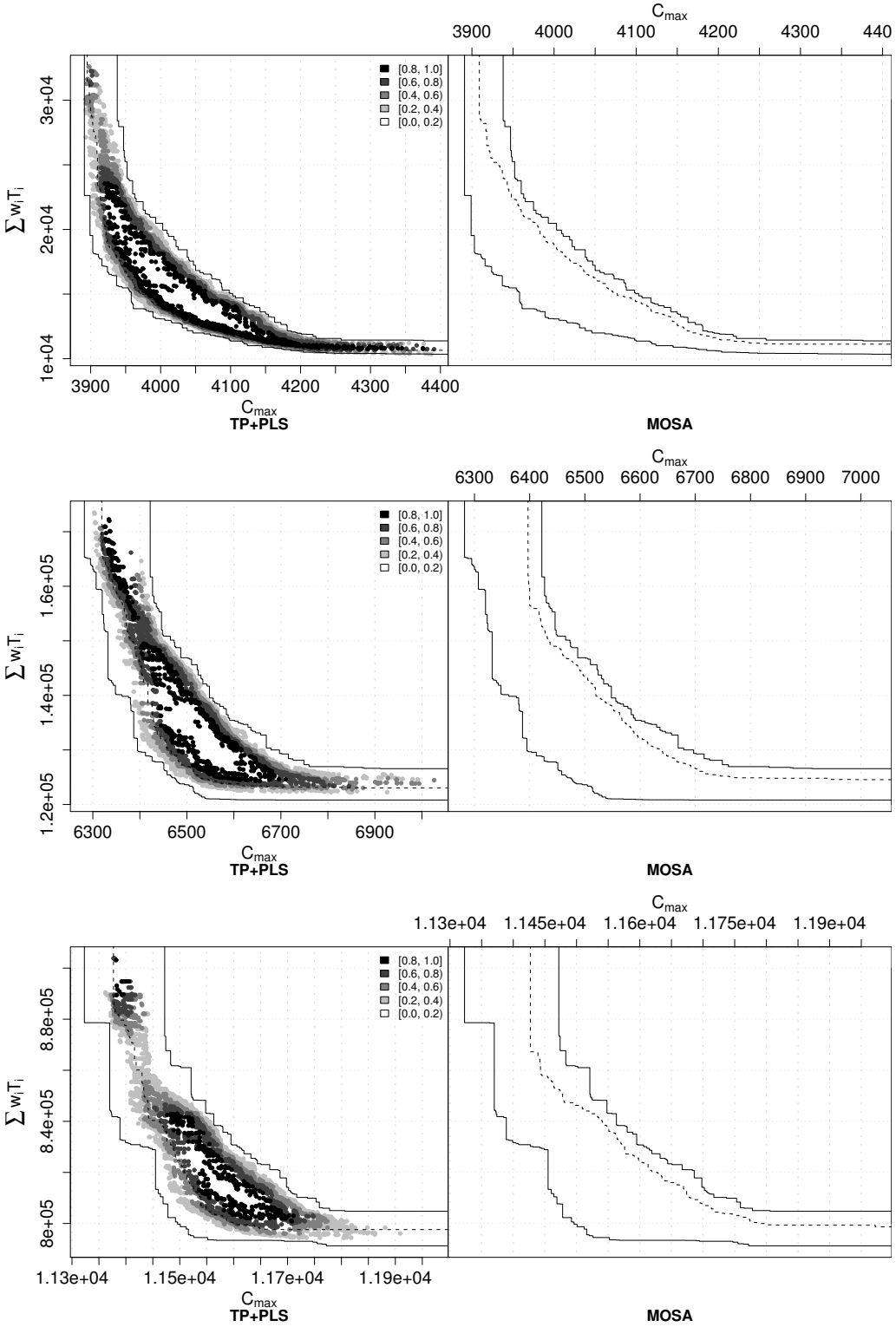

Figure 4.7: EAF difference for *PFSP*-($C_{\max}$,TT) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

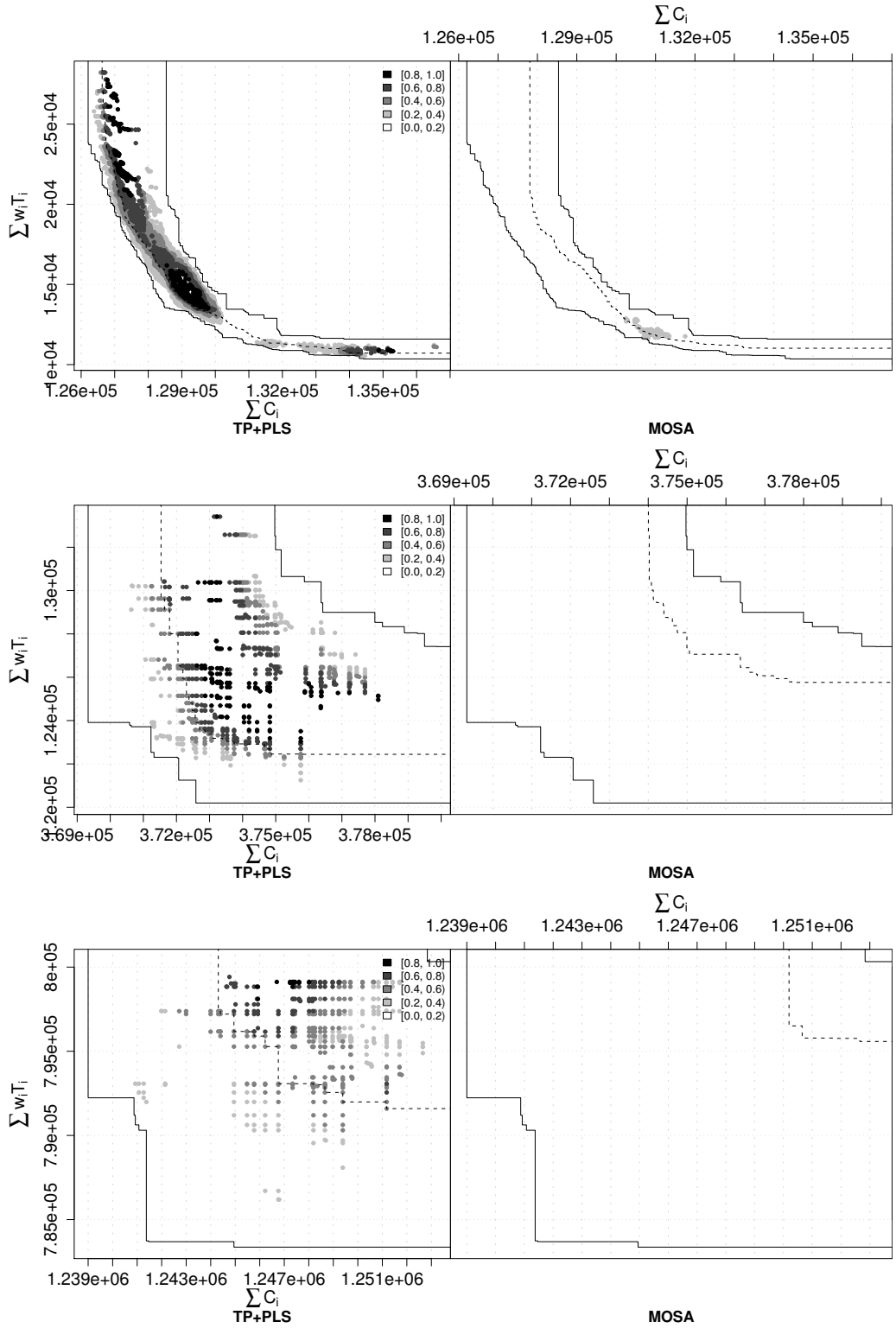

Figure 4.8: EAF difference for *PFSP*-(*SFT*, *TT*) on instances (from top to bottom) `DD_Ta051 (50x20)`, `DD_Ta081 (100x20)`, `DD_Ta101 (200x20)`.

Chapter 5

Anytime Pareto Local Search

In the previous chapter, we made use of the Pareto Local Search (PLS) algorithm and have shown that its usage in hybrid algorithms can lead to state-of-the-art results. Typically, the number of solutions in the non-dominated sets that are found for the flowshop problem is limited, and PLS can be used in its original version as we did. For problems with many more solutions, however, PLS might progress at a slower pace, requiring long computation times before finding high-quality solutions.

Our goal in this chapter is to carefully engineer further improvements over the original PLS algorithm that exhibit a better anytime behavior (see Section 2.9), and, thus, are more adapted to tackling problems where non-dominated sets typically have many solutions.

We first broaden in Section 5.1 our perspective on the algorithmic aspects of PLS by decomposing it into different, clearly defined algorithmic components. Then, the next two sections detail different attempts to improve the anytime behavior of PLS and report the experimental results. Section 5.2 focuses on the variants of the algorithmic components and Section 5.3 on the objective space discretization. In Section 5.4, we compare the best alternatives obtained from the two different approaches, and we compare them to the original PLS algorithm. Finally, we give some concluding remarks in Section 5.5 and highlight promising directions for future research in this field.

5.1 A Generalized View of Pareto Local Search

In this section, we take a step back to look at the algorithmic outline of PLS, with the goal of decomposing PLS into different, clearly defined, algorithmic components. This more abstract view will cover in a common outline both the original PLS and subsequent variants that we propose in this chapter.

We give in Algorithm 12 a more general view of the PLS framework of PLS. It is initialized by an initial set A of mutually non-dominated solutions, called *archive*. These solutions are initially marked as unexplored (line 2). PLS then iteratively applies the following steps. First, a solution s is selected among all unexplored ones (*selection step*, line 5). Then, some (or all) of the neighbors of s, are explored (*neighborhood exploration*) and all the neighbors that are accepted (*acceptance criterion*) w.r.t. the archive A are added to A (lines 8 to 11). Solutions in A that are dominated by the newly added solutions are removed (procedure Update in line 10). Once the termination criterion for the exploration of the neighborhood of s is met, s is marked as explored (line 13). When all solutions have been explored, and no more new non-dominated solutions can be discovered, the algorithm stops in a Pareto local optimum (Paquete *et al.*, 2007). Algorithm 12 is a generic outline and different variants of PLS can be obtained by different instantiations of the components SelectSolution, NeighborhoodExploration and AcceptSolution

Selection step. This component determines how to select the next solution for neighborhood exploration. In the original PLS, this solution is selected uniformly at random among the unexplored ones.

Neighborhood exploration. This component performs the neighborhood exploration of the selected solution. In particular, it defines the part of the neighborhood that will be explored before switching to a different solution. The original PLS algorithm always explores the entire neighborhood of a solution; this choice corresponds to the *best-improvement* rule in single-objective local search algorithms.

Acceptance criterion. This component determines the conditions for new solutions to enter the archive. The original PLS accepts all solutions identified in the neighborhood exploration that are non-dominated.

Algorithm 12 General Framework for Pareto Local Search

```
1: Input: An initial set of non-dominated solutions A
2: explored(s) := FALSE   ∀s ∈ A
3: A_0 := A
4: repeat
5:    s := SelectSolution(A_0)
6:    repeat
7:       s' := NeighborhoodExploration(s)
8:       if AcceptSolution(A, s') then
9:          explored(s') := FALSE
10:         A := Update(A, s')
11:      end if
12:   until (termination criterion)
13:   explored(s) := TRUE
14:   RemoveDominated(A)
15:   A_0 := {s ∈ A | explored(s) = FALSE}
16: until A_0 = ∅
17: Output: A
```

PLS requires a possibly very long time to complete, and even a long time to reach good approximations to the Pareto front. If stopped too early, PLS can deliver poor-quality results, and, thus, would be of little help when a good anytime behavior is desirable.

In the next section, we explore some alternative designs of PLS components and we evaluate empirically their impact on the performance from an anytime behavior perspective.

5.2 Alternative Algorithmic Strategies

The use of alternative components may affect strongly the behavior of PLS, and some recent studies have been proposed to better understand how they can affect the quality of the results (Liefooghe *et al.*, 2009, 2011).

Here we introduce alternative designs for the algorithmic components of PLS, and how they affect the anytime behavior of the resulting algorithms for our benchmark problems.

Alternatives for the selection step. PLS does not make use of any information on the current state of the archive in the selection step but uses a uniform random choice. An alternative approach is to select solutions whose exploration may have the largest potential to improve the current archive. In the previous chapter (see Section 3.6), we proposed a formal indicator to measure the improvement that can be expected from the selection of specific pair of solutions (as a replacement to the Euclidean norm). This indicator, the Optimistic Hypervolume Contribution (ohvc), estimates the improvement to the archive in terms of the hypervolume indicator (see Section 2.6.1), of selecting a specific pair of solution. It is based on the idea that solutions that are close in the objective space are also close in the search space. Hence, by exploring the neighborhood of a given solution, one can expect to find new non-dominated solutions in the region between the current solution and its closest neighbors in the objective space. This idea is also underlying the PLS algorithm, and the optimistic hypervolume contribution is therefore relevant in the PLS concept. In this chapter, we will use a similar measure. However, instead of estimating the improvement from the selection of a pair of solution, we want to estimate the improvement from the selection of a single solution (as PLS selects solutions one by one to explore their neighborhoods). We call this measure the Optimistic Hypervolume Improvement (OHI), and we define the OHI of a solution s as

$$\mathsf{OHI}(s) = \begin{cases} 2 \cdot \mathsf{ohvc}(s, s_{\text{sup}}) & \text{if } \nexists s_{\text{inf}}, \\ 2 \cdot \mathsf{ohvc}(s_{\text{inf}}, s) & \text{if } \nexists s_{\text{sup}}, \\ \mathsf{ohvc}(s, s_{\text{sup}}) + \mathsf{ohvc}(s_{\text{inf}}, s) & \text{otherwise,} \end{cases} \tag{5.1}$$

where s_{sup} and s_{inf} are the closest neighbors of s in the objective space from the current archive $\mathcal{A}_0$ defined as

$$\begin{aligned} s_{\text{sup}} &= \underset{s_i \in \mathcal{A}_0}{\arg\min} \{ f_2(s_i) \mid f_2(s_i) > f_2(s) \} \\ s_{\text{inf}} &= \underset{s_i \in \mathcal{A}_0}{\arg\max} \{ f_2(s_i) \mid f_2(s_i) < f_2(s) \}. \end{aligned} \tag{5.2}$$

Either s_{sup} or s_{inf} may not exist if s is the best solution for f_2 or f_1, respectively. In such a case, we define the OHI to be two times the optimistic hypervolume con-

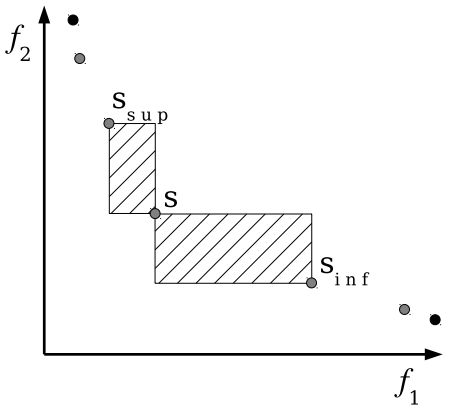

Figure 5.1: Representation of the normalized objective space. The OHI of solution s is the sum of the two hatched areas that lie between s and its two closest neighbors in the objective space, s_{inf} and s_{sup}. The OHI of extreme solutions (in black) is twice the area between them and their closest neighbor.

tribution of the existing solution, in order to avoid a strong bias against extreme solutions. Fig. 5.1 shows a graphical representation of the OHI indicator.

We refer to the random selection component of the original PLS algorithm as $\langle$RND$\rangle$, and to the new selection component that uses OHI with $\langle$OHI$\rangle$.

Alternatives for acceptance criterion. The original PLS algorithm accepts any non-dominated solution for inclusion in the archive. We call this component $\langle\nprec\rangle$ for non-dominated. However, different criteria could be used, particularly more restrictive ones in order to avoid an explosion of the number of solutions in the archive. In particular, accepting only neighbors that dominate the current solution may allow a quick convergence to the Pareto front at the price of a possible loss of quality. We call this component $\langle\succ\rangle$ for dominating. It is also possible to use a component that switches from one rule to another: if a solution that dominates the current one is found, only such solution is accepted, and if no dominating solution can be found, the acceptance criterion switches to accepting solutions that are non-dominated. We call this component $\langle\succ\nprec\rangle$.

Alternatives for neighborhood exploration. The rule that PLS uses when exploring the neighborhood of the selected solution corresponds to the *best-improvement* neighborhood exploration in single-objective local search, i.e., all neighboring solutions are explored (and potentially added to the archive if accepted) before moving to a new solution. We call this component $\langle*\rangle$. In single-objective local search, the alternative *first-improvement* neighborhood exploration is often more efficient. In the multi-objective case, it corresponds to stopping the neighborhood exploration

as soon as one neighboring solution has been accepted. We call this component $\langle 1 \rangle$. It is also possible to design a *switching rule*: when all solutions in the archive have been explored using the *first-improvement* rule, the algorithm will mark all solutions as unexplored and switches to the *best-improvement* rule. This allows to explore the part of the neighborhood that was not explored by using the first-improvement rule, but it is triggered only when PLS gets stuck in what is already a closer approximation to the Pareto front, w.r.t. the initial set of solutions. We call this latter component $\langle 1* \rangle$.

In what follows, we denote each variant of the original PLS algorithm by specifying which alternative components it is using. For instance, PLS$\langle$OHI, $\succ\nmid, 1* \rangle$ denotes the variant that

- uses OHI for the selection step;
- uses a rule for the acceptance criterion that switches from dominating to non-dominated;
- uses a rule for the neighborhood exploration that switches from stopping after the first accepted neighbor to stopping only after considering all accepted neighbors.

If the variant for a component is not specified for a given strategy, it uses the original version of this component as described in Section 5.1. Thus, following our notation, the original PLS is noted PLS$\langle$RND, $\nmid, * \rangle$.

In the rest of this section, we first present an additional benchmark that we use for our study, the Quadratic Assignment Problem, then we detail the experimental setup and the results of the analysis of the different variants of PLS components.

5.2.1 The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is one of the most widely studied NP-hard combinatorial optimization problems, given its high relevance in real-world situations (Burkard *et al.*, 1998; Çela, 1998). This problem arises when designing the layout for a set of "facilities" (for instance, hospitals, factories, keyboard keys, electronic components on boards, etc) that must be placed on a set of possible locations. More precisely, in

the QAP is given a set of n facilities $P = \{p_i, \ldots, p_n\}$ and a set of n possible locations $L = \{e_i, \ldots, e_n\}$. A flow matrix F defines, for each pair of facilities (p_i, p_j) a flow (also called weight) between them, f_{ij}. A second matrix defines for each pair of locations (e_k, e_l) a distance d_{kl}. A feasible assignment of facilities to locations can be described by a permutation π, where $\pi(i)$ gives the location of a facility i. The cost contribution c_{ij} of assigning facilities to locations is then the product of their flow and distances; which is $f_{ij} \cdot d_{\pi(i)\pi(j)}$. The objective in the QAP is to assign facilities to locations such that the sum of all contributions $\sum f_{ij} \cdot d_{\pi(i)\pi(j)} \forall i, j, i \neq j$, is minimized.

Instances of the bi-objective version of this problem (bQAP) that we use in this chapter are obtained by generating two different flow matrices that are combined with a single distance matrix. Note that the distance matrix can be switched with the flow matrix without changing the characteristics of an instance, that is, the designation "flow" and "distance" is arbitrary.

5.2.2 Experimental Setup

Initial Sets to Start Pareto Local Search

In our experimental analysis, we cover all possible uses of PLS by using three different initial conditions, that is, sets of solutions that PLS starts from.

- **High-quality sets (`HQS`).** State-of-the-art algorithms for several problems (Lust and Teghem, 2010b,a) use PLS in a second phase to refine a set of high quality solutions obtained from a first phase, which often is based on scalarizations of the problem. To cover this usage of PLS, we use as the initial set five high quality solutions that are well-spread over the objective space. In fact, we use the improved adaptive TPLS algorithm proposed in Chapter 3, that tends to distribute solutions as evenly as possible in terms of the hypervolume of the non-dominated set.

- **Two high-quality solutions (`TS`).** It may be possible that there is no algorithm available to solve the scalarized problems, but an algorithm is available to solve the two possible single objective versions of the bi-objective problems. In this case, one may be able to obtain two solutions that are of high-quality for each objective. PLS can then be started from these two solutions.

- **Random solution (`RS`).** Finally, it is also possible that no algorithm is available to solve any single-objective version of the problem, i.e., one must rely only on PLS to tackle the problem. In this case, PLS could start from a random initial solution.

In the next section, we focus on the algorithmic components of PLS, test possible variants of them and examine their impact on the anytime behavior of the resulting algorithms.

Experimental Benchmark and Initial Solutions

We generated three bTSP instances with 500 cities. The two distance matrices of each instance are generated independently of each other and are symmetric, isometric Euclidean TSP instances (generated in the same way as explained in Section 3.3.2).

Additionally to the bTSP, we use the bQAP and the instance generator proposed in Knowles and Corne (2003a). We generated 3 instances with correlations between the flow matrices in $\{-0.75, -0.5, 0, 0.5, 0.75\}$. The lower the correlation, the higher are the run times of PLS to reach completion, since the number of non-dominated solutions increases strongly with negative correlation (Paquete and Stützle, 2006). On the other hand, instances with correlation zero or larger take very short time for PLS to terminate, therefore, they are not useful for illustrating improvements in anytime behavior, and we only present in this chapter results with correlation -0.75 and -0.5.

The initial solutions, when starting from two or more high-quality solutions are obtained solving scalarized problems. For the bTSP, the single-objective algorithm used to tackle these problems is an iterated local search based on 3-opt moves. For the bQAP, we use a simulated annealing algorithm (Hussin and Stützle, 2010). For both problems, the single-objective algorithms were given two seconds for each scalarization.

Performance Assessment

In this chapter we rely on the hypervolume (HV) indicator, presented in Chapter 2, Section 2.6.1. Due to the large size of the non-dominated sets that are involved during the search, it is not feasible to record them and find a posteriori the bounds to be used for the normalization and the computation of the hypervolume of the sets. Instead, we rely

on bounds that are found a priori so the hypervolume of the non-dominated sets can be computed on the fly and recorded directly. For the bTSP, we found the lower bound for normalization using the exact Concorde solver (Applegate *et al.*), release 03.12.19, and the upper bound for normalization by taking the worst solution value of $100\,000$ random solutions that are sampled uniformly at random. For the bQAP, we do not use the same procedure since no exact solver is available that could solve instances of the size we use in reasonable computation times. Therefore, we ran the original PLS algorithm three times, using all initial conditions, and we record the best and the worst value obtained over all results. We then define the lower bound as being the *best value* multiplied by 0.95, and the upper bound as being the *worst value* multiplied by 1.05. Note that we check during the experimental analysis that the bounds are never attained or exceeded in any result we obtained.

All objective values are then normalized into the range $[1, 2]$, and we use the coordinates $(2.1, 2.1)$ as the reference point for computing the hypervolume of the normalized sets.

Computational Environment

We assess graphically the anytime behavior by plotting the average hypervolume (over 25 runs) of the normalized sets over computation time. To do so, we define a priori a sequence of time steps, at which we normalize the current non-dominated set obtained so far by the algorithm, we compute its hypervolume and we record it. Note that the computation time required to perform these steps, which are required to observe the behavior of the algorithms but are not actually part of them, are not deducted from the overall computation time which is measured and reported in our results. By using an exponential scale for the sequence of time steps, and therefore also for the plot, we can observe the behavior of the algorithms both after short periods of time and at larger scales. More precisely, we use 100 time steps, computed as $t_i = \exp(i \cdot \ln(\texttt{max\_time})/100) - 1, \quad i \in 1, \ldots, 100$, where `max_time` is a cut-off time (see Table 5.1), determined from preliminary runs of the algorithms. We also show graphically the variance of each algorithm across the multiple runs by plotting in gray the confidence intervals corresponding to each curve.

Problem	Instance type	Cut-off time (s)
bTSP	Size = 500	10 000
bQAP	Size = 100, Correlation = -0.75	1200
	Size = 100, Correlation = -0.5	100

Table 5.1: Cut-off times used for each problem and type of instances.

The algorithms are implemented in C++, compiled with gcc 4.4.6, and the experiments were run on a single core of AMD Opteron 6272 CPUs, running at 2.1 Ghz with a 16 MB cache under Cluster Rocks Linux version 6/CentOS 6.3, 64bits.

5.2.3 Experimental Evaluation of Alternative Components

We now present the experimental evaluation of the anytime behavior of the PLS variants. To make the presentation concise, we present in this thesis plots for only one instance of each problem (and correlation for the bQAP). Different instances for the test problems show remarkably similar results, and the conclusions drawn in this chapter are true for all instances we tested. All plots for all additional instances are available on-line as supplementary material (Dubois-Lacoste *et al.*, 2013a).

Selection Step

We present in Fig. 5.2 the evaluation of the selection component that uses the OHI against the original random one, ⟨RND⟩. The top three plots present the result obtained for one bTSP instances, while the other present results for two bQAP instances (with correlation -0.75 for the middle and correlation -0.5 for the bottom ones) for three experimental conditions, that is, starting with a random solution (left), two high-quality solutions (middle) and a set of high-quality solutions (right). In most cases, PLS⟨OHI⟩ outperforms PLS⟨RND⟩ at any moment of the search, sometimes by a large gap. It shows that the selection of the most promising regions actually help reaching better results in less computation time. Other plots show results that are more similar for both strategies, such as the top-left plot that presents results for a bTSP instance when starting from

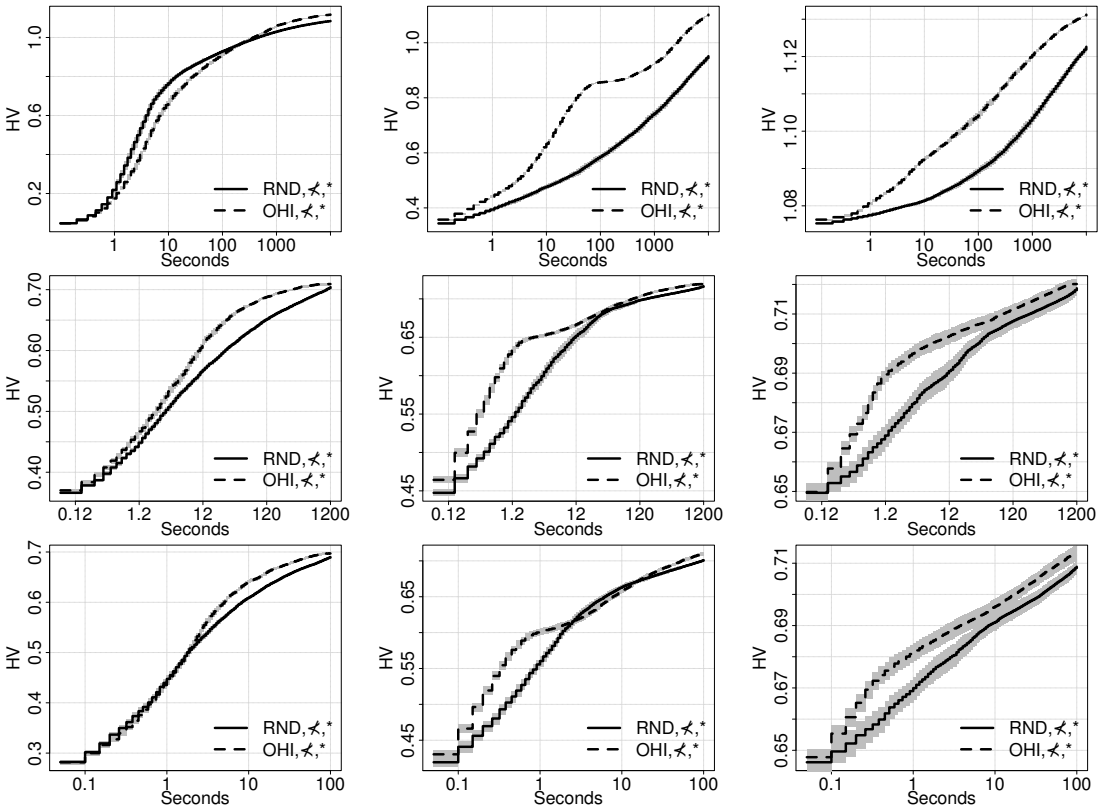

Figure 5.2: Selection Step: PLS⟨RND, $\nprec$, $*$⟩ vs PLS⟨OHI, $\nprec$, $*$⟩ for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). Initial conditions are RS (left), TS (middle) and HQS (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

an initial set that is RS. This configuration is the only one where OHI appears slightly worse for a short period of time (between 1 and 100 seconds); this small difference is likely due to the overhead of computation time required to perform the selection using OHI.

However, overall there is no ambiguity, and the selection based on OHI is a clear improvement over the original component. For this reason, we use it in the rest of this section, and we test whether we can improve the anytime behavior further by using the variants for the other components.

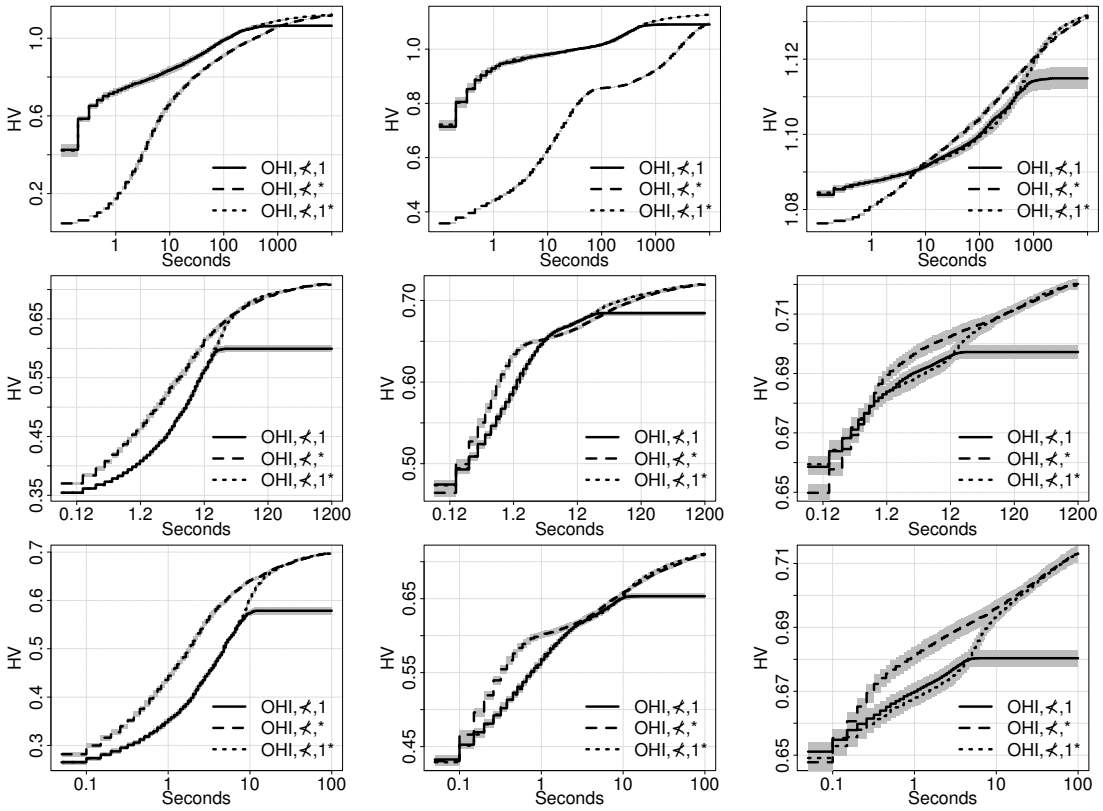

Figure 5.3: Neighborhood Exploration: PLS$\langle$OHI, $\prec$, 1$\rangle$ vs PLS$\langle$OHI, $\prec$, $*\rangle$ vs PLS$\langle$OHI, $\prec$, $1*\rangle$ for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). Initial conditions are RS (left), TS (middle) and HQS (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

Neighborhood Exploration

Fig. 5.3 reports a comparison of the different alternatives for the neighborhood exploration. It shows the anytime behavior of three alternatives to PLS: PLS$\langle$OHI, $\prec$, 1$\rangle$, which stops the exploration of the neighborhood after accepting one solution; PLS$\langle$OHI, $\prec$, $*\rangle$ which explores the full neighborhood, and PLS$\langle$OHI, $\prec$, $1*\rangle$, which switches from the first to the second rule. The behavior of PLS$\langle$OHI, $\prec$, $1*\rangle$ before the switch is exactly the same as PLS$\langle$OHI, $\prec$, 1$\rangle$, therefore the two curves coincide. PLS$\langle$OHI, $\prec$, 1$\rangle$ does finish rather quickly due to the limited neighborhood exploration. This can be seen as the fact that the curve for the hypervolume development is parallel to the x-axis. PLS$\langle$OHI, $\prec$, $1*\rangle$ then allows to follow from a solution quality perspective PLS$\langle$OHI, $\prec$, $*\rangle$. It is clear,

therefore, that PLS$\langle$OHI, $\nprec$, $1*\rangle$ should be preferred over PLS$\langle$OHI, $\nprec$, $1\rangle$ in every situation.

For the bTSP, when the initial set is either RS or TS, PLS$\langle$OHI, $\nprec$, $1*\rangle$ shows a much better anytime behavior than the original component used by PLS$\langle$OHI, $\nprec$, $*\rangle$. However, for the bTSP with the HQS set and all cases for the bQAP, PLS$\langle$OHI, $\nprec$, $*\rangle$ has better anytime than PLS$\langle$OHI, $\nprec$, $1*\rangle$. Therefore, the strategy of choice for this component is dependent of the situation. Overall, considering a situation without any preliminary knowledge, these results indicate that the original component for the neighborhood exploration might be preferred.

Since no strategy outperforms the other in every case, it could be interesting to test different trade-offs between PLS$\langle$OHI, $\nprec$, $*\rangle$ and PLS$\langle$OHI, $\nprec$, $1*\rangle$, for example, by accepting more than one solution (keeping the original behavior after it switches). However, it is likely that the best number of solutions to accept is strongly problem-dependent. Hence, we leave a further exploration in this direction for future research.

Acceptance Criterion

Fig. 5.4 presents a comparison of the acceptance criteria in PLS. While PLS$\langle$OHI, $\succ\nprec$, $*\rangle$ is able to find dominating neighboring solutions, its behavior is the same as PLS$\langle$OHI, $\succ$, $*\rangle$; this explains why their two curves can coincide in the initial phases of the search (see results with RS initial solution). However, PLS$\langle$OHI, $\succ$, $*\rangle$ quickly reaches completion since from some point on it cannot find any new dominating solutions. An extreme case happens when the initial set in the objective space is close to the Pareto front. In that case, no solution can be found that dominates any of the initial ones. This explains the entirely flat curve of PLS$\langle$OHI, $\succ$, $*\rangle$ when starting from TS or HQS.

PLS$\langle$OHI, $\succ\nprec$, $*\rangle$ compares very positively to PLS$\langle$OHI, $\nprec$, $*\rangle$; in many cases it shows a large improvement of quality at any time of the execution (see for instance the top plots, for the bTSP starting from RS or TS). There is no case showing the opposite situation, and overall the switching component of PLS$\langle$OHI, $\succ\nprec$, $*\rangle$ yields a better anytime behavior than the original one.

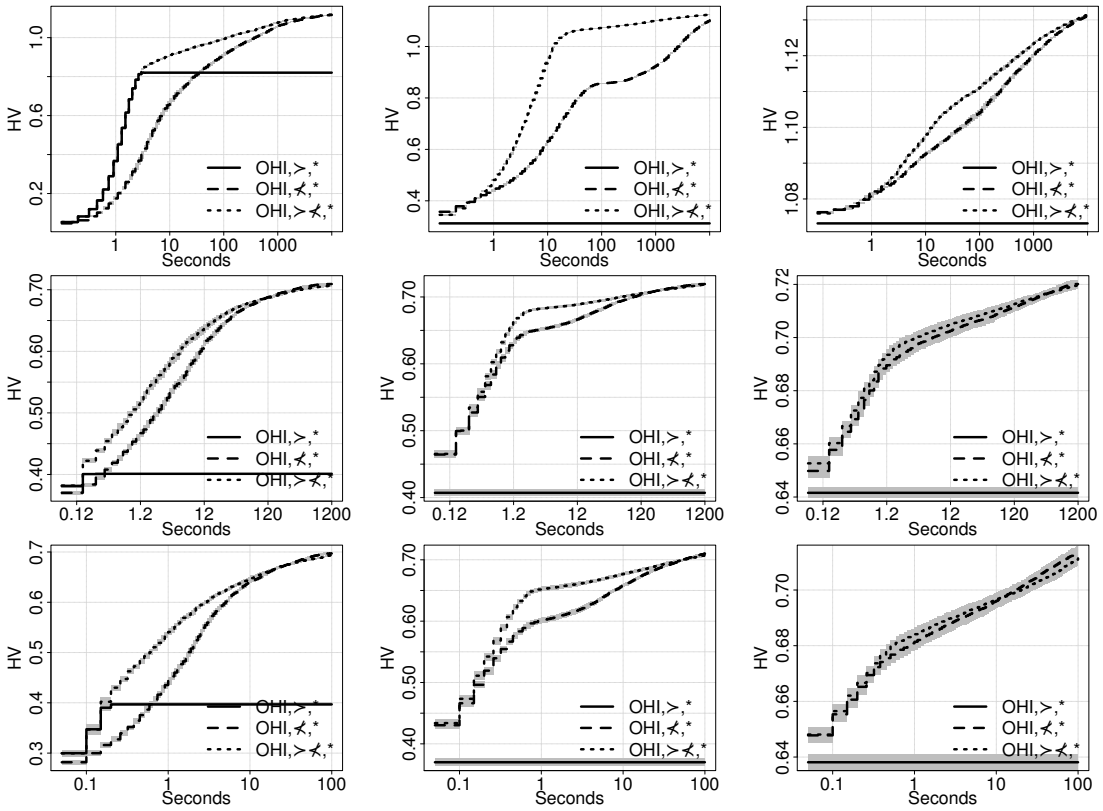

Figure 5.4: Acceptance Criterion: PLS$\langle$OHI, $\succ$, $*\rangle$ vs PLS$\langle$OHI, $\nprec$, $*\rangle$ vs PLS$\langle$OHI, $\succ\nprec$, $*\rangle$ for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). Initial conditions are RS (left), TS (middle) and HQS (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

Combination of Components

We have shown that the switching alternative for the acceptance criterion, improves significantly the anytime behavior over the original component in most of the cases. We also have shown that the switching alternative for the neighborhood exploration can be helpful, but unfortunately not in all cases. As a last step, we explore whether even better anytime behavior can be obtained by combining these two components in PLS$\langle$OHI, $\succ\nprec$, $1*\rangle$. Fig. 5.5 compares PLS$\langle$OHI, $\succ\nprec$, $1*\rangle$ with the strategies evaluated in the previous two sections. The plots show that in none of the cases, PLS$\langle$OHI, $\succ\nprec$, $1*\rangle$ improves consistently over its two competitors. Hence, there are interactions between the components that prevent their respective advantages to result in even better be-

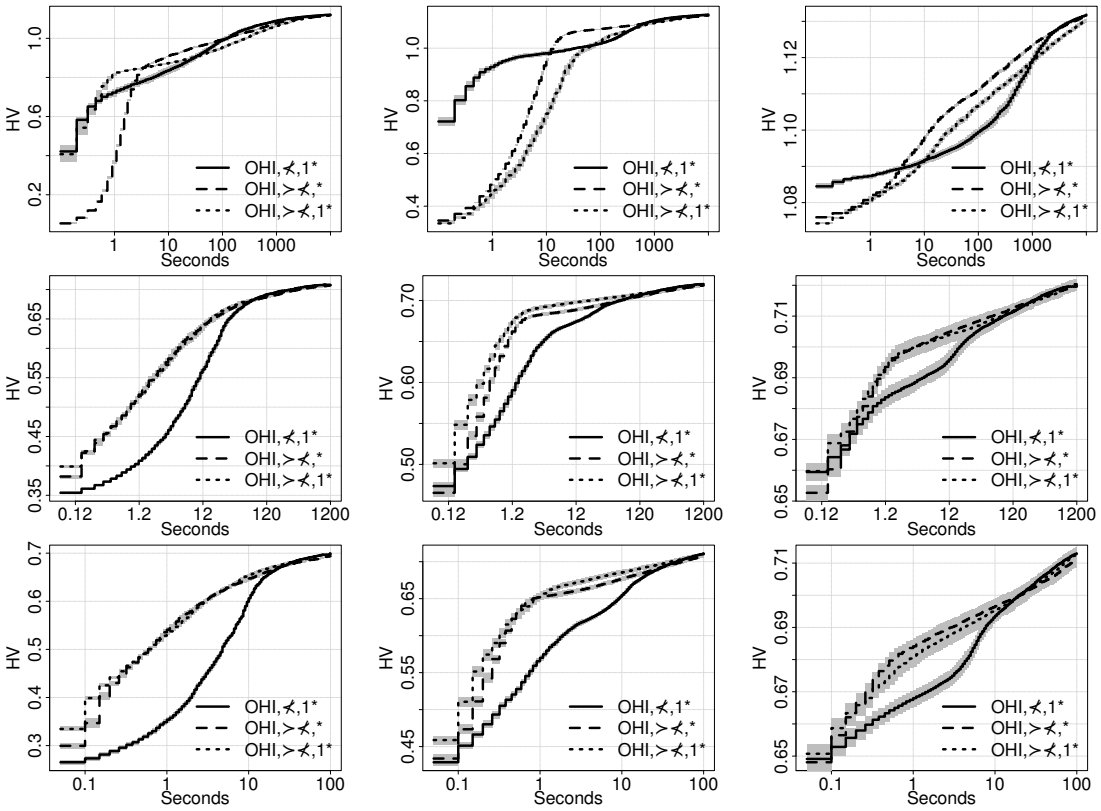

Figure 5.5: Combination of alternative components: PLS⟨OHI, $\nprec$, $1*$⟩ vs PLS⟨OHI, $\succ\nprec$, $*$⟩ vs PLS⟨OHI, $\succ\nprec$, $1*$⟩ for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). Initial conditions are RS (left), TS (middle) and HQS (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

havior. We conjecture that this is because PLS⟨OHI, $\succ\nprec$, $*$⟩ and PLS⟨OHI, $\nprec$, $1*$⟩ restrict in two different ways the number of solutions that are considered for inclusion in the archive and that probably the restriction incurred by their combination is too strong.

5.3 Objective Space Discretization

The best performing PLS variants for neighborhood exploration and acceptance criterion have in common that they restrict the number of non-dominated solutions added to the archive. In particular, such a restriction improves the anytime behavior in the

early phases of the search (before the "switches") probably by pressuring PLS towards regions in the objective space close to the Pareto front. This pressure is exerted by avoiding the acceptance of too many non-dominated solutions into the archive.

In this section, we exploit further this observation. We explicitly limit the potential number of non-dominated solution to enter the archive by using, in addition to Pareto dominance, a specific mechanism that we will explain next.

5.3.1 Epsilon-Grid Discretization

Many different methods have been proposed to limit the size of the archive of non-dominated solutions when tackling multi-objective problems. Some of these methods focus on limiting explicitly the number of solutions; these are called *archiving* mechanisms. For a review of the most common methods and their respective properties, we refer to López-Ibáñez *et al.* (2011b).

Other methods do not explicitly restrict the number of non-dominated solutions to a fixed value. Instead, they aim at obtaining well-distributed solutions in the objective space. To do so, these methods avoid to have solutions that are too close to each other, by defining a grid that discretizes the objective space into regions that can contain at most one solution. In this chapter, we use the *epsilon-grid* mechanism proposed in Angel *et al.* (2004). It discretizes the objective space into boxes the size of which increases with the distance to the axis; in this way, more solutions can be accepted in the center of the Pareto front than in its tails. The size of the boxes is defined by a parameter ϵ. Fig. 5.6 shows graphically this epsilon-grid mechanism.

More formally, for an objective d that we assume to be normalized, the bounds of a box i are computed as:

$$B_i^d = \left[\lfloor \epsilon^i \rfloor, \lfloor \epsilon^{i+1} \rfloor\right[, \quad \epsilon > 1. \tag{5.3}$$

Then we can determine directly in which box a solution s is for objective d with:

$$B^d(s) = B^d_{\lfloor log(norm(f_d(s)))/log(\epsilon) \rfloor}, \quad \epsilon > 1. \tag{5.4}$$

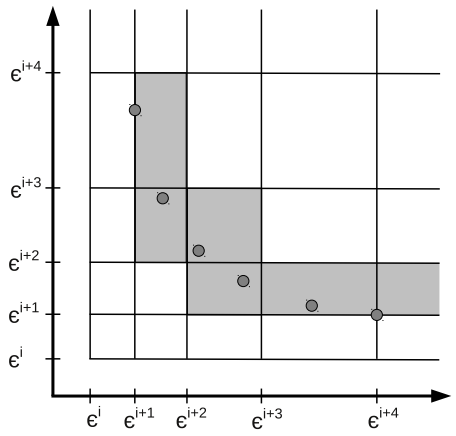

Figure 5.6: Representation of the normalized objective space with the epsilon-grid mechanism. This mechanism does not allow a new solution s' to enter a box already filled by another solution s (those boxes are shown in gray), unless s' dominates s.

The $norm()$ function gives the normalized value for a given objective. We use a mapping of the objectives such that the lower bound provided for a given objective (see Section 5.2.2) is mapped to 1, the upper bound to $100\,000$, and any value in between is mapped linearly to the range $[1, 100\,000]$.

The main drawback of the epsilon-grid mechanism is that the parameter ϵ (denoted by ϵ in Eq. 5.3 and 5.4) must be defined carefully before solving a problem. Therefore, it can not be applied without previous knowledge of the problem (and even instance) to be tackled (Paquete, 2005).

5.3.2 A Generic Grid Mechanism: Dynamic Adaptation

In this chapter, we propose a new mechanism to dynamically adapt the value of ϵ and, hence, the grid size, within the PLS algorithm. The aim of this mechanism is threefold. First, it allows to apply this method without any previous knowledge on the problem or the instance to be tackled. Second, it may help the PLS algorithm to improve its anytime behavior. Third, it is generic and it can therefore be applied directly to other problems and possibly also to other methods.

Algorithm 13 Dynamic Epsilon Grid

1: **Input:** An initial set of non-dominated solutions: $\mathcal{A}_0$,
2: A value that defines the initial grid: ϵ_0,
3: A ratio to decrease the value of ϵ: r.
4: $\mathcal{A} := \mathcal{A}_0$
5: $\epsilon := \epsilon_0$
6: **while** ! `termination criterion` **do**
7: $\mathcal{A} := \mathsf{PLS}_{Grid}(\mathcal{A}, \epsilon)$
8: $\epsilon := 1 + (\epsilon - 1) \cdot r$
9: $\mathsf{explored}(s) := \text{FALSE}, \quad \forall s \in \mathcal{A}$
10: **end while**
11: **Output:** $\mathcal{A}$

The dynamic adaptation of the ϵ value is triggered by the convergence of PLS. Each time the PLS algorithm with a current grid setting reaches completion, that is, all solutions are marked as explored, the value of ϵ is reduced by a fixed factor. This mechanism is described also in Algorithm 13. A PLS algorithm using the grid defined by ϵ (denoted by PLS_{Grid}, line 7 of the pseudo-code) starts from the current solution archive $\mathcal{A}$. When PLS_{Grid} reaches completion and all solutions in the archive are marked as explored, the value of ϵ is decreased by a factor $r < 1$ (line 8), and the solutions are marked again as unexplored. Then PLS_{Grid} is re-started using the grid defined by the updated epsilon value (line 9). This process (lines 6 to 10) is repeated until a given termination criterion is met.

The rationale behind this algorithm is to obtain quickly well-distributed solutions in the objective space, thus reaching quickly a good Pareto front approximation in terms of hypervolume, using large values for ϵ, and then reducing the value of ϵ to deal with more solutions only when PLS actually needs them to improve the quality of the results.

In our experimental analysis, we set the initial value for ϵ to 5. This is a rather large value (recall that ϵ has an exponential effect on the boxes size), which makes unlikely even the acceptance of one new solution when starting from the initial set. This is done on purpose to avoid the usage of problem-specific knowledge as much as possible. For the value of r, we tested values in $\{0.9, 0.7, 0.5, 0.3, 0.1\}$ during preliminary experiments. The differences due to different values of parameter the r appeared minor, resulting in rather similar curves; we provide the plots for this comparison as supplementary

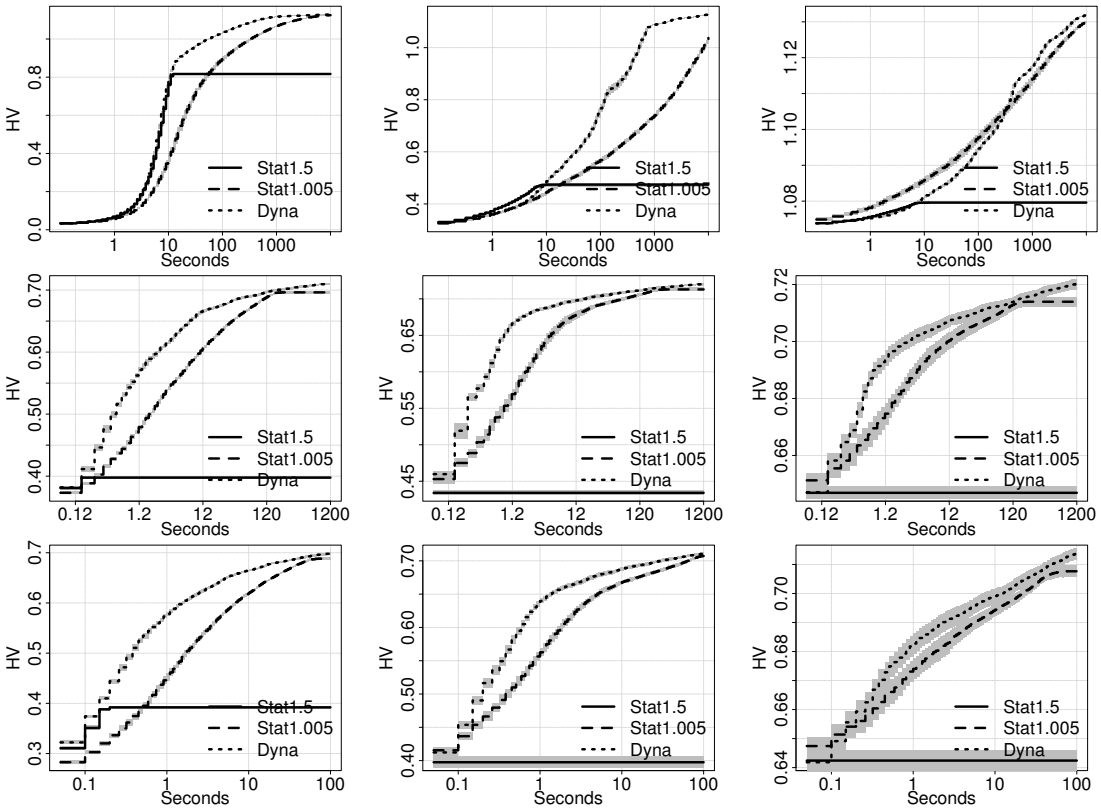

Figure 5.7: Experimental evaluation of PLS combined with Epsilon-Grid for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). We compare here two variants that use a static grid (with $\epsilon \in \{1.05, 1.005\}$), against the version that uses a dynamic one. Initial conditions are `RS` (left), `TS` (middle) and `HQS` (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

material (Dubois-Lacoste *et al.*, 2013a). For the following experiments, we chose a setting of $r = 0.5$ for the dynamic update of the grid. We call the resulting algorithm *Dynagrid*.

Fig. 5.7 compares the *Dynagrid* algorithm against variants that use different static grids, that is, they use different fixed values for ϵ. We tested values of $\epsilon \in \{1.5, 1.05, 1.005\}$, but for the sake of clarity we show here only those for value 1.5 and 1.005; the results for 1.05 are a trade-off between the results obtained by the two extreme values. Clearly, the larger ϵ, the faster is the quality improvement in short computation times, but the earlier PLS also stagnates, as visible by the fact that curves of the hypervolume development become flat, indicating that PLS actually stops.

For the bTSP with initial conditions RS and TS, the adaptive mechanism is able to improve the quality of the archive as quickly as does the coarse static grid ($\epsilon = 1.5$). Moreover, it outperforms the fine static grid ($\epsilon = 1.005$) at any time. Starting from HQS, it gives slightly worse solution quality than the fine grid until roughly 200 seconds, but it is better for larger computation times. For the bQAP, the results obtained by the dynamic grid are clearly better than those for fixed values of ϵ. The dynamic grid improves quality quickly and reaches higher quality results than the fine grid at any time for all initial conditions.

Overall, the high performance of the dynamic grid is remarkable because it does not require preliminary knowledge on an appropriate setting of ϵ when being applied.

5.3.3 An Improved Dynamic Grid: Hypervolume Enhancement

In the original epsilon-grid method (Angel *et al.*, 2004) a new candidate solution s' that falls into a same box as an already existing solution s in the archive only replaces s if s' dominates s. Therefore, solutions that enter the archive earlier are preferred over more recent candidate solutions for the sole reason that they were encountered earlier. However, since our goal is to obtain a set of non-dominated solutions with the highest possible quality at any time, it may be preferable to replace this mechanism, for example, allowing a replacement if some indicator value of the solution quality would improve. More precisely, since we evaluate the quality of the non-dominated sets in terms of hypervolume (see Section 2.6.1), we extend here our dynamic epsilon-grid (see Section 5.3.2) to optimize this indicator. In particular, here we allow a solution s' to replace a solution s if it increases the hypervolume of the archive. More formally, given a new solution s' that falls into the same box as another solution $s \in \mathcal{A}$, where $\mathcal{A}$ is the current archive, s' replaces s if $s' \prec s$ or $\mathsf{HV}(\mathcal{A}/\{s\} \cup \{s'\}) > \mathsf{HV}(\mathcal{A})$. Note that if $s' \prec s$, due to the properties of the hypervolume indicator (Zitzler *et al.*, 2003), the hypervolume of the archive resulting from the removal of s and the addition of s' can only increase. Therefore, the dynamic grid with *hypervolume enhancement* aims explicitly at optimizing the hypervolume of the current archive at any time.

Note that after a change of the value of ϵ the grid may change and, as a result, there may (after the update of the grid) be new boxes that contain more than one solution. In

this case, the solution that has the least hypervolume contribution among those in the box competes with the new solution for staying in the box. Finally, note that the consideration of the hypervolume enhancement is computationally cheap. For bi-objective problems, the archive can be kept sorted according to one objective, and for computing the hypervolume contribution only the neighboring solutions in the given ordering need to be considered.

Next, we evaluate the performance of the proposed hypervolume enhancement in terms of anytime behavior. In Fig. 5.8, we compare the *Dynagrid* algorithm, which was shown to outperform the static grid in the previous section, and *Dynagrid* extended by the hypervolume enhancement, called *Dynagrid-HV*. For the bTSP, with initial conditions `RS`, the hypervolume enhancement slightly slows down the initial increase of the solution quality between roughly 2 and 20 seconds, but for higher computation times it clearly improves over the *Dynagrid* algorithm. For the initial conditions `TS` and `HQS`, the improvement is very large: the hypervolume enhancement reaches a same quality than the *Dynagrid* algorithm in about one order of magnitude smaller computation times. For the bQAP, the hypervolume enhancement shows for a given computation time usually the same or better average quality than the *Dynagrid* algorithm, although the improvement is typically smaller than in the bTSP case. The improvements are clear for `RS` and `TS` initial sets; when starting from `HQS`, both alternatives bring similar results.

In summary, the hypervolume enhancement overall improves further the anytime behavior of PLS. Since it is not really worse, but sometimes clearly better than the basic *Dynagrid* algorithm, we would recommend its use.

5.4 Comparison of the Two Approaches

In sections 5.2 and 5.3, we designed alternatives to PLS by exploring two fundamentally different directions. First, we have shown in Section 5.2 that the best strategies for both problems obtained by designing alternative algorithm components are based on *switching* rules, namely PLS$\langle$OHI, $\nprec$, $1*\rangle$ for the bTSP and PLS$\langle$OHI, $\succ\nprec$, $*\rangle$ for the bQAP. Second, in Section 5.3, we designed alternatives to PLS that rely on a discretization of the objective space. We have shown that the best variant is obtained by exploiting two improvements that we proposed: the dynamic adaptation of the grid and the hyper-

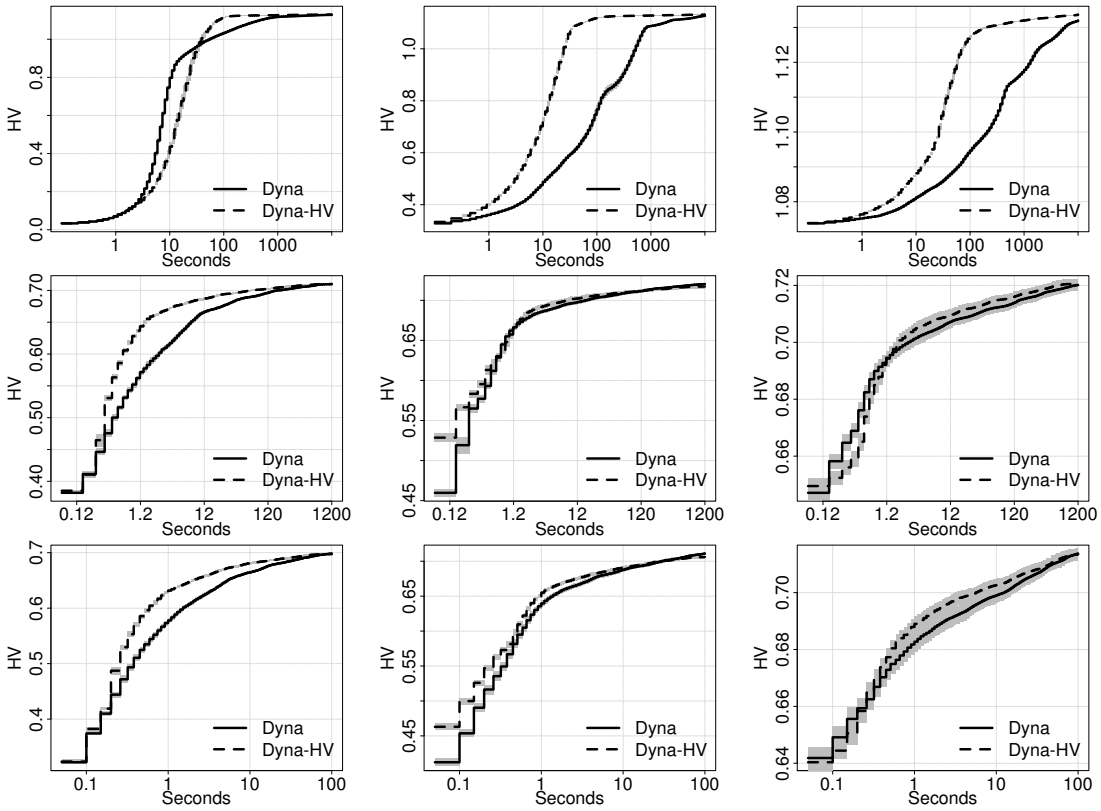

Figure 5.8: Experimental analysis of PLS combined with Epsilon-Grid for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). The plots present the comparison of the quality obtained by the dynamic epsilon grid, with and without the proposed hypervolume enhancement. The initial set is made of a random seed (left), 2 solutions (middle), a set of solutions (right). Initial conditions are `RS` (left), `TS` (middle) and `HQS` (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

volume enhancement. The latter variant, *Dynagrid-HV*, is consistently the best for both problems.

Here, we compare these two approaches. Fig. 5.9 presents a comparison of the best algorithms obtained from each of the two alternatives. Additionally, to highlight the improvement obtained by these new alternatives over the original PLS algorithm, we included also the original PLS in this comparison. For the bTSP, the new variant based on alternative components, PLS$\langle$OHI, $\nmid$, $1*\rangle$ dominates completely the original PLS algorithm. It also performs better than *Dynagrid-HV* up to roughly 20 seconds, that point

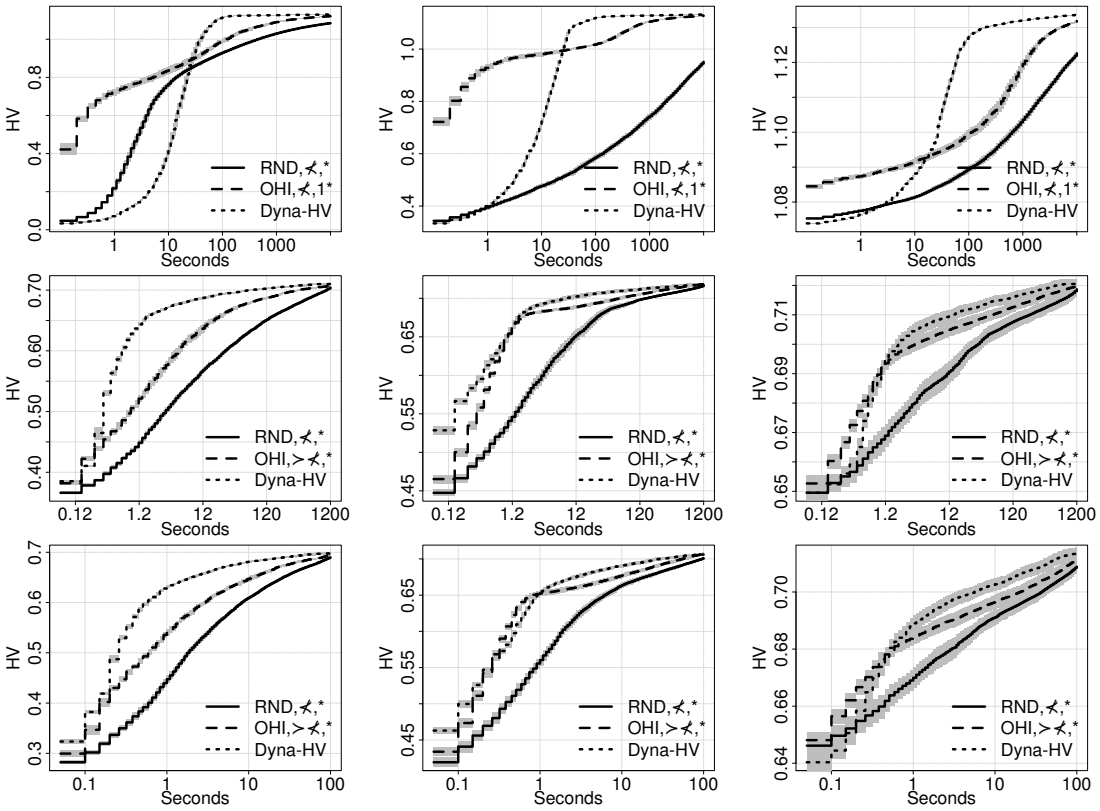

Figure 5.9: Experimental comparison of the original PLS algorithm against the best strategy obtained from the components variants and against the best strategy obtained from the objective space discretization, for one bTSP instance (top), and two bQAP instances with correlation -0.75 (middle) and -0.5 (bottom). The initial set is made of a random seed (left), 2 solutions (middle), a set of solutions (right). Initial conditions are `RS` (left), `TS` (middle) and `HQS` (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

in time at which the performance curves of the two algorithms cross. This crossing may be explained by the fact that *Dynagrid-HV* requires some time to adapt the value of ϵ, which penalizes it during the initial period. For the bQAP, PLS$\langle$OHI, $\succ\!\!\!\!\!\!\prec$, $*\rangle$ consistently and completely surpasses the original PLS algorithm, but it is itself mostly outperformed by *Dynagrid-HV*.

Problem	Instance type	Cut-off time (s)
bTSP	Size = 1000	30 000
bQAP	Size = 150, Correlation = −0.75	2000
	Size = 150, Correlation = −0.5	1000

Table 5.2: Cut-off times used for each problem and each type of large instances.

5.4.1 Scaling Behavior for Larger Instances

To determine whether our conclusions are consistent across different instance sizes, we analyze how the experimental results scale when tackling larger instances. To do so, we generated larger instances of size 1000 for the bTSP and 150 for the bQAP, following the same setup used for the other instances (see Section 5.2.2). Table 5.2 presents the cut-off times for these instances, for each problem.

We present a comparison of the same algorithms on these larger instances in Fig. 5.10. All the trends observed for smaller instances not only remain true, but are strengthened. In fact, the gap between the original PLS algorithm and the two new variants widens as it is most clearly visible on the bTSP. Hence, the performance improvements introduced by our new variants over the original PLS algorithm are particularly important when tackling large-scale bi-objective problems.

5.4.2 Statistical Comparison of the Best Strategies

We perform a statistical comparison of the selected strategies that were previously evaluated in this section. For the statistical tests we focus on specific snapshots in time that are at a 1000^{th}, 100^{th}, and a 10^{th} of the cut-off time, and the cut-off time itself. Since the time steps that we use increase exponentially (see Section 5.2.2), they do not necessarily correspond exactly to these selected times: in this case we use the closest ones.

We use the Friedman test to assess the significance of the differences, with a standard confidence level of 0.95 (thus, a p-value < 0.05 indicates that the null hypothesis is to be rejected). In the Friedman tests each instance and run is considered as a different

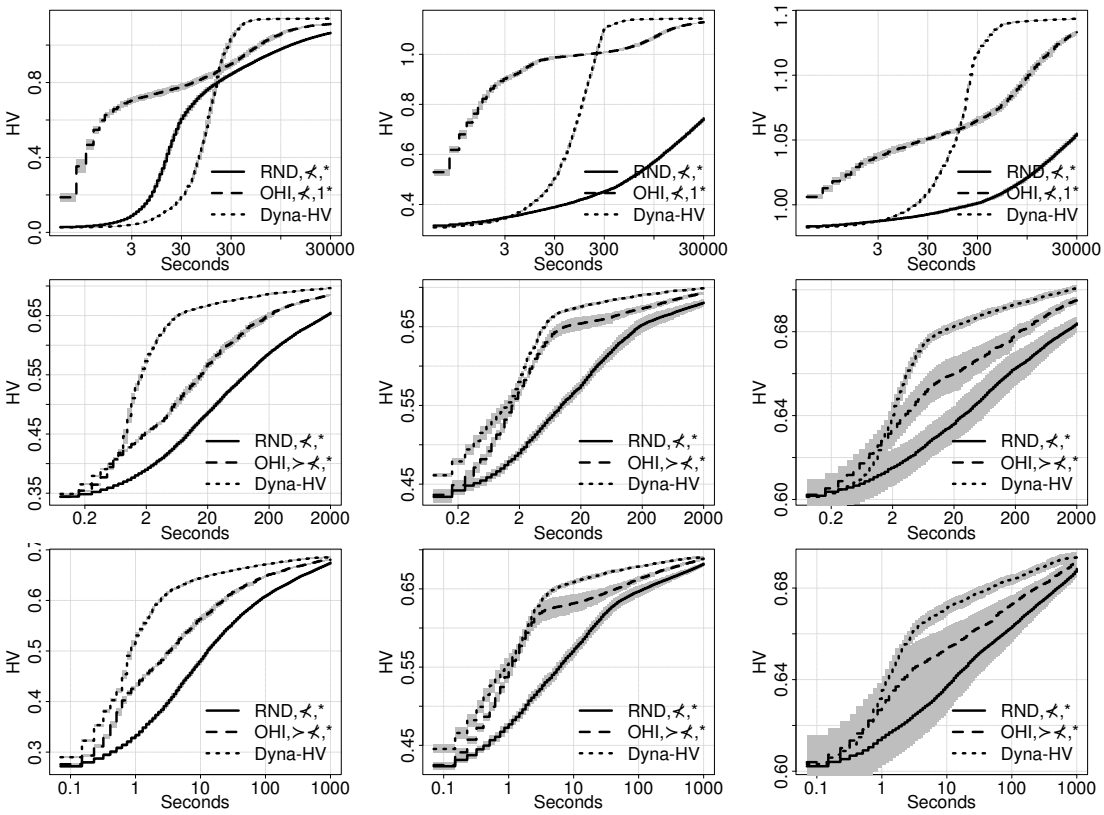

Figure 5.10: Experimental comparison of the original PLS algorithm against the best strategy obtained from the components variants and against the best strategy obtained from the objective space discretization, for one large bTSP instance of size 1000 (top), and two large bQAP instances of size 150 with correlation -0.75 (middle) and -0.5 (bottom). Initial conditions are `RS` (left), `TS` (middle) and `HQS` (right). The gray area corresponding to each curve shows the 95% confidence interval across different runs.

block. Table 5.3 presents this comparison for both problems. These results confirm (i) the improvement of the variants based on algorithmic component alternatives over the original PLS algorithm, and (ii) that *Dynagrid-HV* is almost always better in a statistically significant way than the other strategies. This, plus the fact that *Dynagrid-HV* does not require any previous knowledge on the problem being tackled, make it the strategy of choice.

5.5 Summary

In this chapter, we decompose, analyze and improve the Pareto Local Search algorithm, a well-known multi-objective algorithm, and an important component of state-of-the art algorithms for several high-impact problems (Paquete and Stützle, 2003; Lust and Teghem, 2010b,a).

We performed our experimental analysis on two widely studied problems, the bi-objective Traveling Salesman Problem and the bi-objective Quadratic Assignment Problem. Beyond the sole analysis of PLS' anytime behavior, we implemented and tested different alternatives to the original algorithm, studying the impact of these modifications to the anytime behavior of the resulting PLS variants. First, we considered different algorithm components for the selection of the solutions to be explored, the chosen solution's neighborhood exploration and the acceptance criteria for new solutions during the local search. This approach has previously shown promising results when PLS is restarted several times (Liefooghe *et al.*, 2009, 2011), and our experimental results indicate that such PLS variants also outperform the original algorithm in terms of anytime behavior. However, our results also indicated a drawback of this approach: the best variants to be used depend on the problem to be tackled.

The second approach that we explored is a discretization of the objective space into boxes, which are defined by a grid imposed on the objetive space. We significantly improve this discretization by adding two new mechanisms: (i) a *dynamic adaptation* of the grid size, and (ii) an *hypervolume enhancement*. In addition to the improved anytime behavior, our results show that the resulting PLS variant is fully generic, since it does not require a priori knowledge on the problem being tackled, and it is high-performing as it performs better than other variants: it is the best method for both problems we studied, all types of instances we tested and all initial conditions we tested.

For all problems where PLS was shown to be useful as a stand-alone algorithm or as part of a hybrid algorithm, our proposed variants can have a significant impact on the state-of-the-art in bi-objective optimization. Particularly promising is a combination of the variants proposed in this chapter with the mechanisms that were proposed to restart PLS several times when the computation time available is larger than the time required for PLS to terminate (Drugan and Thierens, 2010, 2012); such a combination

could make the resulting algorithm applicable to a very wide range of computation times, and, thus, real world situations. Another direction for future research is to focus on hybrid algorithms that make use of PLS and that have been shown to be state-of-the-art for several problems, to study the improvement on the quality results by using the new PLS variants within them. Given the improvements obtained by the two mechanisms we propose to enhance the epsilon-grid, it appears promising to apply them to different algorithms that also keep an archive of non-dominated solutions. Finally, it would be interesting to apply a similar experimental analysis to investigate additional multi-objective problems, in particular those with more than two objectives.

Table 5.3: Statistical analysis of the best variants of PLS (and the original one) at different time steps. The time steps are the cut-off time, and the closest ones to the cut-off time divided by 1000, 100 and 10. The numbers in parenthesis are the differences of the sum of ranks relative to the best variant, and ΔR_α gives the difference of the sum of ranks that is significant. The best variant, and other that are not significantly different are indicated in bold face. The alpha value for significance is 0.05.

QAP

Time	ΔR_α	Strategies (ΔR)
		Size 100, Correlation -0.75, Initial conditions RS
0001.18	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0011.84	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0123.19	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
1200.00	18.722	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (48), PLS⟨RND, ⊰, ∗⟩ (96)
		Size 100, Correlation -0.75, Initial conditions TS
0001.18	12.095	***Dynagrid-HV*, PLS⟨OHI, ≻⊰, ∗⟩ (9)**, PLS⟨RND, ⊰, ∗⟩ (117)
0011.84	3.966	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (74), PLS⟨RND, ⊰, ∗⟩ (148)
0123.19	11.559	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (63), PLS⟨RND, ⊰, ∗⟩ (132)
1200.00	23.767	**PLS⟨OHI, ≻⊰, ∗⟩, *Dynagrid-HV* (15)**, PLS⟨RND, ⊰, ∗⟩ (33)
		Size 100, Correlation -0.75, Initial conditions HQS
0001.18	11.937	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (15), PLS⟨RND, ⊰, ∗⟩ (120)
0011.84	12.173	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (51), PLS⟨RND, ⊰, ∗⟩ (129)
0123.19	16.328	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (63), PLS⟨RND, ⊰, ∗⟩ (111)
1200.00		p-value > alpha (no significant difference)
		Size 100, Correlation -0.5, Initial conditions RS
0000.10	6.08	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (65), PLS⟨RND, ⊰, ∗⟩ (145)
0001.00	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0010.02	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0100.00	18.65	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (57), PLS⟨RND, ⊰, ∗⟩ (96)
		Size 100, Correlation -0.5, Initial conditions TS
0000.10	8.31	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (72), PLS⟨RND, ⊰, ∗⟩ (141)
0001.00	11.87	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (17), PLS⟨RND, ⊰, ∗⟩ (121)
0010.02	6.82	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (72), PLS⟨RND, ⊰, ∗⟩ (144)
0100.00	18.51	**PLS⟨OHI, ≻⊰, ∗⟩, *Dynagrid-HV* (10)**, PLS⟨RND, ⊰, ∗⟩ (89)
		Size 100, Correlation -0.5, Initial conditions HQS
0000.10	16.74	**PLS⟨OHI, ≻⊰, ∗⟩**, PLS⟨RND, ⊰, ∗⟩ (53), *Dynagrid-HV* (109)
0001.00	9.82	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (61), PLS⟨RND, ⊰, ∗⟩ (137)
0010.02	12.92	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (78), PLS⟨RND, ⊰, ∗⟩ (126)
0100.00	21.68	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (53), PLS⟨RND, ⊰, ∗⟩ (64)
		Size 150, Correlation -0.75, Initial conditions RS
0001.90	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0019.92	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0203.58	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
2000.00	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 150, Correlation -0.75, Initial conditions TS
0001.90	9.75	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (45), PLS⟨RND, ⊰, ∗⟩ (135)
0019.92	2.79	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (73), PLS⟨RND, ⊰, ∗⟩ (149)
0203.58	5.56	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (76), PLS⟨RND, ⊰, ∗⟩ (146)
2000.00	7.92	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (57), PLS⟨RND, ⊰, ∗⟩ (141)
		Size 150, Correlation -0.75, Initial conditions HQS
0001.90	14.52	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (38), PLS⟨RND, ⊰, ∗⟩ (118)
0019.92	7.67	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (80), PLS⟨RND, ⊰, ∗⟩ (142)
0203.58	6.73	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (78), PLS⟨RND, ⊰, ∗⟩ (144)
2000.00	6.08	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (65), PLS⟨RND, ⊰, ∗⟩ (145)
		Size 150, Correlation -0.5, Initial conditions RS
0001.00	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0010.22	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0101.40	0	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
1000.00	15.62	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (62), PLS⟨RND, ⊰, ∗⟩ (115)
		Size 150, Correlation -0.5, Initial conditions TS
0001.00	11.6	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (23), PLS⟨RND, ⊰, ∗⟩ (124)
0010.22	2.79	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (76), PLS⟨RND, ⊰, ∗⟩ (149)
0101.40	5.47	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (79), PLS⟨RND, ⊰, ∗⟩ (146)
1000.00	17.67	***Dynagrid-HV*, PLS⟨OHI, ≻⊰, ∗⟩ (15)**, PLS⟨RND, ⊰, ∗⟩ (96)
		Size 150, Correlation -0.5, Initial conditions HQS
0001.00	16.06	***Dynagrid-HV*, PLS⟨OHI, ≻⊰, ∗⟩ (7)**, PLS⟨RND, ⊰, ∗⟩ (101)
0010.22	6.18	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (77), PLS⟨RND, ⊰, ∗⟩ (145)
0101.40	6.73	***Dynagrid-HV***, PLS⟨OHI, ≻⊰, ∗⟩ (78), PLS⟨RND, ⊰, ∗⟩ (144)
1000.00	22.03	***Dynagrid-HV*, PLS⟨OHI, ≻⊰, ∗⟩ (1)**, PLS⟨RND, ⊰, ∗⟩ (56)

TSP

Time	ΔR_α	Strategies (ΔR)
		TSP
		Size 500, Initial conditions RS
0009.97	4.77	**PLS⟨OHI, ⊰, 1∗⟩**, PLS⟨RND, ⊰, ∗⟩ (69), *Dynagrid-HV* (147)
0099.00	2.79	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (76), PLS⟨RND, ⊰, ∗⟩ (149)
0999.07	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
10000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 500, Initial conditions TS
0009.97	0	**PLS⟨OHI, ⊰, 1∗⟩**, *Dynagrid-HV* (75), PLS⟨RND, ⊰, ∗⟩ (150)
0099.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0999.07	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
10000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 500, Initial conditions HQS
0009.97	5.47	**PLS⟨OHI, ⊰, 1∗⟩**, *Dynagrid-HV* (67), PLS⟨RND, ⊰, ∗⟩ (146)
0099.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
0999.07	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
10000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 1000, Initial conditions RS
0029.02	0	**PLS⟨OHI, ⊰, 1∗⟩**, PLS⟨RND, ⊰, ∗⟩ (75), *Dynagrid-HV* (150)
0289.02	10.22	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (62), PLS⟨RND, ⊰, ∗⟩ (136)
3104.75	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
30000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 1000, Initial conditions TS
0029.02	0	**PLS⟨OHI, ⊰, 1∗⟩**, *Dynagrid-HV* (75), PLS⟨RND, ⊰, ∗⟩ (150)
0289.02	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
3104.75	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
30000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
		Size 1000, Initial conditions HQS
0029.02	2.79	**PLS⟨OHI, ⊰, 1∗⟩**, *Dynagrid-HV* (76), PLS⟨RND, ⊰, ∗⟩ (149)
0289.02	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
3104.75	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)
30000.00	0	***Dynagrid-HV***, PLS⟨OHI, ⊰, 1∗⟩ (75), PLS⟨RND, ⊰, ∗⟩ (150)

Chapter 6

Automatic Configuration of Hybrid Algorithms

This chapter focuses on the offline automatic configuration of algorithms. Automatic algorithm configuration, sometimes called tuning, is playing an increasingly important role in the design of effective heuristic algorithms. Proposed relatively recently, convenient and generic tools for this task are nowadays becoming feasible and available to support the design of effective heuristic algorithms. For this reason, the improvement of existing tools and the development of new ones is nowadays an active field of research. For an overview of prominent automatic algorithm configuration methods, we refer the reader to Section 2.8 in Chapter 2.

One of the side contributions of this thesis is the `irace` software package for offline automatic algorithm configuration (López-Ibáñez *et al.*, 2011a) which is also described in the appendix. In this chapter, we use this tool to automatically configure multi-objective algorithms based on Two-phase Local Search and Pareto Local Search, that were designed previously in this thesis. The automatic configuration of multi-objective algorithms is a recent methodology and this chapter contributes towards identifying its enormous advantages.

In Section 6.1, we explain how existing automatic configuration methods can be used to configure multi-objective optimizers. Next, in Section 6.2, we focus on the algorithm development carried out previously and we study whether automatic methods can achieve similar or better final quality than the careful manual setting derived pre-

viously in Chapter 4, which already was shown to improve over the state-of-the-art for bi-objective Permutation Flowshop Scheduling Problems. In Section 6.3, we perform similar experiments for the bi-objective Traveling Salesman Problem. Third, in Section 6.4, we make use of the `irace` software with the same goal as what was done previously "by hand" in Chapter 5, that is obtaining a good anytime behavior. Finally, Section 6.5 summarizes the main contributions of this chapter and discusses promising directions for future research.

6.1 Automatically Configuring Multi-objective Algorithms

If the actual use of automatic configuration methods to help the design of algorithms is relatively novel, their application to multi-objective algorithms is even more recent (López-Ibáñez and Stützle, 2010; Wessing *et al.*, 2010). The reason is that all proposed methods use either statistical tests that must be computed over samples that are numerical values, or they compare results directly, which assumes values are always comparable. These methods cannot handle results that are in the form of non-dominated sets of solutions that are very often incomparable with one another (see Section 2.5.2 in Chapter 2). This aspect makes existing methods hardly suitable to be directly used in the multi-objective context that we consider in this thesis.

Hence, one approach, which was proposed in López-Ibáñez and Stützle (2010), is to convert the non-dominated sets into a single number and then to use existing automatic configuration methods. This allows to configure multi-objective algorithms even though automatic configuration tools were originally proposed for the single-objective case. In this thesis, we rely on the same idea, and we use the hypervolume indicator (see Section 2.6.1) to apply `irace` to multi-objective algorithms. Note that any other unary indicator could potentially be suitable for this task (this can even be a way for the decision-maker to "inject" implicitly some preferences), and the general approach that we follow would be the same for any other indicator.

One key aspect of the experimental setup for the automatic configuration process is to use common bounds to compute the hypervolume to evaluate candidates. One pos-

sibility is to compute some bounds a priori, either per-instance or general ones. This can be done in research studies, but it is unlikely to be a satisfying solution in practical situations as it would require first to solve the training instances. Another possibility, that we use here, is to first run all candidate configurations on one instance, and then compute bounds based on the minimum and maximum values found for each objective. Using these bounds, the results can be normalized and then used to compute a normalized hypervolume.

6.2 Automatic Configuration for Final Quality: Permutation Flowshop Scheduling Problem

In Chapter 4, we proposed a new hybrid algorithm for bi-objective variants of the permutation flowshop problem, combining a first phase that solves scalarizations, based on Two-phase Local Search (TPLS) (more precisely, using new variants that we developed in Chapter 3), and a second phase that uses local search to search for non-dominated solutions, Pareto Local Search (PLS). This hybrid algorithm (TP+PLS) clearly outperforms the previous state-of-the-art. The main effort we invested in proposing this new state-of-the-art algorithm has been to carry out a large number of experiments in order to make the proper choices for the algorithmic design and to set the parameters values as good as we possibly could "by hand".

This algorithmic framework is an ideal candidate to apply automatic configuration techniques for two reasons: first, it has a large number of parameters that are difficult to set, and second, we have shown that it could reach state-of-the-art results, which makes any improvement over it highly valuable. Therefore, in this section we focus on configuring this hybrid algorithm with the goal of optimizing the final quality (in terms of hypervolume) of the Pareto front approximations after a given computation time.

We compare the automatically obtained configurations to the manually obtained configurations that were previously proposed in Chapter 4, where it was shown that the hybrid algorithm using the parameterization given there improves greatly upon previous state-of-the-art algorithms for five different bi-objective permutation flowshop scheduling problems. Hereafter, we call $conf_{hand}$ the reference configurations from

Table 6.1: Candidate configurations from Chapter 4, leading to state-of-the-art performance. We call these configurations $conf_{hand}$.

	tpls_tratio	ratioInitScal	nb_scal	two_seeds	restart	theta	pls_operator
50x20, all problems	0.9	1.5	12	1	0	0.25	exins
100x20, all problems	0.5	1.5	12	1	0	0.25	exins

Chapter 4. They differ for each instance size but they are the same for all problems; wherever we mention $conf_{hand}$ it refers to the size of instances that is precised by the context. These two configurations are presented in Table 6.1. For more details on the parameters we refer to Chapter 3 (p. 71, 73, 111) and Chapter 4 (p. 100 and 103).

6.2.1 Experimental Setup

We tackle the same bi-objective Permutation Flowshop Scheduling Problems (bPFSPs) as in Chapter 4, that is, all pairings of objectives makespan ($C_{\max}$), total completion time (SFT), tardiness (TT), and weighted tardiness (WT). For more details on these problems, we refer to Section 3.1.

Benchmark Set

For the automatic configuration process, we generated 500 training instances with two different sizes, 50 jobs and 20 machines (50x20) and 100 jobs and 20 machines (100x20). These instances were produced following the same procedure described in Chapter 4.

Use of Previous Knowledge

The configurations $conf_{hand}$ were obtained through a large set of careful "manual" experiments to understand the effect of each algorithm component, and the best design choice for each of them. The information acquired by doing so can be used during the tuning process: we run `irace` adding $conf_{hand}$ to the initial set of candidate configurations. The idea is to "seed" the automatic configuration process with the ex-

pertise gained from the manually-tuned configuration. This can hopefully improve the final configuration found by `irace`. We call $conf_{tun-rnd}$ the best configuration obtained from running `irace` without any knowledge of the $conf_{hand}$ configuration, and we call $conf_{tun-ic}$ the best configuration obtained from running `irace` using $conf_{hand}$ as an initial configuration. Again, from the context it should be clear when we mention $conf_{tun-rnd}$ and $conf_{tun-ic}$ to which size of instances and which problem we are referring to.

Experimental Setup

Each run of the hybrid algorithm, for the configuration process and for the experimental analysis, uses a time limit of $0.1 \cdot n \cdot m$ seconds (n being the number of jobs and m being the number of machines) to allow a computation time proportional to the instance size, as previously suggested by Minella *et al.* (2008). The automatic configuration process is stopped after 5000 runs, thus, the overall time used by the automatic configuration process for the instances with 100 jobs and 20 machines is 10^6 seconds. However, in practice, much less wall-clock time is required since `irace` was run using its parallel mode (see Section 2.8.1 and appendix).

For the experimental analysis of the configurations, we use 100 test instances for each size, produced following the same procedure as for the tuning instances. These instances, however, are different from the ones used in the tuning process, to avoid an overtuning effect that could bias the comparison. Each experiment is repeated 10 times with different random seeds.

To normalize the hypervolume value across all instances, we first normalize all non-dominated points to the range $[1, 2]$. This is done by considering for each instance the smallest values found for an objective across all runs and mapping these values to one; analogously, we map the largest values ever found for the objectives to 2. Then we compute the hypervolume of the set of points we obtained, using as reference point $(2.1, 2.1)$.

6.2.2 Experimental Results

For each problem and each instance size we report in Table 6.2 the best configurations we obtained. Surprisingly, we can see that for *PFSP*-(*SFT*, *TT*) and instances of sizes `100x20`, the best configurations obtained from the two runs of `irace` almost do not make use of PLS, TPLS being run for most of the available computation time. This configuration is clearly different from the one used in Chapter 4 and shown in Table. However, given the results in Minella *et al.* (2008), this choice is actually understandable. In fact, for the *PFSP*-(*SFT*, *TT*) only very few non-dominated solutions exist and the connectedness of non-dominated solutions is very low; this also implies that PLS is not a very relevant choice for these instances. For more details on the connectedness of non-dominated solutions, that is the ability for a local search algorithm to find additional non-dominated solutions starting from any of them, we refer to Naccache (1978), Ehrgott (2000) or Paquete and Stützle (2009a).

Statistical Analysis

We first analyze the results by means of statistical tools. We present boxplots of the hypervolume for three different problems and four instances of each size in Figure 6.1. The results are rather similar for the two instance sizes and the different problems. There is a high variability, but $conf_{hand}$ often obtains the worst results among the three configurations and it is never clearly better than the others.

To assess whether the performance differences among the configurations are significant, we perform a statistical test on the overall results. Table 6.3 presents the mean and the standard deviation of the hypervolume for each problem and each configuration, for instances of size `50x20`. We perform a paired Wilcoxon signed-rank test with the null hypothesis of equal performance and a confidence level of 0.99 between the $conf_{hand}$ configuration and each of the other two. An entry in bold face indicates that the difference is statistically significant in favor of one of the automatically derived configurations, while an entry in italic font indicates that the difference is statistically significant in favor of $conf_{hand}$. Table 6.4 presents the same results for instances of size `100x20`.

Table 6.2: Candidate configurations obtained from the automatic configuration process, for each instance size and each problem.

		tpls_tratio	ratioInitScal	nb_scal	two_seeds	restart	theta	pls_operator
50x20								
($C_{\max}$,SFT)	$conf_{tun-rnd}$	0.94	8	18	1	0	0.10	exins
	$conf_{tun-ic}$	0.91	8	7	1	0	0.01	exins
($C_{\max}$,TT)	$conf_{tun-rnd}$	0.85	3	9	1	0	0.10	exins
	$conf_{tun-ic}$	0.81	3	9	1	0	0.09	exins
($C_{\max}$, WT)	$conf_{tun-rnd}$	0.81	8	19	1	0	0.18	exins
	$conf_{tun-ic}$	0.81	3	16	1	0	0.49	exins
(SFT, TT)	$conf_{tun-rnd}$	0.91	3	18	1	0	0.12	ins
	$conf_{tun-ic}$	0.9	3	25	1	0	0.01	ins
(SFT, WT)	$conf_{tun-rnd}$	0.90	4	17	1	0	0.07	exins
	$conf_{tun-ic}$	0.87	3	21	1	0	0.05	exins
100x20								
($C_{\max}$,SFT)	$conf_{tun-rnd}$	0.64	8	1	1	0	0.41	ins
	$conf_{tun-ic}$	0.58	4	0	0	1	0.12	exins
($C_{\max}$,TT)	$conf_{tun-rnd}$	0.59	4	14	1	0	0.00	exins
	$conf_{tun-ic}$	0.49	2	7	1	0	0.01	exins
($C_{\max}$, WT)	$conf_{tun-rnd}$	0.39	4	22	1	0	0.02	ins
	$conf_{tun-ic}$	0.43	3	10	1	0	0.07	ins
(SFT, TT)	$conf_{tun-rnd}$	0.95	1.5	8	1	0	0.09	exins
	$conf_{tun-ic}$	0.96	1	3	1	0	0.06	exins
(SFT, WT)	$conf_{tun-rnd}$	0.98	1.5	8	1	0	0.08	ex
	$conf_{tun-ic}$	0.94	3	11	1	0	0.03	exins

In all cases except one (*PFSP*-(*SFT*, *WT*) on Table 6.4), $conf_{hand}$ obtains the worst results, the difference being often statistically significant. In particular, $conf_{tun-rnd}$ improves in nine out of the ten cases significantly over $conf_{hand}$ (see Tables 6.3 and 6.4). For the only case where $conf_{hand}$ appears to be better than $conf_{tun-rnd}$ (*PFSP*-(*SFT*, *WT*) on Table 6.4), one can observe that using $conf_{hand}$ as an initial candidate ensures that we reach at least the same final quality, as shown by the result of $conf_{tun-ic}$. Even if the absolute differences in hypervolume are not very large, this is a remarkable result given that the hybrid TP+PLS using the $conf_{hand}$ configuration proved to be a new state-of-the-art algorithm.

Table 6.3: Mean and standard deviation of the normalized hypervolume for each configuration, evaluated over 10 runs and 100 instances of size 50x20. A bold face indicates that there is a statistically significant difference in favor of a given configuration versus $conf_{hand}$.

	$conf_{hand}$		$conf_{tun-rnd}$		$conf_{tun-ic}$	
	mean	sd	mean	sd	mean	sd
($C_{\max}$,SFT)	0.974	0.043	**0.99**	0.043	**0.989**	0.043
($C_{\max}$,TT)	0.996	0.032	**1.002**	0.031	**1.002**	0.031
($C_{\max}$, WT)	1.033	0.03	**1.04**	0.027	**1.039**	0.028
(SFT, TT)	0.983	0.04	**0.985**	0.041	**0.988**	0.042
(SFT, WT)	1.028	0.031	**1.034**	0.032	**1.032**	0.032

Table 6.4: Mean and standard deviation of the normalized hypervolume for each configuration, evaluated over 10 runs and 100 instances of size 100x20. A bold face indicates that there is a statistically significant difference in favor of a given configuration versus $conf_{hand}$, and an italic face that the difference is in favor of $conf_{hand}$.

	$conf_{hand}$		$conf_{tun-rnd}$		$conf_{tun-ic}$	
	mean	sd	mean	sd	mean	sd
($C_{\max}$,SFT)	0.921	0.056	**0.959**	0.053	**0.953**	0.055
($C_{\max}$,TT)	0.99	0.049	**0.998**	0.047	**1.001**	0.047
($C_{\max}$, WT)	1.007	0.039	**1.022**	0.039	**1.019**	0.039
(SFT, TT)	0.563	0.137	**0.727**	0.144	**0.714**	0.146
(SFT, WT)	0.939	0.038	*0.915*	0.05	0.94	0.046

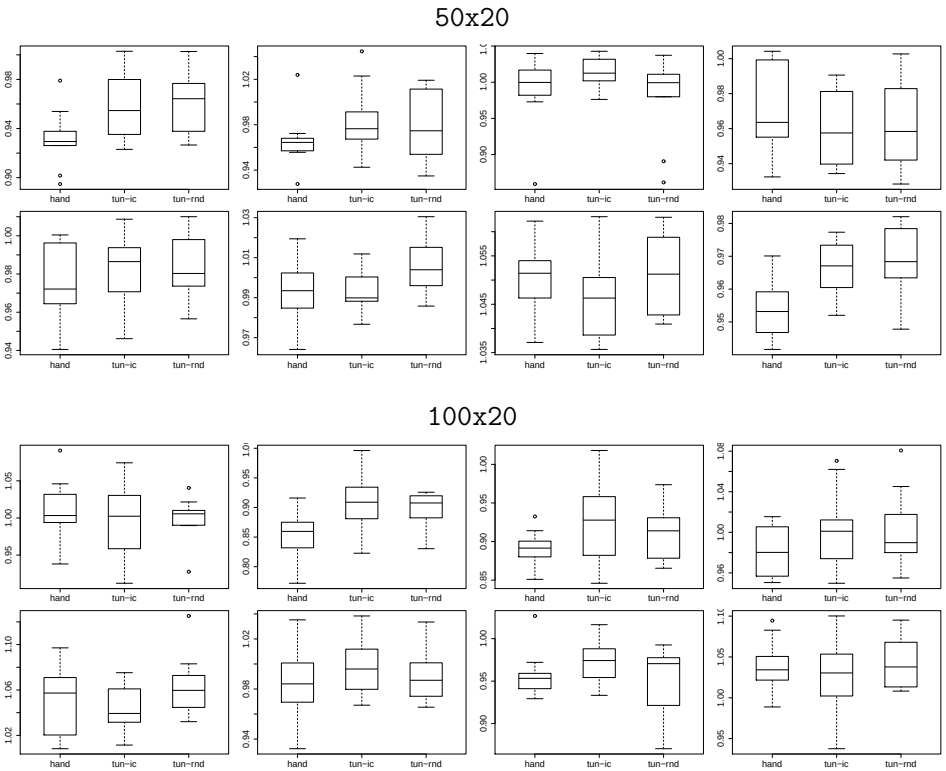

Figure 6.1: Boxplots of the hypervolume across 10 runs, for 4 instances of each size (the same instance being on the same column). The problems are *PFSP*-($C_{\max}$,SFT) (first row) and *PFSP*-($C_{\max}$,TT) (second row). On each boxplot $conf_{hand}$ is on the left, $conf_{tun-ic}$ in the middle and $conf_{tun-rnd}$ on the right.

Graphical Analysis

To explore graphically the performance of each configuration, we examine their empirical attainment functions (EAF, see Section 2.6.2).

Figure 6.2 presents the differences of the EAFs for $conf_{hand}$ and $conf_{tun-rnd}$ for four instances of size 50x20. Other instances and objectives show the same trend for all automatically derived configurations: each algorithm performs better in different regions, but one can hardly assess that one outperforms the other across the whole non-dominated front.

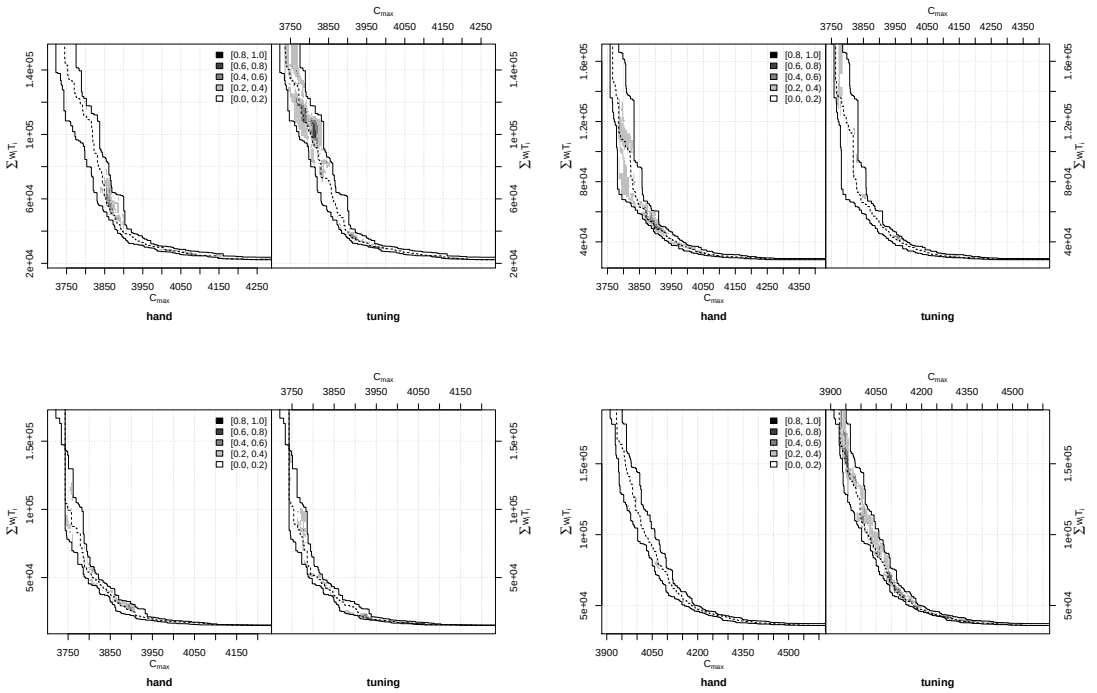

Figure 6.2: Differences in EAF estimated over 10 runs for $conf_{hand}$ and $conf_{tun-rnd}$ for 4 instances of size 50x20. The problem is *PFSP*-($C_{\max}$, *WT*).

Summary

In this section, we demonstrated the impact on the algorithm design of automatic configuration methods. It does not only save a lot of time for the algorithm designer, relieving humans from the tedious and repetitive task that is a manual setting, but it also can improve the results quality beyond a manual setting. In this case study, it leads to improvements of the quality of the results even beyond the state of the art.

6.3 Automatic Configuration for Final Quality: Permutation Flowshop Scheduling Problem

In this section, we perform a study similar to the previous one, but on the bi-objective Traveling Salesman Problem (see Section 2.2). In this thesis, we did not push towards

getting state-of-the-art results for the bTSP as we did for the PFSP. The reason is that the state-of-the-art algorithm for the bTSP was already demonstrated to be a combination of scalarization-based and dominance-based search algorithms (Lust and Teghem, 2010b) and it was not our primary goal to improve over this algorithm (maybe only very marginally) in a time consuming manual tuning process. The authors report results obtained by running PLS until completion, which takes a variable time for each instance and makes it difficult for future comparisons. When comparing their results with ours (with a similar parametrization), we expect that differences would mainly be due to implementation details but no other fundamental differences. Therefore, we do not have a configuration that was obtained by hand after a set of exhaustive preliminary studies as we did for the bPFSP in Chapter 4.

We assume that without preliminary knowledge on a specific problem, testing a candidate configuration that was shown to be high-performing for a different problem is a reasonable start. This is actually what is done in numerous research studies where "default" parameters of well-known algorithms are used. Hence, we use the configuration that was found previously for the bPFSP (see Section 6.2), and we call it $conf_{hand}^{TSP}$.

6.3.1 Experimental Setup

In this section, we detail the framework that is automatically configured, and we detail the experimental setup for the tuning and the evaluation process.

Framework To Be Tuned

The framework that we automatically configure is made of AA-TPLS (developed in Chapter 3) and the original PLS algorithm. Each scalarization is tackled by using an Iterated Local Search algorithm (ILS) based on 3-opt moves (Hoos and Stützle, 2005) [1], and the PLS algorithm makes use of 2-opt moves (using the standard delta-evaluation that can be used for the TSP). We also implemented speed-up techniques for applying PLS to the TSP, proposed in Lust and Jaszkiewicz (2010). They are based on computing candidate sets to reduce the number of neighboring solutions to be explored. These

[1] This algorithm is available online at `http://www.sls-book.net/`

Table 6.5: Candidate configurations compared for the bTSP. Note that the parameter `theta` is not relevant when only one scalarization is performed for each couple of solution (that is if `two_seeds` is equal to 0 – false). There is no use of candidate lists for PLS (thus the value "FULL").

	`tpls_tratio`	`single_algo`	`ratioInitScal`	`nb_scal`	`two_seeds`	`restart`	`theta`	`cand_list_type`
$conf_{hand}^{TSP}$	0.5	ILS	1.5	12	1	0	0.25	FULL
$conf_{tun}^{TSP}$	0.9808	ILS	3	93	0	0	-	FULL

candidate sets are computed based on a Pareto ranking of edges (which excludes edges unlikely to be interesting) or nearest-neighbors (to limit the number of vertices that can be reached from a given vertex); for more information on these techniques we refer to Lust and Jaszkiewicz (2010). The choice of the speed-up technique (as well as not using any of them) is then determined by the parameter `candListType`. If a candidate list is used, the parameter `candListSizeParam` specifies its length (relatively to the instance size).

Automatic Configuration Setup and Instances

The budget used to configure the framework was 2000 experiments, with a computation time limit of 100 seconds for each. We generated isometric, Euclidean instances of size 1000 obtained by drawing random points in a square of side 10^5. For the tuning we used 500 instances, and a separate set of 3 instances for the evaluation of the winning configurations. We use static bounds (0 and 141421357) to normalize the objective values to the range $[1, 2]$, and we compute the hypervolume using $(2.1, 2.1)$ as reference point. Note that 141421357 is the longest possible distance in a square of side 10^5 (on the diagonale), multiplied by the instance size.

Experimental Evaluation

The two candidates that are evaluated, obtained with ($conf_{tun}^{TSP}$) and without ($conf_{hand}^{TSP}$) automatic configuration, are shown in Table 6.5.

Table 6.6: Mean and standard deviation of the normalized hypervolume obtained after 100 seconds for the configurations obtained by hand ($conf_{hand}^{TSP}$) and using an automatic configuration process ($conf_{tun}^{TSP}$), evaluated over 25 runs and three instances of size 1000. The results obtained by $conf_{tun}^{TSP}$ are significantly better than those obtained by $conf_{hand}^{TSP}$ according to a paired Wilcoxon signed-rank test.

$conf_{hand}^{TSP}$		$conf_{tun}^{TSP}$	
mean	sd	mean	sd
1.1626	1.48e-4	**1.1655**	**9.41e-5**

Each candidate is run 25 independent times on three instances (not included in the set of training instances). The hypervolume is computed in the same way as for the tuning process.

6.3.2 Statistical Analysis of the Results

We present a statistical analysis of the results in Table 6.6. The configuration obtained after the automatic configuration process obtains better results than the one using a manual "default" setting. The variance in the case of the TSP is very low (both over runs and instances) and the difference observed is significant according to a paired Wilcoxon signed-rank test, with a p-value lower than $5e-14$. This confirms the results obtained previously on the bPFSP, and demonstrates again (i) the feasibility of multi-objective automatic configuration by the use of an unary indicator as the value to be considered by the tuner, and (ii) the improvements that can be expected in terms of quality of the results.

Next, we go further and consider the tuning of multi-objective algorithms taking into account the *anytime behavior* instead of the final quality.

6.4 Automatic Configuration for Anytime Behavior

In this section we focus on tuning algorithms for the anytime behavior (see Section 2.9), instead of the final results quality (as it is commonly done in the literature). This is in coherence with the goals of this PhD thesis. There are very few works that considered the combination of three aspects: (i) multi-objective algorithms, (ii) anytime behavior and (iii) automatic configuration. The only study we are aware of in this direction was proposed in Radulescu *et al.* (2013), and it focuses on evolutionary algorithms.

Here we want to explore the improvement that can be obtained in terms of anytime behavior when using the TP+PLS framework. In particular, we apply this framework to the Traveling Salesman Problem (see Section 2.2), which is particularly challenging in terms of anytime behavior due to the very high number of non-dominated solutions that must be considered in the dominance-based phase. We use again the `irace` software to set the parameter values.

In what follows, we first present the experimental setup that we used, then we present and discuss the results obtained.

6.4.1 Experimental Setup

Framework To Be Tuned

We use a combination of the AA-TPLS (see Chapter 3) and the generalized PLS algorithm (see Chapter 5). These two algorithms provide the high-level framework, to which TSP specific components are plugged to (i) optimize single-objective problems in the scalarization-based phase and (ii) explore the neighborhoods of solutions in the dominance-based phase. For the scalarization-based phase, we used two different algorithms, whose choice is left as a parameter. The first one is the same Iterated Local Search algorithm (ILS) algorithm as used previously in this chapter, and the second one is an effective implementation of the Lin-Kernighan heuristic (LKH, Helsgaun (2000)), version 2.0.3. The TPLS and PLS framework are implemented in `C++`, and the ILS and LKH algorithms are implemented in `C`. Note that these two algorithms have been optimized to handle multiple calls without reserving new memory each time; all compo-

nents are compiled together in a single executable. Every second, the current results are recorded, which allows to measure the quality (in terms of hypervolume) obtained over time during the search.

All experiments were run on a single core of AMD Opteron 6272 CPUs, running at 2.1 Ghz with a 16 MB cache under Cluster Rocks Linux version 6/CentOS 6.3, 64bits.

The different parameters of the overall algorithms are presented in Table 6.7. For each parameter is given its type, and the possible values either as bounds for numerical parameters or as a set of discrete values for categorical ones. The constraint of a parameter defines when this parameter is enabled (if the constraint is fulfilled) or not. The table also indicates the relevant chapters or publications for more information on a specific parameter.

Instances

We use the same isometric, euclidean instances as for the previous section for the tuning and the evaluation. The tuning set is made of 500 instances and the (disjoint) evaluation set is made of 3 instances.

Automatic Configuration Setup

We used a budget for the configuration process of 2000 experiments, and we allowed 100 seconds for each of them. The tuning tool (`irace`) was run with default parameters.

Two different tuning setups were used, differing only by the goal that `irace` was trying to achieve. The first "classical" goal is to obtain the best possible quality for the final results, as measured by the hypervolume. The second goal is to achieve the best possible anytime behavior. To measure the anytime behavior we sum the hypervolume obtained after each second, and the objective is then to maximize that sum. For each of these two different setups we performed three independent runs of the tuner.

The normalization that we use is the same as in the previous section: we use static bounds (0 and 141421357) to normalize the objective values to the range $[1, 2]$, and we then compute the hypervolume using $(2.1, 2.1)$ as reference point.

Parameter	Type	Bounds or Values	Constraints	Comments
		Hybridization of paradigms together		
tplsTime	r	(0.1, 1)	-	Time allocated to TPLS (relative)
		AA-TPLS		
nb_scal	i	(2,50)	-	Number of scalarizations
ratioInitScal	i	(1, 3)	-	Additional time for initial scal.
two_seeds	c	{1 solution, 2 solutions}	-	see Chapter 3, p. 71
restart	c	{true, false}	two_seeds == 1 seed	see Chapter 3, p. 103
theta	r	(0,0.5)	two_seeds == 2 seeds	see Chapter 3, p. 73
		TSP specific		
singleAlgo	c	{ILS, LKH}	-	Hoos and Stützle (2005); Helsgaun (2000)
		Anytime PLS		
archiving	c	{true, false}	-	see Chapter 5 p. 139
candListType	c	{FULL, NN, PR}	-	see Lust and Jaszkiewicz (2010)
candListSizeParam	r	(0.02, 0.5)	candListType $\in$ (NN,PR)	Lust and Jaszkiewicz (2010)
dynamicArchiving	c	{0, 1}	-	see Chapter 5, p. 141
epsilon	r	(1.0000005, 1.1)	dynamicArchiving == 0	see Chapter 5, p. 141
epsilonDecreaseFactor	r	(0.1, 0.99)	dynamicArchiving == 1	see Chapter 5, p. 142
hvEnhancement	c	{0, 1}	dynamicArchiving == 1	see Chapter 5, p. 144
selectionStrategy	c	{⟨RND ⟩, ⟨OHI ⟩}	dynamicArchiving == 0	see Chapter 5, p. 128
statusAcceptance	c	{⟨>⟩, ⟨∤⟩, ⟨>∤⟩}	dynamicArchiving == 0	see Chapter 5, p. 129
numberAcceptance	c	{⟨1⟩, ⟨*⟩, ⟨1*⟩}	dynamicArchiving == 0	see Chapter 5, p. 129

Table 6.7: Parameters used during the configuration process. The first column gives the parameter to be tuned; the second column gives the type of parameter ('c' stands for categorical, 'i' for integer, and 'r' for real-valued); the third column gives the bounds for integer and real parameters or the values for the categorical parameters; the fourth column indicates possible constraints and the last column indicates where the associated parameters have been described. `candListSizeParam` is a parameter that defines the candidate list size relatively to the instance size. For categorical parameters, '0' and '1' stands for `false` and `true`, respectively.

The reason to use static bounds is that it allows algorithms to output directly the hypervolume of the current approximations to the Pareto front. The use of dynamic bounds on the other hand is not feasible, as it would require to record all sets of solutions obtained after every time step by each candidate on a single instance, in order to find the maximum and minimum values for each objective. This in infeasible in the case of TSP, as it would produce dozens of gigabytes of data.

6.4.2 Experimental Evaluation

The winning configurations obtained from the tuning are shown in Table 6.8. In terms of parameter settings, each repetition of the tuners leads to some rather similar configurations for the TPLS part; configurations seem more diverse in the PLS part, where the parameter space is probably harder to explore or more configurations lead to relatively similar performance.

Each of the three winning configurations for the two setups, that is six configurations, have been run on the evaluation instances, using 25 independent runs. The results for each of the three configurations obtained for tuning the anytime behavior, or the final quality, are extremely similar, showing a high consistence of the tuning output in terms of results obtained. The results are also very similar for each instance. For this reason, and for the sake of clarity, in what follows we simply compare the first configuration of each group.

We present the hypervolume attained by each of these two configurations in Table 6.3. The anytime behavior of the configuration obtained from a tuning specifically aiming at this goal is much better than that of the configuration targeting the final quality. This is not surprising on its own, but the margin of this improvement is very significant, demonstrating how much the anytime behavior can be improved as soon as it is taken into account during the design process. If compared to the specific implementation used in Lust and Teghem (2010b), the improvement in terms of anytime behavior would likely be even larger as the implementation of the TPLS part in that paper was a depth-first search, that results in an exploration of restricted parts of the Pareto front, instead of covering quickly different part of the Pareto front.

Table 6.8: Candidate configurations that are obtained from the different runs of the configuration tool, obtained from the taking into account the final quality (FQ) and the anytime behavior (AB). Parameters whose value is '-' are not relevant because they don't apply in the specific configuration.

	AB_1	AB_2	AB_3	FQ_1	FQ_2	FQ_3
`nb_scal`	95	84	100	100	97	100
`tplsTime`	0.45	0.40	0.48	0.97	0.98	0.95
`ratioInitScal`	1	1	1	3	3	3
`two_seeds`	1	1	1	1	1	1
`singleAlgo`	ILS	ILS	ILS	ILS	ILS	ILS
`candListType`	PR	NN	NN	FULL	NN	FULL
`archiving`	0	0	1	0	0	0
`dynamicArchiving`	1	0	0	1	1	1
`restart`	0	-	0	0	0	0
`theta`	-	-	-	-	-	-
`candListSizeParam`	0.156	0.051	0.188	-	0.046	-
`epsilon`	-	1.035	1.008	-	-	-
`epsilonDecreaseFactor`	0.3025	-	-	0.3410	0.7027	0.4725
`hvEnhancement`	0	-	-	0	0	-
`selectionStrategy`	-	1	1	-	-	-
`statusAcceptance`	-	3	2	-	-	-
`numberAcceptance`	-	1	3	-	-	-

We present a comparison of the differences of the EAF (see Section 2.6.2) of these two configurations in 6.4, after four different computation times: 3, 10, 32 and 100 seconds (the later being the maximum time). The difference after 3 and 10 seconds is clearly in favor of the "anytime behavior" configuration, while there is no clear difference visible after 32 and 100 seconds. This graphical illustration of the difference of the quality that can be expected from each configuration confirms that the"anytime" version should be preferred over the "classical" one only aiming at the final quality.

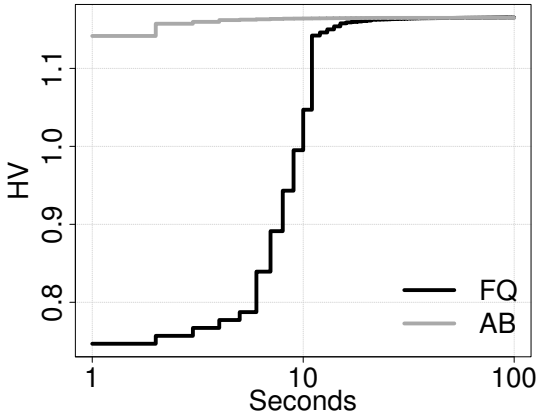

Figure 6.3: Hypervolume of "Final Quality" vs "Anytime Behavior" These plots show the development of the hypervolume over time (averaged over 25 runs) attained by a configuration obtained by tuning the final quality (FQ) and the anytime behavior (AB), on three TSP instances. Confidence intervals are not indicated as the variance is very small across different runs. The results are also very similar across instances.

6.5 Summary

Historically, the design of effective optimization algorithms has involved a significant human effort in order to set the parameters of algorithms, whose proper choice is crucial to reach high performance. This renders the proper engineering of an algorithm a long and work-intensive task; an example was reported in Chapter 5 of this thesis. In recent years, the increasing use of automatic configuration techniques has relieved the algorithm designer from repetitive and time-consuming tasks, which is an important advantage. In this chapter we have demonstrated a second advantage, namely that the quality of the results obtained can be improved by using such techniques.

First, an automatic configuration tool was applied to the framework of Chapter 4, on the same multi-objective algorithms and problem. The multi-objective aspect makes this work a pioneering study as this is the first time we are aware of that a state-of-the-art algorithm for a well-studied problem (bi-objective flowshop variants) was obtained by the use of automatic configuration techniques. This is encouraging as it demonstrates that automatically configuring multi-objective algorithms can be implemented relatively easily by combining existing tuning tools and unary indicators such as the

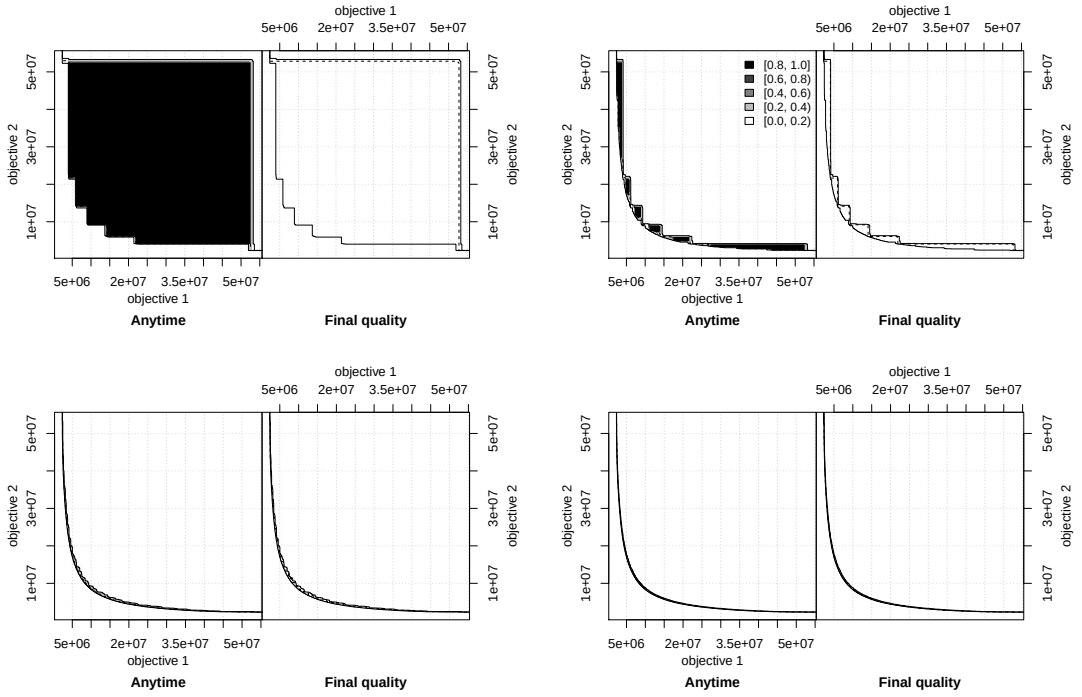

Figure 6.4: Differences in EAF of configurations tuned for "Final Quality" vs "Anytime Behavior" These plots show the differences of EAF between two configurations obtained by tuning for the final quality (right) and the anytime behavior (left), over 25 independent runs, after 3 seconds (top-left), 10 seconds (top-right), 32 seconds (bottom-left) and 100 seconds (bottom-right).

hypervolume. It would also be interesting to compare the results obtained for the bTSP with some recent developments in the field of exact algorithms (For example, Florios and Mavrotas (2014) solved bi-objective instances of size 100 to optimality) to assess the quality of the Pareto front approximations we obtain.

Additionally, even though the bTSP was not studied as thoroughly as the bPFSP in previous chapters, we performed a similar study on this problem, by using the same "manual" configuration as for the PFSP. The results obtained confirm on a second problem that one can achieve, without a priori knowledge, results whose quality matches or even surpasses those obtained after a complex and time-consuming manual setting of parameters.

Second, we went further and applied a tuning tool with the goal of obtaining the best possible anytime behavior. The benefit of doing so over using standard settings was shown to be very large. In the light of this outcome, we believe that a promising direction for future work is to apply the same configuration process to well-known algorithms that may then show new advantages and weaknesses over what is currently known when considering only the final solution quality.

Chapter 7

Conclusion

In this chapter, we conclude this thesis with a summary of its main contributions and a discussion of promising directions for future research.

7.1 Summary of Contributions

7.1.1 Improvement of Stand-Alone Algorithms for Multi-objective Optimization

In this thesis, we designed improved local search algorithms for multi-objective optimization. The first one, Two-phase Local Search (TPLS), was introduced in Paquete and Stützle (2003) and is based on solving scalarized problems. The second one, Pareto Local Search (PLS), uses Pareto dominance in the acceptance test and was introduced in Paquete *et al.* (2004).

We took a broader perspective on the original algorithms by considering them as one of many possible instantiations of more general frameworks. We identified the algorithmic components of these general frameworks, and proposed several variants for each of these components. We designed variants that are elaborations of the original ones, for instance, by replacing random decisions by decisions based on the current search state. The impact of these new components on the search process, in terms of the quality of the results they produce, was evaluated by use of a careful experimenta-

tion. Insights from the experimental studies were used to refine further the algorithmic variants and to propose additional algorithm components. The adoption of a broader perspective on some successful algorithms in order to propose even better ones, can be seen as a general algorithmic engineering process. This engineering process was proven to be highly successful by the outcome of our experimental analysis.

Improved algorithms based on TPLS (Paquete and Stützle, 2003) were proposed in Chapter 3. TPLS is a simple, rather abstract mechanism based on which we have explored the design of algorithms that follow the scalarization-based search paradigm. We proposed new designs for several algorithmic components of TPLS considering two aspects. The first aspect is inspired from a mechanism originally developed for exact algorithms, whose aim is to adapt the search to the shape of the Pareto front. This idea is equally valuable in the heuristic case except that the goal becomes to adapt to the current approximation of the Pareto front known to the algorithm. Wed have shown that its application outperforms simpler deterministic strategies that were used before (Dubois-Lacoste, 2009a; Lust and Teghem, 2010b; Dubois-Lacoste *et al.*, 2010c, 2011a). The second main aspect is the selection of the region of the objective space (more precisely, the selection of a pair of solutions) that is the most promising to explore at a given time. This idea, specific to the optimization of problems using heuristic algorithms, has been demonstrated to be very successful (Dubois-Lacoste *et al.*, 2010c, 2011a,b). We have shown that these two ideas are relevant both in terms of the quality of final results and in terms of the anytime behavior, that is, a better trade-off between the quality of the results obtained and computation time.

Improved algorithms based on PLS (Paquete *et al.*, 2004) were developed in Chapter 5. PLS, unlike TPLS, has a natural stopping criterion when its entire archive of solutions has been fully explored. Therefore, improving the final quality requires to change the exploration capability of PLS, which also increases the time required to reach completion. However, there was room for improvement w.r.t. the results quality reached during the search process before completion, that is, its anytime behavior. We proposed several ways to improve PLS in this aspect, following two different directions. The first direction was to use alternative algorithmic components, following a similar approach as what Chapter 3 did for TPLS. The second direction consists in discretizing the objective space, and allowing only one solution in each region. This direction is derived from the observation that the poor anytime behavior of PLS is due to the high number

of non-dominated solutions it must deal with. An adaptive refinement of the discretization allows to keep the number of non-dominated solutions in the archive relatively low while ensuring that solutions are well-spread in the objective space.

7.1.2 Combination of Two-Phase Local Search and Pareto Local Search for Multi-objective Optimization

In Chapter 2, we have reviewed the relevant literature of heuristic multi-objective optimization, in particular, papers that combine a scalarization-based and a dominance-based search approach. Often, this hybridization is not reported (or even done) explicitly, but it results from attempts to reach high performance. We believe that a clear view of the scalarization-based and the dominance-based paradigms provides better opportunities to design optimization algorithms that combine them as efficiently as possible, a view that we wanted to stress in our literature review.

In Chapter 4, we explored the combination of the two search paradigms, using TPLS and PLS. We did so in a sequential manner, first using TPLS to find a relatively small subset of high-quality solutions that are well spread over the objective space. This high-quality set of non-dominated solutions is then used to seed the PLS algorithm, whose aim is to progress further towards the Pareto front, filling the gaps between the initial solutions.

The sequential hybrid algorithm that we designed in this thesis was applied to several permutation flowshop scheduling problems. The different problems are obtained from tackling different combinations of objectives, resulting in five different bi-objective versions of the flowshop problem. We experimentally evaluated the hybrid algorithm, and compared it to previous state-of-the-art algorithms that were carefully reimplemented. Our algorithm outperformed these competitors by such a wide margin that the use of a quality indicator, usually required to compare non-dominated sets of solutions, was unnecessary. This strong results gives increased evidence of the effectiveness of algorithms that are designed by combining components that are scalarization-based and dominance-based.

7.1.3 Anytime Behavior of Multi-objective Algorithms

The anytime behavior is rarely considered in the literature on heuristic optimization, which has historically been limited to the final solution quality obtained by algorithms after a fixed computation time. We believe that, in the future the anytime behavior of algorithms will be more studied as its importance will be more recognized than it is today. The underlying reason is its high relevance in practice, where in many situations the computation time available is unknown a priori and results may be needed at potentially any time.

We believe the main reason explaining the under-representation of the anytime behavior in the research literature, is the increased difficulty of comparing algorithms performance in an empirical study[1]. We believe that an increased attention will be givenx in the future to the anytime behavior, and this thesis constitutes one step in that direction.

7.1.4 Automatic Configuration of Algorithms

The use of automatic configuration tools has increased in the last years and this trend is likely to continue in the future. The contributions of this thesis to push further this trend are three-fold. The first is the `irace` software package, presented in the appendix, which has been used in numerous research papers by now; the second is the empirical demonstration of the advantages of automatic configuration techniques, which we have used to obtain high-quality results; the third is the demonstrated advantages of applying automatic configuration to multi-objective algorithms.

Chapter 6 was devoted entirely to this topic, it explained how to apply existing tools (and in particular, the `irace` software) to automatically configure multi-objective algorithms.

[1] In some sense, in terms of experimental evaluation, evaluating the anytime behavior of an algorithm constitutes a certain overhead w.r.t evaluating only the final quality. This is the same overhead required for the evaluation of multi-objective algorithms w.r.t. single-objective ones. This inherent, additional difficulty likely explains why multi-objective optimization has been extensively considered much later than single-objective optimization.

In the first part of Chapter 6 we applied automatic configuration to the hybrid algorithm designed in Chapter 4, which was already shown to reach state-of-the-art performance after a careful manual parameter setting. The automatically generated instantiation of this hybrid algorithm was shown to outperform the parameters set "by hand". We then confirmed the results by following the same approach on the bi-objective Traveling Salesman Problem.

In the second part of Chapter 6, we used the `irace` tool with a different target in mind: the anytime behavior of a multi-objective algorithm. These last experiments can be seen as a combination of all the aspects that were considered throughout this thesis. We have shown that combining automatic configuration with the objective of obtaining a good anytime behavior can substantially improve the anytime behavior, over the one obtained when considering only final quality. Moreover, targeting the anytime behavior instead of the final quality within the `irace` software can be done with a very slight modification. This opens many possibilities of applications to new or existing algorithms.

7.2 Promising Directions for Future Research

7.2.1 Improvement of Stand-Alone Algorithms for Multi-objective Optimization

An interesting direction for improving stand-alone algorithms is to better understand the relationship between the performance of algorithmic components and the characteristics of the problem at hand. This is a difficult aspect during the design of heuristic algorithms, where advances can be made in terms of performance regardless of a deep understanding of the underlying reasons. This difficulty is further increased when considering multi-objective algorithms. The algorithms that we use in this thesis have many different components, and a broader understanding of the relation between their effectiveness and the problem characteristics would be very valuable. This could, for instance, allows to design specialized frameworks for specific classes of problems.

7.2.2 Combination of Search Paradigms for Multi-objective Optimization

The hybrid algorithms developed in this thesis are *sequential* in the sense that they consist of two phases that are executed one after the other. Careful manual settings based on preliminary experiments, or better, an automatic configuration tool can determine empirically the best moment to switch from one phase to the other. In the algorithms developed in this thesis, we switch from a search algorithm that is scalarization-based to an algorithm that is dominance-based. The proper moment is determined by taking (implicitly as this is done through preliminary experiments or automatic configuration) into consideration the search state of the whole archive of non-dominated solutions. We think that higher performance could be achieved by allowing a hybrid algorithm to adapt its behavior in a more refined way: to adapt to the local characteristics of a search space and the current approximation to the Pareto front attained in a specific region. In the case of TPLS and PLS, a simple example of such a refined behavior would be to switch from TPLS to PLS at different times for different regions of the objective space. This is a challenging, but promising research direction. It would involve a number of algorithmic challenges, and the need for some ways to "judge" the current search state in different regions in order to take appropriate decisions. Following this direction and going one step further, one can imagine that the best decision at a given time could be to switch from PLS back to TPLS, effectively transforming the sequential hybrid algorithm into an iterative hybrid algorithm, able to freely change the search paradigm over time and regions.

Another interesting direction for future research is the study of the challenges involved in using the algorithms developed in this thesis for a higher number of objectives. It would not only increase the computational complexity required to perform algorithmic tasks but it would also require to design algorithms that could be very different from their two-objective counterparts.

7.2.3 Anytime Behavior of Multi-objective Algorithms

The fact that the hybrid algorithms developed in this thesis are sequential, leaves margins for improvement in terms of the final quality as we just explained. Moreover, this

is also true in terms of the anytime behavior, because it is difficult to estimate the proper time at which to switch from one search paradigm to the other when the remaining computation time is unknown. Iterative hybridization, and in particular an iterative hybrid algorithm based on the new anytime TPLS and PLS presented in this thesis, would be a promising direction to further improve the anytime behavior.

In general, we do believe that the anytime behavior will play a larger role in the research community, as there seems to be a gap currently between its relevance in practice and the relatively poor recognition it receives in the design of effective heuristic algorithms. Many algorithm designers are actually aware of its relevance, even if not consciously: we design constructive heuristics that deliver quickly reasonable results, and more complex algorithms that deliver better results but require more time. The co-existence and relevance of the two categories is known and obvious to many. What is thus understood, but often implicitly, is that they are both equally relevant as they just offer different trade-offs in terms of anytime behavior.

7.2.4 Automatic Configuration of Algorithms

In this thesis, the usage of an unary indicator allowed us to configure multi-objective algorithms using an existing tool (the `irace` software) designed for single-objective algorithms. An interesting direction for future research on the automatic configuration of algorithms is the design of automatic configuration tools that can inherently handle multi-objective algorithms based on Pareto dominance instead of (or in addition to) unary quality indicators. This would likely require some significant changes in the way the configuration tool works internally, as many candidates would be mutually non-dominated. In the case of `irace`, for instance, the statistical tests that are used to discard inferior candidates could not be used anymore. Two directions could be followed here: the adaptation of existing tools to multi-objective algorithms or the design of entirely new specialized configuration tools. We believe both directions are very promising to explore.

In general, the use of automatic configuration techniques to develop new state-of-the-art algorithms will likely increase. The wide-spread use of such tools to design efficient algorithms will not only save human effort by doing repetitive tasks, but also

allow human designers to focus on what they do best: being creative, designing new components, and finding patterns that indicate which algorithms work best for which problems. In many fields automatic configuration techniques have the potential to create designs that were never tested or even thought of before, and therefore open whole new directions to explore towards even better performance.

Appendix: The IRACE Software Package

This appendix has been published as a technical report (López-Ibáñez *et al.*, 2011a).

1 Introduction

The `irace` package implements the *iterated racing* procedure for automatic algorithm configuration. Iterated racing is a generalization of the Iterated F-race (I/F-Race) procedure proposed by Balaprakash *et al.* (2007) and further developed by Birattari *et al.* (2010). `irace` is implemented as an R package (R Development Core Team, 2008) and it builds upon the **race** package by Birattari (2003).

In the context of optimization, algorithm configuration is the process of finding the most appropriate parameter settings of an optimization algorithm for a particular problem. Many optimization algorithms have a large number of parameters that need to be specified. This is particularly true for general-purpose solvers, such as CPLEX, and metaheuristics (Gendreau and Potvin, 2010; Hoos and Stützle, 2005), such as evolutionary algorithms (Goldberg, 1989). When tackling a particular problem, for example, the daily routing of delivery trucks, an appropriate setting of algorithmic parameters may lead to higher-performing optimization algorithm. In this example, the solver tackles a different instance of the routing problem every day. However, one can expect that finding good parameter settings for past problem instances will result in a high-performing algorithm for future problem instances.

Automatic algorithm configuration (also known as, *offline parameter tuning*) can be described from a machine learning perspective as the problem of finding good param-

eter settings for solving unseen problem instances by learning on a set of training problem instances (Birattari, 2009). Thus, there are two clearly delimited phases. In a primary tuning phase, an algorithm configuration is chosen, given a set of tuning instances representative of a particular problem. In a secondary production (or testing) phase, the chosen algorithm configuration is used to solve unseen instances of the same problem. The goal is to find, during the tuning phase, an algorithm configuration that minimizes some cost measure over the set of instances that will be seen during the production phase. In other words, the ultimate purpose is that the high-quality configuration of the algorithm found during the tuning phase generalizes to similar but unseen instances.

Algorithm configuration is still nowadays frequently performed in an ad-hoc fashion by algorithm designers, who test a limited number of algorithm configurations in a few benchmark instances. By contrast, automatic methods for algorithm configuration involve the use of experimental design techniques (Coy *et al.*, 2001; Adenso-Díaz and Laguna, 2006), evolutionary algorithms (Nannen and Eiben, 2006; Ansótegui *et al.*, 2009), local search (Hutter *et al.*, 2009b), or statistical models (Hutter *et al.*, 2011; Bartz-Beielstein, 2006).

A notable example is sequential parameter optimization (SPO) (Bartz-Beielstein, 2006), and the associated **SPOT** package (Bartz-Beielstein *et al.*, 2010b, 2011). **SPOT** uses statistical models for finding optimal parameters of optimization algorithms. The main differences between **SPOT** and `irace` are that the former is more oriented towards analyzing the impact and interactions of parameters when applying an algorithm to a single instance or function, whereas the goal of `irace` is to find parameter configurations that perform well over a large set of heterogeneous instances.

Other automatic configuration methods have been used in the literature to configure new state-of-the-art algorithms. For example, FocusedILS (Hutter *et al.*, 2009b) was used to automatically configure a highly parameterized tree search (Hutter *et al.*, 2007), and a framework of SAT solvers (KhudaBukhsh *et al.*, 2009) that won several prizes in the International SAT competition; as well as tuning the commercial mixed-integer programming solver CPLEX, and obtaining a significant speedup over the default settings (Hutter *et al.*, 2010).

In the context of machine learning, a similar problem is hyperparameter tuning, where the goal is to find the most appropriate parameter settings of a machine learning

model (called hyperparameters) for a specific data set. In this case, there is a cost function, such as the prediction rate, which must be optimized. An instance of the problem is a partition of a particular data set into a training set and a validation set. The goal is to find good hyperparameters of a model for predicting unseen data. The traditional method for hyperparameter tuning is based on exhaustive (grid) search. Exhaustive search becomes impractical if the hyperparameter space is large or the evaluation of the cost function is expensive. Recently, random search has been shown to perform better than grid search for hyperparameter tuning (Bergstra and Bengio, 2012). Nonetheless, automatic configuration methods are very well suited for this task, given the similarity of hyper-parameter tuning to algorithm configuration.

The `irace` package builds upon previous methods for automatic algorithm configuration. Birattari *et al.* (2002); Birattari (2004, 2009) proposed an automatic configuration approach, F-Race, based on *racing* (Maron and Moore, 1997) and Friedman's non-parametric two-way analysis of variance by ranks. This proposal was later improved by sampling configurations from the parameter space, and refining the sampling distribution by means of repeated applications of F-Race. The resulting automatic configuration approach was called Iterated F-race (I/F-Race) (Balaprakash *et al.*, 2007; Birattari *et al.*, 2010). Although a formal description of the I/F-Race procedure is given in the original publications, no implementation of it has been made publicly available. The `irace` package implements a general *iterated racing* procedure, which includes I/F-Race as a special case. It also implements several extensions already described by Birattari (2004, 2009), such as the use of the paired t-test instead of Friedman's test. Finally, `irace` incorporates several improvements never published before, such as sampling from a truncated normal distribution, a parallel implementation, and a restart strategy that avoids premature convergence.

The appendix is structured as follows. Section 2 introduces the algorithm configuration problem. Section 3 describes the iterated racing procedure as implemented in the `irace` package. Section 5 describes the `irace` package itself, its components and options. Section 6 describes in detail how to apply `irace` to two simple scenarios: the tuning of the hyperparameters of a neural network classifier, and the configuration of the parameters of a metaheuristic optimization algorithm.

2 Automatic Configuration

2.1 Configurable Algorithms

Many algorithms for computationally hard optimization problems are configurable, that is, they have a number of parameters that may be set by the user. As an example, evolutionary algorithms (EAs) (Goldberg, 1989) often require the user to specify settings like the mutation rate, the recombination operator and the population size. Another example is CPLEX, a mixed-integer programming solver, that has more than 76 configurable parameters affecting the main algorithm used internally by CPLEX, e.g., one can select among different branching strategies.

The reason these parameters are configurable is that there is no single optimal setting for every possible application of the algorithm, and, in fact, the optimal setting of these parameters depends on the problem being tackled (Adenso-Díaz and Laguna, 2006; Birattari, 2009).

There are two main classes of parameters: *categorical* and *numerical* parameters. Categorical parameters represent discrete values without any implicit order or sensible distance measure. An example is the different recombination operators in EAs. Numerical parameters have an implicit order of their values. Examples are the population size and the mutation rate in EAs. There are also seemingly categorical parameters but with an implicit order of their values. An example would be a parameter with three values {`low`, `medium`, `high`}. Such parameters are called *ordinal*, and we handle them as numerical parameters. Finally, parameters may be *subordinate* to other parameters, that is, they are only relevant for particular values of other parameters. For example, in a genetic algorithm there may be a parameter that defines the selection operator. More concretely, the selection operator could take the values `roulette_wheel` or `tournament`. The value `roulette_wheel` does not have any specific additional parameters, whereas the value `tournament` requires to specify the value of parameter "tournament size". In this case, the parameter "tournament size" is subordinate to the fact that the selection operator takes the value `tournament`. Subordinate parameters are not the same as constraints on the values of parameters. For example, given parameters a and b, a constraint may be that $a < b$. Such constraints limit the range of values that a certain parameter can take in

dependence of other parameters, whereas subordinate parameters are either disabled or they have a value according to a predefined range. In some cases, parameter constraints may be modeled by replacing one of the parameters by a surrogate parameter, e.g., $a' \in (0, 1)$, such that $a = a' \cdot b$.

2.2 The Algorithm Configuration Problem

In the following, we briefly introduce the algorithm configuration problem, a formal definition is given by Birattari (2009). Let us assume that we have a parametrized algorithm with N^{param} parameters, X_d, $d = 1, \ldots, N^{\text{param}}$, and each of them may take different values (settings). A configuration of the algorithm $\theta = \{x_1, \ldots, x_{N^{\text{param}}}\}$ is a unique assignment of values to parameters, and Θ denotes the possibly infinite set of all configurations of the algorithm.

When considering a problem to be solved by this parametrized algorithm, the set of possible instances of the problem may be seen as a random variable $\mathcal{I}$ from which instances to be solved are sampled. We are also given a cost measure $\mathcal{C}(\theta, i)$ that assigns a value to each configuration when applied to a single problem instance i, which is a realization of $\mathcal{I}$. Since the algorithm may be stochastic, this cost measure is often a random variable and the value $c(\theta, i)$ is a realization of the random variable $\mathcal{C}(\theta, i)$. The cost value may be the best objective function value found within a given computation time, or, perhaps, the deviation from the optimum value if the latter is known. In the case of decision problems, it may correspond to the computation time required to reach a decision, possibly bounded by a maximum cut-off time. In any case, the cost measure assigns a cost value to one run of a particular configuration on a particular instance. The criterion that we want to optimize when configuring an algorithm for a problem is a function c_θ of the cost of a configuration θ with respect to the distribution of the random variable $\mathcal{I}$. The goal of automatic configuration is finding the best configuration θ^* that minimizes c_θ.

A usual definition of c_θ is the expected cost of θ. The definition of c_θ determines how to rank the configurations over a set of instances. If the cost values over different instances are incommensurable, the median or the sum of ranks may be more meaningful. The precise value of c_θ is generally unknown, and it can only be estimated by

sampling. This sampling is performed in practice by obtaining realizations $c(\theta, i)$ of the random variable $\mathcal{C}(\theta, i)$. In other words, by evaluating an algorithm configuration on instances sampled from $\mathcal{I}$. Since most algorithms of practical interest are sufficiently complex to preclude an analytical approach, the configuration of such algorithms follows an experimental approach, where each experiment is a run of an implementation of the algorithm under specific experimental conditions (Bartz-Beielstein, 2006).

3 Iterated Racing

3.1 An Overview of Iterated Racing

The `irace` package that we propose in this appendix is an implementation of iterated racing, of which I/F-Race (Balaprakash *et al.*, 2007; Birattari *et al.*, 2010) is a special case that uses the Friedman's non-parametric two-way analysis of variance by ranks.

Iterated racing is a method for automatic configuration that consists of three steps: (1) sampling new configurations according to a particular distribution, (2) selecting the best configurations from the newly sampled ones by means of racing, and (3) updating the sampling distribution in order to bias the sampling towards the best configurations. These three steps are repeated until a termination criterion is met.

In iterated racing, each configurable parameter has an independent sampling distribution, which is either a normal distribution for numerical parameters, or a discrete distribution for categorical parameters. The update of the distributions consists of modifying the sampling distributions, the mean and standard deviation in the case of the normal distribution, or the discrete probability values of the discrete distributions. The update biases the distributions to increase the probability of sampling, in future iterations, the parameter values in the best configurations found.

After new configurations are sampled, the best configurations are selected by means of racing. Racing was first proposed in machine learning to deal with the problem of model selection (Maron and Moore, 1997). Birattari *et al.* (2002) adapted the procedure for the configuration of optimization algorithms. A race starts with a finite set of candidate configurations. At each step of the race, the candidate configurations are evaluated

on a single instance. After each step, those candidate configurations that perform statistically worse than at least another one are discarded, and the race continues with the remaining *surviving* configurations. This procedure continues until reaching a minimum number of surviving configurations, a maximum number of instances that have been used or a pre-defined computational budget. This computational budget may be an overall computation time or a number of experiments, where an experiment is the application of a configuration to an instance.

The next subsection (Section 3.2) gives a complete description of the iterated racing algorithm as implemented in the `irace` package. We mostly follow the description of the original appendixs (Balaprakash *et al.*, 2007; Birattari *et al.*, 2010), adding some details that were not explicitly given there. Later in Section 4, we mention several extensions that were not proposed in the original publications.

3.2 The Iterated Racing Algorithm in the `irace` Package

In this section, we describe the implementation of iterated racing as proposed in the `irace` package. The setup of the `irace` package itself is given in Section 5.

An outline of the iterated racing algorithm is given in Algorithm 1. Iterated racing requires as input: a set of instances I sampled from $\mathcal{I}$, a parameter space (X), a cost function (C), and a tuning budget (B).

Iterated racing requires an estimation of the number of iterations N^{iter} (races) that it will execute. The default setting of N^{iter} depends on the number of parameters with $N^{\text{iter}} = \lfloor 2 + \log_2 N^{\text{param}} \rfloor$. Each iteration performs one race with a limited computation budget $B_j = (B - B_{\text{used}})/(N^{\text{iter}} - j + 1)$, where $j = 1, \ldots, N^{\text{iter}}$. Each race starts from a set of candidate configurations Θ_j. The number of candidate configurations is calculated as $|\Theta_j| = N_j = \lfloor B_j/(\mu + \min(5, j)) \rfloor$. Thus, the number of candidate configurations decreases with the number of iterations, which means that more evaluations per configuration will be performed in later iterations. The parameter μ allows the user to influence the ratio between budget and number of configurations, which also depends on the iteration number j. The idea behind this setting is that configurations generated in later iterations will be more similar, and, hence, more evaluations will be necessary

Require: $I = \{I_1, I_2, \dots\} \sim \mathcal{I}$,
parameter space: X,
cost measure: $\mathcal{C}(\theta, i) \in \mathbb{R}$,
tuning budget: B

$\Theta_1 \sim \mathsf{SampleUniform}(X)$
$\Theta^{\text{elite}} := \mathsf{Race}(\Theta_1,\ B_1)$
$j := 2$
while $B_{\text{used}} \leqslant B$ **do**
  $\Theta^{\text{new}} \sim \mathsf{Sample}(X, \Theta^{\text{elite}})$
  $\Theta_j := \Theta^{\text{new}} \cup \Theta^{\text{elite}}$
  $\Theta^{\text{elite}} := \mathsf{Race}(\Theta_j,\ B_j)$
  $j := j + 1$
end while
Output: Θ^{elite}

Table 1: Algorithm outline of iterated racing.

to identify the best ones. On the other hand, we do not consider for computing this setting more than five iterations, in order to avoid having too few configurations in a single race.

In the first iteration, the initial set of candidate configurations is generated by uniformly sampling the parameter space X. When a race starts, each configuration is evaluated on the first instance by means of the cost measure $\mathcal{C}$. Configurations are iteratively evaluated on subsequent instances until a number of instances have been seen (T^{first}). Then, a statistical test is performed on the results. If there is enough statistical evidence to identify some candidate configurations as performing worse than at least another configuration, the worst configurations are removed from the race, while the others, the *surviving* candidates, are run on the next instance.

There are several alternatives for selecting which configurations should be discarded during the race. The F-Race algorithm (Birattari *et al.*, 2002; Birattari, 2009) relies on the non-parametric Friedman's two-way analysis of variance by ranks, the Friedman test, and its associated post-hoc test, as described by Conover (Conover, 1999). Nonetheless, the **race** package (Birattari, 2003) implements various alternatives based on the paired t-test with and without p-value correction for multiple comparisons, which are also available in the proposed `irace` package.

A new statistical test is performed every T^{each} instances. By default $T^{\text{each}} = 1$, but in some situations it may be helpful to only perform each test after the configurations have been evaluated on a number of instances. The race continues until the budget of the current iteration is not enough to test all remaining candidate configurations on a new instance ($B_j < N_j^{\text{surv}}$), or when at most N^{min} configurations remain, $N_j^{\text{surv}} \leqslant N^{\text{min}}$.

At the end of a race, the surviving configurations are assigned a rank r_z according to the sum of ranks or the mean cost, depending on which statistical test is used during the race. The $N_j^{\text{elite}} = \min(N_j^{\text{surv}}, N^{\text{min}})$ configurations with the lowest rank are selected as the set of elite configurations Θ^{elite}.

In the next iteration, before a race, a number of $N_j^{\text{new}} = N_j - N_{j-1}^{\text{elite}}$ new candidate configurations are generated. For generating a new configuration, first one parent configuration θ_z is sampled from the set of elite configurations Θ^{elite} with a probability:

$$p_z = \frac{N_{j-1}^{\text{elite}} - r_z + 1}{N_{j-1}^{\text{elite}} \cdot (N_{j-1}^{\text{elite}} + 1)/2}, \tag{1}$$

which is proportional to its rank r_z. Hence, higher ranked configurations have a higher probability of being selected as parents.

Next, a new value is sampled for each parameter X_d, $d = 1, \ldots, N^{\text{param}}$, according to a distribution that its associated to each parameter of θ_z. Parameters are considered in the order determined by the dependency graph of conditions, that is, non-subordinate parameters are sampled first, those parameters that are subordinate to them are sampled next if the condition is satisfied, and so on. Moreover, if a subordinate parameter that was disabled in the parent configuration becomes enabled in the new configuration, then the parameter is sampled uniformly, as in the initialization phase.

If X_d is a numerical parameter defined within the range $[\underline{x}_d, \overline{x}_d]$, then a new value is sampled from the truncated normal distribution $\mathcal{N}(x_d^z, \sigma_d^j)$, such that the new value is within the given range.[2] The mean of the distribution x_d^z is the value of parameter d in elite configuration θ_z. The parameter σ_d^j is initially set to $(\overline{x}_d - \underline{x}_d)/2$, and it is decreased

[2]For sampling from a truncated normal distribution, we use the **msm** package (Jackson, 2011).

at each iteration before sampling:

$$\sigma_d^j := \sigma_d^{j-1} \cdot \left(\frac{1}{N_j^{\text{new}}} \right)^{1/N^{\text{param}}} \tag{2}$$

By reducing σ_d^j in this manner at each iteration, the sampled values are increasingly closer to the value of the parent configuration, focusing the search around the best parameter settings found as the iteration counter increases. Roughly speaking, the multi-dimensional volume of the sampling region is reduced by a constant factor at each iteration, but the reduction factor is higher when sampling a larger number of new candidate configurations (N_j^{new}).

If the numerical parameter is of integer type, we round the sampled value to the nearest integer. The sampling is adjusted to avoid the bias against the extremes introduced by rounding after sampling from a truncated distribution.

If X_d is a categorical parameter with levels $X_d \in \{x_1, x_2, \ldots, x_{n_d}\}$, then a new value is sampled from a discrete probability distribution $\mathcal{P}^{j,z}(X_d)$. In the first iteration ($j = 1$), $\mathcal{P}^{1,z}(X_d)$ is uniformly distributed over the domain of X_d. In subsequent iterations, it is updated before sampling as follows:

$$\mathcal{P}^{j,z}(X_d = x_j) := \mathcal{P}^{j-1,z}(X_d = x_j) \cdot \left(1 - \frac{j-1}{N^{\text{iter}}}\right) + \Delta\mathcal{P} \tag{3}$$

where

$$\Delta\mathcal{P} = \begin{cases} \dfrac{j-1}{N^{\text{iter}}} & \text{if } x_j = x_z \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Finally, the new configurations generated after sampling inherit the probability distributions from their parents, and a new race is launched with the union of the new configurations and the elite configurations.

The algorithm stops if the budget is exhausted ($B_{\text{used}} > B$) or if the number of candidate configurations to be evaluated at the start of an iteration is not greater than the number of elites ($N_j \leqslant N_{j-1}^{\text{elite}}$), since in that case no new configurations would be generated. If the iteration counter j reaches the estimated number of iterations N^{iter} but

there is still enough remaining budget to perform a new race, we simply increase N^{iter} and continue the algorithm.

4 Extensions

We have implemented several extensions that were not proposed in the original publications.

4.1 Initial Configurations

We can seed the iterated race procedure with a set of initial configurations. In that case, only enough configurations are sampled to reach N_1 in total.

4.2 Soft-restart

Our implementation incorporates a "soft-restart" mechanism to avoid premature convergence. In the original I/F-Race proposal (Balaprakash *et al.*, 2007), the standard deviation, in the case of numerical parameters, or the discrete probability of unselected parameter settings, in the case of categorical ones, decreases at every iteration. Diversity is introduced by the variability of the sampled configurations. However, if the tuning converges to a few, very similar elite configurations in few iterations, the diversity is lost and newly generated candidate configurations will not be very different from the ones already tested. Such a premature convergence wastes the remaining budget on repeatedly evaluating minor variations of the same configurations, without exploring new alternatives.

We implemented a "soft-restart" mechanism that checks for premature convergence after generating each new set of candidate configurations. We consider that there is premature convergence when the "distance" between two candidate configurations is zero. The distance between two configurations is defined as the maximum distance between their parameter settings, which is defined as follows:

- If the parameter is subordinate and disabled in both configurations, the distance is zero;
- if it is disabled in one configuration but enabled in the other, the distance is one;
- if the parameter is enabled in both configurations (or it is not subordinate), then:
  - in the case of numerical parameters (integral or real), the distance is the absolute normalized difference between their values;
  - in the case of ordinal and categorical parameters, the distance is one if the values are different and zero otherwise.

When premature convergence is detected, a "soft-restart" is applied by partially reinitializing the sampling distribution. This reinitialization is applied only to the elite configurations that were used to generate the candidate configurations with zero distance. The other elite configurations do not suffer from premature convergence, thus they may still lead to new configurations, whereas reinitializing their sampling distribution would mean to lose all the knowledge accumulated on them.

In the case of categorical parameters, the discrete sampling distribution of elite configuration z, $\mathcal{P}^{j,z}(X_d)$, is adjusted by modifying each individual probability value $p \in \mathcal{P}^{j,z}(X_d)$ with respect to the maximum value $p_{\max} = \max\{\mathcal{P}^{j,z}(X_d)\}$ as follows:

$$p := \frac{0.9 \cdot p + 0.1 \cdot p_{\max}}{\sum_{p' \in \mathcal{P}^{j,z}(X_d)} 0.9 \cdot p' + 0.1 \cdot p_{\max}}.$$

For numerical and ordinal parameters, the standard deviation of elite configuration z, $\sigma_d^{j,z}$, is "brought back" two iterations, with a maximum limit of its value in the second iteration, as follows:

$$\sigma_d^{j,z} := \min\left\{\sigma_d^{j,z} / \left(\frac{1}{N_j^{\text{new}}}\right)^{2/N^{\text{param}}}, \frac{\overline{x}_d - \underline{x}_d}{2} \cdot \left(\frac{1}{N_j^{\text{new}}}\right)^{1/N^{\text{param}}}\right\}$$

After adjusting the sampling distribution of all affected elite configurations, the set of candidate configurations that triggered the soft-restart is discarded and a new set of N^{new} configurations is sampled from the elite configurations. This procedure is applied at most once per iteration.

5 The irace Package

We provide here a brief summary of the irace package. The full documentation is available together with the package.

The scheme in Figure 1 describes how the different parts of irace interact with each other. The program irace requires three main inputs:

1. A description of the parameter space X, that is, the parameters to configure, their types, ranges and constraints. Section 5.2 summarizes how to define a parameter space in irace.
2. The set of tuning instances $\{I_1, I_2, \dots\}$, which in practice is a finite, representative sample of $\mathcal{I}$. The particular options for specifying the set of tuning instances are given in Section 5.1.
3. The configuration of irace itself, which is defined by a number of options. Table 2 maps the description of iterated racing in Section 3.2 to the configuration options in irace. The complete list of options is available in the software documentation.

In addition, irace requires a function (or an auxiliary program) called hookRun, which is responsible of applying a particular configuration to an instance and returning the corresponding cost value.

The irace package is designed to be used either from within R, or from the command-line by means of a wrapper. We illustrate these two usage modes by means of simple examples in Section 6.

5.1 Tuning Instances

The set of tuning instances $\{I_1, I_2, \dots\}$ may be given explicitly as a configuration option of irace. Alternatively, the instances may be read from an instance file (instanceFile). The string given by option instanceDir will be prefixed to them. If the option instanceFile is not set, then irace considers all files found in instanceDir, and recursively in its subdirectories, as tuning instances. The order in which instances are con-

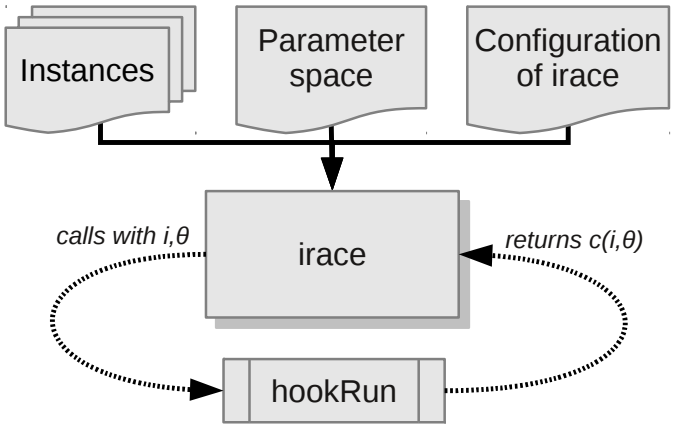

Figure 1: Scheme of `irace` flow of information.

Iterated racing parameter	`irace` configuration option
B	`maxExperiments`
$\mathcal{C}$ (cost measure)	`hookRun`
μ	`mu`
$N^{\min}$	`minNbSurvival`
T^{first}	`firstTest`
T^{each}	`eachTest`
Statistical test	`testType`

Table 2: Configuration options of `irace` corresponding to the description of iterated racing given in Section 3.2. The full list of options is available in the complete documentation.

sidered by `irace` is randomized if the option `sampleInstances` is enabled. Otherwise, the order is the same as given in `instanceFile` if this option is set or in alphabetical order if there is no `instanceFile`.

5.2 Parameter Space

For simplicity, the description of the parameters space is given as a table. Each line of the table defines a configurable parameter:

```
<name> <label> <type> <range> [ | <condition> ]
```

where each field is defined as follows:

`<name>` The name of the parameter as an unquoted alphanumeric string, for instance: '`ants`'.

`<label>` A *label* for this parameter. This is a string that will be passed together with the parameter to `hookRun`. In the default `hookRun` provided with the package (Section 5.3), this is the command-line switch used to pass the value of this parameter, for instance '`"--ants "`'.

`<type>` The type of the parameter, either *integer*, *real*, *ordinal* or *categorical*, given as a single letter: '`i`', '`r`', '`o`' or '`c`'.

`<range>` The range or set of values of the parameter.

`<condition>` An optional *condition* that determines whether the parameter is enabled or disabled, thus making the parameter subordinate. If the condition evaluates to false, then no value is assigned to this parameter, and neither the parameter value nor the corresponding label are passed to `hookRun`. The condition must be a valid R logical expression. The condition may contain the name of other parameters as long as the dependency graph does not contain any cycle. Otherwise, `irace` will detect the cycle and stop with an error.

Parameter types and range. Parameters can be of four types:

- *Real* parameters are numerical parameters that can take any floating-point values within a given range. The range is specified as an interval '`(<lower bound>,<upper bound>)`'. This interval is closed, that is, the parameter value may eventually be one of the bounds. The possible values are rounded to a number of *decimal places* specified by option `digits`. For example, given the default number of digits of 4, the values 0.12345 and 0.12341 are both rounded to 0.1234.
- *Integer* parameters are numerical parameters that can take only integer values within the given range. The range is specified as for real parameters.

- *Categorical* parameters are defined by a set of possible values specified as '(<value 1>, ..., <value n>)'. The values are quoted or unquoted character strings. Empty strings and strings containing commas or spaces must be quoted.
- *Ordinal* parameters are defined by an *ordered* set of possible values in the same format as for categorical parameters. They are handled internally as integer parameters, where the integers correspond to the indexes of the values.

Section 6 shows examples on how to read the parameters table either from a string literal or from a file.

5.3 hookRun

The evaluation of a candidate configuration is done by means of a user-given function or, alternatively, a user-given auxiliary program. The function (or program name) is specified by the option `hookRun`.

When `hookRun` is an R function, then it is invoked for each candidate configuration as:

```
hookRun(instance, candidate, extra.params, config)
```

where `instance` is a single instance, `extra.params` is a user-defined value associated to this instance, `config` is the configuration of `irace`, and `candidate` is a list with three components: *(1)* `$index`, which is a numeric value identifying this candidate; *(2)* `$values`, which is a one-row data frame with one column per parameter value; and *(3)* `$labels`, which is a list of the labels of each parameter. The function `hookRun` must return a numerical value corresponding to the cost measure of the candidate configuration on the given instance.

When `hookRun` is an auxiliary executable program, then it is invoked for each candidate configuration, passing as arguments: the instance, a numeric identifier, and the command-line parameters of the candidate configuration. The numeric identifier uniquely identifies a configuration within a race (but not across the races in a single iterated race). The command line is constructed by appending to each parameter label

(switch), *without separator*, the value of the parameter, following the order given in the parameter table. The program `hookRun` must print (only) a real number, which corresponds to the cost measure of the candidate configuration for the given instance. The working directory of `hookRun` is set to the execution directory specified by the option `execDir`. This allows the user to execute several runs of `irace` in parallel without the runs interfering with each other.

6 Examples of Tuning Scenarios

The next two sections (Sections 6.1 and 6.2) illustrate two different ways of using `irace`. The first example shows how to set up and use the `irace` package by means of R programming. The second example shows how to tune an external program via the command-line options of the `irace` stand-alone program provided with the package.

6.1 Hyperparameter Tuning of Neural Networks

In this simple example, the goal is to tune the hyperparameters of a neural network for a classification task. For this purpose, we use the neural network implementation provided by the **nnet** package, and the **iris** dataset.

```
R> require(nnet)
R> require(datasets)
R> data(iris)
```

We generate samples of 10-fold cross-validations of the **iris** data set:

```
R> k <- 10
R> n <- nrow(iris)
R> sample.n <- sample(n)
R> msk <- k * c(0:(ceiling(n / k) - 1))
```

For simplicity, we only consider two hyperparameters: the number of units in the hidden layer of the network (`size`) and the weight decay (`decay`). The hyperparameters

are specified as described in Section 5.2, and we use the `irace` function `readParameters` to parse their description:

```
R> parameters <- readParameters(text = '
+  size  "" i (1, 50)
+  decay "" r (1e-4, 1e-1)
+  ')
```

Next, we define the cost function, which evaluates one candidate configuration of the hyperparameters on a single instance of the 10-fold cross-validation:

```
R> hookRun <- function(instance, candidate, extra.params = NULL, config = list())
 {
   idx <- sample.n[instance + msik] \# Index of examples to hold out
   \# Train the network
   training.set <- iris[-idx, ]
   nn <- nnet(Species ~ . , data = training.set, maxit = 10,
   size = as.numeric(candidate\$values[["size"]]),
   decay = as.numeric(candidate\$values[["decay"]]),
   trace = FALSE)

   \# Test the network on hold-out examples
   unseen.input <- iris[idx, 1:4]
   predicted.output <- predict(nn, unseen.input, type = "class")
   target.output <- as.character(iris\$Species[idx])
   \# Return error rate
   return(mean(predicted.output != target.output))
 }
```

We are now ready to launch `irace`. We do it by means of the `irace` function by setting `hookRun` to the function defined above, `instances` to the indexes of the 10-folds previously created, and we specify a maximum budget of 250 calls to `hookRun`.

```
R> result <- irace(tunerConfig = list(
                   hookRun = hookRun,
                   instances = 1:n,
```

```
                    maxExperiments = 250,
                    logFile = ""),
                parameters = parameters)
```

The function `irace` will print information about its progress. For the sake of conciseness, we do not repeat this information here. At the end, we can examine the best configuration of the hyperparameters as follows:

```
R> candidates.print(result)
```

```
   size  decay
26   50 0.0343
29   44 0.0758
27   45 0.0727
```

Now, we can compare the cost of the best configuration found by `irace` versus the default hyperparameters of `nnet` as follows:

```
R> default <- sapply(1:n, hookRun, candidate=list(values=list(size=2, decay=0)))
R> result.list <- as.list(removeCandidatesMetaData(result[1,]))
R> tuned <- sapply(1:n, hookRun, candidate=list(values = result.list))
R> boxplot(list(default=default, tuned=tuned))
```

The resulting plot is given in Figure 2. The boxplot clearly shows that the tuned configuration is able to find better solutions than the default configuration of SANN. This small example is included in the documentation of the `irace` package and can be run with the command:

```
R> example(irace)
```

6.2 Tuning ACOTSP

ACOTSP (Stützle, 2002) is a software package that implements various ant colony optimization algorithms to tackle the symmetric traveling salesman problem (TSP). The example proposed here concerns the automatic configuration of all its 11 parameters. The goal is to find a configuration of **ACOTSP** that obtains the lowest solution cost in

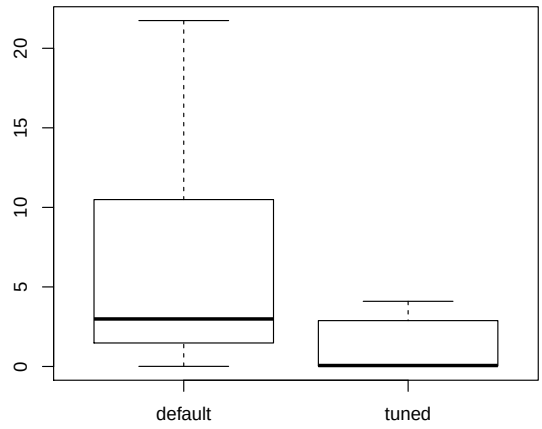

Figure 2: Boxplot of default vs. tuned configuration of SANN.

```
\# name      switch           type values    [| conditions (using R syntax)]
algorithm   "--"             c    (as, mmas, eas, ras, acs)
localsearch "--localsearch " c    (0, 1, 2, 3)
alpha       "--alpha "       r    (0.01, 5.00)
beta        "--beta "        r    (0.01, 10.00)
rho         "--rho  "        r    (0.00, 1.00)
ants        "--ants "        i    (5, 100)
nnls        "--nnls "        i    (5, 50)    | localsearch %in% c(1, 2, 3)
q0          "--q0 "          r    (0.0, 1.0) | algorithm %in% c("acs")
dlb         "--dlb "         c    (0, 1)     | localsearch %in% c(1,2,3)
rasrank     "--rasranks "    i    (1, 100)   | algorithm %in% c("ras")
elitistants "--elitistants " i    (1, 750)   | algorithm %in% c("eas")
```

Figure 3: Parameter file (`parameters.txt`) for tuning **ACOTSP**.

TSP instances within a given computation time limit. The setup of the tuning procedure is defined through various files.

First, we define a parameter file (`parameters.txt`, Figure 3) that describes the parameter space. We also create a configuration file (`tune-conf`, Figure 4) to overwrite some default options of `irace`. In particular, we set an execution directory (e.g., `./tuning/`)

```
execDir <- "./tuning/"
maxExperiments <- 1000
```

Figure 4: Configuration file (`tune-conf`) for tuning **ACOTSP**.

where temporary files are stored, and we set the tuning budget to 1 000 runs of **ACOTSP**. Next, we place the tuning instances in the subdirectory `./Instances/`, which is the default value of the option `instanceDir`. We create a basic `hook-run` script that simply runs the **ACOTSP** software for 20 seconds and prints the objective value of the best solution found.[3] We can now launch the tuning procedure as follows:

```
R> library("irace")
R> irace.cmdline()
```

The package provides a convenient command-line wrapper for Unix environments, called `irace`, located in `file.path(system.file(package="irace"), "bin")`, that basically invokes R and executes the commands above. The command-line wrapper also allows the user to specify many options directly from the command-line.

Most of the output is generated by the underlying **race** package, which prints a detailed progress of each race. After each race finishes, the set of elite configurations are printed. At the end, the best configurations found are printed as a table and as command-line parameters:

```
\# Best candidates
   algorithm localsearch alpha   beta    rho ants nnls    q0 dlb rasrank elitistants
189       acs          3 1.268 6.0930 0.6846   20   17 0.2812   1      NA         NA
332       acs          3 3.100 0.7011 0.4789   55   11 0.2630   1      NA         NA
320      mmas          3 1.361 9.0390 0.6852   64   25     NA   1      NA         NA
\# Best candidates (as commandlines)
                                                                                command
189  --acs --localsearch 3 --alpha 1.268 --beta 6.0930 --rho  0.6846 --ants 20 --nnls 17  \
    --q0 0.2812 --dlb 1
332  --acs --localsearch 3 --alpha 3.100 --beta 0.7011 --rho  0.4789 --ants 55 --nnls 11  \
    --q0 0.2630 --dlb 1
320  --mmas --localsearch 3 --alpha 1.361 --beta 9.0390 --rho  0.6852 --ants 64 --nnls 25 \
    --dlb 1
```

In addition, `irace` saves an R dataset file, by default as `irace.Rdata`, which may be read from R by means of the function `load()`. This dataset contains a list `tunerResults`, whose elements are:

[3] The package includes an example of this `hook-run` script for Unix environments, which can be found at `file.path(system.file(package="irace"), "examples","acotsp")`.

- `tunerConfig`: the configuration of `irace`.
- `parameters`: the parameter space.
- `experiments`: a matrix storing the result of all experiments performed across all iterations. Each entry is the result of evaluating one configuration on one instance at a particular iteration. The first column ('`instance`') indicates the instance tested in the experiments of the same row. The second column ('`iteration`') gives the iteration (race) number in which the experiments in the same row where performed. The remainder of the columns represent configurations, and their column names correspond to their IDs. Finally, '`NA`' represents that for some reason the candidate was not evaluated on a particular instance at that iteration, either because it did not exist yet or it was removed earlier.
- `allCandidates`: a data frame with all candidate configurations tested during the execution of `irace`.

7 Summary

This appendix presents the `irace` package, which implements the iterated racing procedure for automatic algorithm configuration. Iterated racing is a generalization of the iterated F-race procedure. The primary purpose of `irace` is to automatize the arduous task of configuring the parameters of an algorithm. However, it may also be used for determining good settings in other computational systems. The `irace` package has been designed with simplicity and ease of use in mind. Despite being implemented in R, no previous knowledge of R is required. In GNU/Linux and MacOS X, a command-line wrapper makes the use of R completely transparent to the user.

The `irace` package is available from CRAN. More information about `irace` is available at `http://iridia.ulb.ac.be/irace`.

List of Abbreviations

AA-TPLS	Adaptive Anytime Two-Phase Local Search
ACO	Ant Colony Optimization
AF-TPLS	Adaptive Anytime Two-Phase Local Search "Focus"
AF-TPLS$_{\mathcal{HV}}$	Adaptive Anytime Two-Phase Local Search "Focus" with Optimistic Hypervolume Improvement
AN-TPLS-1seed	Adaptive Anytime Two-Phase Local Search (using one seed)
AN-TPLS-1seed$_{\mathcal{HV}}$	Adaptive Anytime Two-Phase Local Search (using one seed) with Optimistic Hypervolume Improvement
AN-TPLS-2seed	Adaptive Two-Phase Local Search (without "Focus", using two seeds)
bPFSP	Bi-objective Permutation Flowshop Scheduling Problem
bQAP	Bi-objective Quadratic Assignment Problem
bTSP	Bi-objective Traveling Salesman Problem
D-TPLS	Double Two-Phase Local Search
$C_{\max}$	Makespan

Dynagrid	Dynamic Epsilon Grid
Dynagrid-HV	Dynamic Epsilon Grid with Hypervolume enhancement
EAF	Empirical Attainment Function
GRASP	Greedy Randomized Adaptive Search Procedures
`HQS`	High-quality sets
IG	Iterated Greedy
ILS	Iterated Local Search
`irace`	Iterated Race software package
MCOP	Multi-objective combinatorial problem
OHI	Optimistic Hypervolume Improvement
PFSP	Permutation Flowshop Scheduling Problem
***PFSP-*($C_{\max}$, *WT*)**	PFSP with Makespan and Weighted Tardiness Objectives
***PFSP-*(*SFT*, *TT*)**	PFSP with Sum of Flowtimes and Total Tardiness Objectives
***PFSP-*(*SFT*, *WT*)**	PFSP with Sum of Flowtimes and Weighted Tardiness Objectives
***PFSP-*$C_{\max}$**	PFSP with Makespan Objective
PFSP-SFT	PFSP with Sum of Flowtimes Objective

PFSP-TT	PFSP with Total Tardiness Objective
PFSP-WT	PFSP with Weighted Tardiness Objective
PLS	Pareto Local Search
QAP	Quadratic Assignment Problem
`RS`	Random solution
RA-TPLS	Regular Anytime Two-Phase Local Search
SA	Simulated Annealing
SLS	Stochastic Local Search
SFT	Sum of Flowtimes
TP+PLS	Hybrid Two-Phase + Pareto Local Search
TT	Total Tardiness
TSP	Traveling Salesman Problem
`TS`	Two high-quality solutions
TPLS	Two-Phase Local Search
WT	Weighted Tardiness

Bibliography

A

B. Adenso-Díaz and M. Laguna. (2006). Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114.

A. Alsheddy and E. Tsang. (2010). Guided Pareto local search and its application to the 0/1 multi-objective knapsack problems. In *Proceedings of MIC 2009, the 8th Metaheuristics International Conference*. University of Hamburg, Hamburg, Germany.

K. A. Andersen, K. Jörnsten, and M. Lind. (1996). On bicriterion minimal spanning trees: An approximation. *Computers & Operations Research*, 23(12):1171–1182.

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

E. Angel, E. Bampis, and L. Gourvés. (2004). Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoretical Computer Science*, 310(1-3): 135–146.

D. Angus and C. Woodward. (2009). Multiple objective ant colony optimisation. *Swarm Intelligence*, 3(1):69–85.

C. Ansótegui, M. Sellmann, and K. Tierney. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Heidelberg, Germany.

D. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. Concorde TSP solver. `http://www.tsp.gatech.edu/concorde`, ().

D. L. Applegate. (2006). *The traveling salesman problem: a computational study*. Princeton University Press, Princeton, NJ.

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

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

B

T. Back, D. B. Fogel, and Z. Michalewicz. (1997). *Handbook of evolutionary computation*. IOP Publishing.

P. Balaprakash, M. Birattari, and T. Stützle. (2007). Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, Heidelberg, Germany.

T. Bartz-Beielstein. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, Berlin, Germany.

T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors. (2010a). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Germany.

T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. (2010b). The sequential parameter optimization toolbox. In Bartz-Beielstein *et al.* (2010a), pages 337–360.

T. Bartz-Beielstein, J. Ziegenhirt, W. Konen, O. Flasch, P. Koch, and M. Zaefferer. *SPOT: Sequential Parameter Optimization*, (2011). R package.

R. Battiti, M. Brunato, and F. Mascia. (2008). *Reactive Search and Intelligent Optimization*, volume 45 of *Operations Research/Computer Science Interfaces*. Springer, New York, NY.

J. Bergstra and Y. Bengio. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305.

N. Beume and G. Rudolph. (2006). Faster S-metric calculation by considering dominated hypervolume as Klee's measure problem. In *Proceedings of the Second IASTED Conference on Computational Intelligence*, pages 231–236. ACTA Press, Anaheim.

N. Beume, B. Naujoks, and M. Emmerich. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3): 1653–1669.

N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. (2009). On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082.

L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. (2012). Automatic generation of multi-objective ACO algorithms for the biobjective knapsack problem. In *Swarm Intelligence, 8th International Conference, ANTS 2012*, volume 7461 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Heidelberg, Germany.

M. Birattari. (2009). *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer, Berlin/Heidelberg, Germany.

M. Birattari. (2004). *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

M. Birattari. (2003). The race package for R: Racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-037, IRIDIA, Université Libre de Bruxelles, Belgium.

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

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. (2010). F-race and iterated F-race: An overview. In Bartz-Beielstein *et al.* (2010a), pages 311–336.

P. C. Borges. (2000). CHESS - changing horizon efficient set search: A simple principle for multiobjective optimization. *Journal of Heuristics*, 6(3):405–418.

V. Bowman and J. Joseph. (1976). On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple Criteria Decision Making*, volume 130 of *Lecture Notes in Economics and Mathematical Systems*, pages 76–86. Springer, Berlin/Heidelberg.

R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis. (1998). The quadratic assignment problem. In *Handbook of Combinatorial Optimization*, volume 2, pages 241–338. Kluwer Academic Publishers.

C

E. Çela. (1998). *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, Kluwer Academic Publishers, Dordrecht, The Netherlands.

C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, NY.

W. J. Conover. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, third edition.

S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97.

P. Czyżżak and A. Jaszkiewicz. (1998). Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47.

D

K. Deb. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.

K. Deb and S. Chaudhuri. (2007). I-mode: An interactive multi-objective optimization and decision-making using evolutionary methods. *Lecture Notes in Computer Science*, 4403:788–802.

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

X. Delorme, X. Gandibleux, and F. Degoutin. (2010). Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research*, 204(2):206–217.

M. Dorigo, V. Maniezzo, and A. Colorni. (1991). The Ant System: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy.

M. Dorigo, V. Maniezzo, and A. Colorni. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26 (1):29–41.

M. M. Drugan and D. Thierens. (2010). Path-guided mutation for stochastic Pareto local search algorithms. In Schaefer *et al.* (2010), pages 485–495.

M. M. Drugan and D. Thierens. (2012). Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18:727–766.

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

J. Dubois-Lacoste. (2009a). Weight setting strategies for two-phase local search: A study on biobjective permutation flowshop scheduling. Technical Report TR/IRIDIA/2009-024, IRIDIA, Université Libre de Bruxelles, Belgium.

J. Dubois-Lacoste. (2009b). A study of Pareto and two-phase local search algorithms for biobjective permutation flowshop scheduling. Master's thesis, IRIDIA, Université Libre de Bruxelles, Belgium.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2009). Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In *Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 100–114. Springer, Heidelberg, Germany.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Supplementary material: Improving the Anytime Behavior of Two-Phase Local Search. http://iridia.ulb.ac.be/supp/IridiaSupp2010-012, (2010a).

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

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2010c). Adaptive "anytime" two-phase local search. In *Learning and Intelligent Optimization, 4th International Conference, LION 4*, volume 6073 of *Lecture Notes in Computer Science*, pages 52–67. Springer, Heidelberg, Germany.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011a). Improving the anytime behavior of two-phase local search. *Annals of Mathematics and Artificial Intelligence*, 61 (2):125–154.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011b). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38 (8):1219–1236.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2011c). Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2019–2026. ACM Press, New York, NY.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2012). Pareto local search algorithms for anytime bi-objective optimization. In *Proceedings of EvoCOP 2012 – 12th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 206–217. Springer, Heidelberg, Germany.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Supplementary material: Anytime pareto local search. http://iridia.ulb.ac.be/supp/IridiaSupp2013-003, (2013a).

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2013b). Combining two search paradigms for multi-objective optimization: Two-phase and Pareto local search. In *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 97–117. Springer, Berlin/Heidelberg.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. (2014). Anytime Pareto local search. *European Journal of Operational Research*. Submitted.

E

R. Eberhart and J. Kennedy. (1995). A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*,

pages 39–43.

M. Ehrgott. (2000). *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Germany.

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

M. Ehrgott and X. Gandibleux. (2008). Hybrid metaheuristics for multi-objective combinatorial optimization. In *Hybrid Metaheuristics: An emergent approach for optimization*, pages 221–259. Springer, Berlin, Germany.

A. E. Eiben, R. Hinterding, and Z. Michalewicz. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141.

F

T. A. Feo and M. G. C. Resende. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–113.

K. Florios and G. Mavrotas. (2014). Generation of the exact pareto set in multi-objective traveling salesman and set covering problems. *Applied Mathematics and Computation*, 237:1–19.

C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors. (2003). *Evolutionary Multi-Criterion Optimization Second International Conference, EMO 2003, Faro, Portugal, April 2003: proceedings*, volume 2632 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany.

C. M. Fonseca, L. Paquete, and M. López-Ibáñez. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pages 1157–1163. IEEE Press, Piscataway, NJ.

G

L. M. Gambardella and M. Dorigo. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 252–260. Morgan Kaufmann Publishers, Palo Alto, CA.

X. Gandibleux, N. Mezdaoui, and A. Fréville. (1997). A tabu search procedure to solve multiobjective combinatorial optimization problem. In *Advances in Multiple Objective and Goal Programming*, volume 455 of *Lecture Notes in Economics and Mathematical Systems*, pages 291–300. Springer, Heidelberg, Germany.

X. Gandibleux, H. Morita, and N. Katoh. (2003). Use of a genetic heritage for solving the assignment problem with two objectives. In Fonseca *et al.* (2003), pages 43–57.

C. García-Martínez, O. Cordón, and F. Herrera. (2007). A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *European Journal of Operational Research*, 180(1):116–148.

M. R. Garey and D. S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Co, San Francisco, CA.

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

M. J. Geiger. (2011). Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology. *Computers and Industrial Engineering*, 61(3): 805–812.

M. Gendreau and J.-Y. Potvin, editors. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, New York, NY, 2 edition.

A. M. Geoffrion, J. S. Dyer, and A. Feinberg. (1972). An interactive approach for multi-criterion optimization, with an application to the operation of an academic department. *Management Science*, 19(4):357–368.

F. Glover. (1989). Tabu search – Part I. *INFORMS Journal on Computing*, 1(3):190–206.

F. Glover. (1998). A template for scatter search and path relinking. In *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 1–51. Springer, Heidelberg, Germany.

D. E. Goldberg. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA.

V. Grunert da Fonseca, C. M. Fonseca, and A. O. Hall. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. In *Evolutionary Multi-criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 213–225. Springer, Heidelberg, Germany.

H

H. W. Hamacher and G. Ruhe. (1994). On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52(4):209–230.

M. P. Hansen. (1997). Tabu search for multiobjective optimization: MOTS. In *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, pages 574–586. Springer Verlag.

P. Hansen and N. Mladenovic. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

K. Helsgaun. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

H. H. Hoos. (2012). Programming by optimization. *Communications of the ACM*, 55(2): 70–80.

H. H. Hoos and T. Stützle. (2005). *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA.

M. S. Hussin and T. Stützle. (2010). Tabu search vs. simulated annealing for solving large quadratic assignment instances. Technical Report TR/IRIDIA/2010-020, IRIDIA, Université Libre de Bruxelles, Belgium.

F. Hutter, D. Babić, H. H. Hoos, and A. J. Hu. (2007). Boosting verification by automatic tuning of decision procedures. In *FMCAD'07: Proceedings of the 7th International Conference Formal Methods in Computer Aided Design*, pages 27–34. IEEE Computer Society, Washington, DC, USA, Austin, Texas, USA.

F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. (2009a). An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009*, pages 271–278. ACM Press, New York, NY.

F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. (2009b). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36: 267–306.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. (2010). Automated configuration of mixed integer programming solvers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, Heidelberg, Germany.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, Heidelberg, Germany.

I

H. Ishibuchi and T. Murata. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics – Part C*, 28(3):392–403.

J

C. H. Jackson. (2011). Multi-state models for panel data: The msm package for R. *Journal of Statistical Software*, 38(8):1–29.

A. Jaszkiewicz. (2002a). Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, 137(1):50–71.

A. Jaszkiewicz. (2002b). On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – a comparative experiment. *IEEE Transactions on Evolutionary Computation*, 6(4):402–412.

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

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

D. S. Johnson and L. A. McGeoch. (1997). The traveling salesman problem: A case study in local optimization. In *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK.

K

A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. (2009). SATenstein: Automatically building local search SAT solvers from components. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 517–524. AAAI Press, Menlo Park, CA.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

J. D. Knowles and D. Corne. (1999). The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999)*, pages 98–105. IEEE Press, Piscataway, NJ.

J. D. Knowles and D. Corne. (2003a). Instance generators and test suites for the multiobjective quadratic assignment problem. In Fonseca *et al.* (2003), pages 295–310.

J. D. Knowles and D. Corne. (2003b). Properties of an adaptive archiving algorithm for storing nondominated vectors. *IEEE Transactions on Evolutionary Computation*, 7(2): 100–116.

L

M. Laumanns, L. Thiele, and E. Zitzler. (2004). Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation*, 8(2):170–182.

E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys. (1985). *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. jws-ny.

A. Liefooghe, S. Mesmoudi, J. Humeau, L. Jourdan, and E.-G. Talbi. (2009). A study on dominance-based local search approaches for multiobjective combinatorial optimization. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS 2009*, volume 5752 of *Lecture Notes in Computer Science*, pages 120–124. Springer, Heidelberg, Germany.

A. Liefooghe, J. Humeau, S. Mesmoudi, L. Jourdan, and E.-G. Talbi. (2011). On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18(2):317–352.

M. López-Ibáñez and T. Stützle. (2010). Automatic configuration of multi-objective ACO algorithms. In *Swarm Intelligence, 7th International Conference, ANTS 2010*, volume 6234 of *Lecture Notes in Computer Science*, pages 95–106. Springer, Heidelberg, Germany.

M. López-Ibáñez and T. Stützle. (2012). The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6): 861–875.

M. López-Ibáñez, L. Paquete, and T. Stützle. (2006). Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms*, 5(1):111–137.

M. López-Ibáñez, L. Paquete, and T. Stützle. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Bartz-Beielstein *et al.* (2010a), pages 209–222.

M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. (2011a). The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

M. López-Ibáñez, J. D. Knowles, and M. Laumanns. (2011b). On sequential online archiving of objective vectors. In *Evolutionary Multi-criterion Optimization (EMO 2011)*, volume 6576 of *Lecture Notes in Computer Science*, pages 46–60. Springer, Heidelberg, Germany.

S. Loudni and P. Boizumault. (2008). Combining VNS with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191:

705–735.

H. R. Lourenço, O. Martin, and T. Stützle. (2002). Iterated local search. In *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

H. R. Lourenço, O. Martin, and T. Stützle. (2010). Iterated local search: Framework and applications. In Gendreau and Potvin (2010), chapter 9, pages 363–397.

T. Lust and A. Jaszkiewicz. (2010). Speed-up techniques for solving large-scale biobjective TSP. *Computers & Operations Research*, 37(3):521–533.

T. Lust and J. Teghem. (2010a). The multiobjective multidimensional knapsack problem: a survey and a new approach. *Arxiv preprint arXiv:1007.4063*.

T. Lust and J. Teghem. (2012). The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 19(4): 495–520.

T. Lust and J. Teghem. (2010b). Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510.

M

O. Maron and A. W. Moore. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Research*, 11(1–5):193–225.

F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. (2013). From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In *Learning and Intelligent Optimization, 7th International Conference, LION 7*, volume 7997 of *Lecture Notes in Computer Science*, pages 321–334. Springer, Heidelberg, Germany.

F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, M.-E. Marmion, and T. Stützle. (2014a). Algorithm comparisons by automatically configurable metaheuristic frameworks: a case study using flow-shop scheduling problems. In *Hybrid Metaheuristics*, Lecture Notes in Computer Science. Springer, Heidelberg, Germany. Accepted.

F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. (2014b). Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*. Accepted subject to minor revisions.

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

G. Minella, R. Ruiz, and M. Ciavotta. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*,

38(11):1521–1533.

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

N

P. Naccache. (1978). Connectedness of the set of nondominated outcomes in multicriteria optimization. *Journal of Optimization Theory and Applications*, 25(3):459–467.

V. Nannen and A. E. Eiben. (2006). A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 183–190. ACM Press, New York, NY.

Y. S. G. Nashed, P. Mesejo, R. Ugolotti, J. Dubois-Lacoste, and S. Cagnoni. (2012). A comparative study of three GPU-based metaheuristics. In *PPSN 2012, Part II*, volume 7492 of *Lecture Notes in Computer Science*, pages 398–407. Springer, Heidelberg, Germany.

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

F. Nerri and C. Cotta. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14.

P

L. Paquete. (2005). *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis*. PhD thesis, FB Informatik, TU Darmstadt, Germany.

L. Paquete and T. Stützle. (2006). A study of stochastic local search algorithms for the biobjective QAP with correlated flow matrices. *European Journal of Operational Research*, 169(3):943–959.

L. Paquete and T. Stützle. (2009a). Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis. In *Multiobjective Programming and Goal Programming: Theoretical Results and Practical Applications*, volume 618 of *Lecture Notes in Economics and Mathematical Systems*, pages 69–77. Springer, Berlin.

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

L. Paquete and T. Stützle. (2003). A two-phase local search for the biobjective traveling salesman problem. In Fonseca *et al.* (2003), pages 479–493.

L. Paquete and T. Stützle. (2007). Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In *Handbook of Approximation Algorithms and Metaheuristics*, pages 29–1—29–15. Chapman & Hall/CRC, Boca Raton, FL.

L. Paquete, M. Chiarandini, and T. Stützle. (2004). Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems, pages 177–200. Springer, Berlin, Germany.

L. Paquete, C. M. Fonseca, and M. López-Ibáñez. (2006). An optimal algorithm for a special case of Klee's measure problem in three dimensions. Technical Report CSI-RT-I-01/2006, CSI, Universidade do Algarve. Superseded by paper in IEEE Transactions on Evolutionary Computation Beume *et al.* (2009).

L. Paquete, T. Schiavinotto, and T. Stützle. (2007). On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, 156:83–98.

S. N. Parragh, K. F. Doerner, R. F. Hartl, and X. Gandibleux. (2009). A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks*, 54(4): 227–242.

R

A. Radulescu, M. López-Ibáñez, and T. Stützle. (2013). Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In *Evolutionary Multi-criterion Optimization (EMO 2013)*, volume 7811 of *Lecture Notes in Computer Science*, pages 825–840. Springer, Heidelberg, Germany.

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

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, (2008). ISBN 3-900051-07-0.

C. Reeves. (2010). Genetic algorithms. In Gendreau and Potvin (2010), chapter 5, pages 109–140.

G. Reinelt. (1994). *The traveling salesman: computational solutions for TSP applications*. Springer Verlag.

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

S

R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors. (2010). *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany.

J. D. Schaffer. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *ICGA-85*, pages 93–100. Lawrence Erlbaum Associates.

P. Serafini. (1992). Simulated annealing for multiple objective optimization problems. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making (MCDM'91)*, volume 1, pages 87–96. Springer Verlag.

K. Sörensen. (2013). Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*.

T. Stützle. **ACOTSP**: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, (2002).

A. Suppapitnarm, K. Seffen, G. Parks, and P. Clarkson. (2000). A simulated annealing algorithm for multiobjective optimization. *Engineering Optimization*, 33(1):59–85.

T

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

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

U

E. Ulungu and J. Teghem. (1995). The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165.

E. Ulungu, J. Teghem, P. H. Fortemps, and D. Tuyttens. (1999). MOSA method: a tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8(4):221–236.

V

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

A. Valsecchi, J. Dubois-Lacoste, T. Stützle, S. Damas, J. Santamiaría, and L. Marrakchi-Kacem. (2013). Evolutionary medical image registration using automatic parameter tuning. In *Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)*, pages 1326–1333. IEEE Press, Piscataway, NJ.

T. K. Varadharajan and C. Rajendran. (2005). 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.

C. Voudouris and E. Tsang. (1999). Guided local search and its application to the travelling salesman problem. *European Journal of Operational Research*, 113(2):469–499.

W

S. Wessing, N. Beume, G. Rudolph, and B. Naujoks. (2010). Parameter tuning boosts performance of variation operators in multiobjective optimization. In Schaefer *et al.* (2010), pages 728–737.

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

Z

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

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

E. Zitzler, M. Laumanns, and L. Thiele. (2002). SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation and Control*, pages 95–100. CIMNE, Barcelona, Spain.

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