Reacting and Adapting to the Environment

Aymeric Blot

UNIVERSITÉ DE LILLE – CRIStAL
ÉCOLE DOCTORALE SCIENCES POUR L'INGÉNIEUR

REACTING AND ADAPTING TO THE ENVIRONMENT

Designing Autonomous Methods for Multi-Objective Combinatorial Optimisation

RÉAGIR ET S'ADAPTER À SON ENVIRONNEMENT

Concevoir des méthodes autonomes pour l'optimisation combinatoire à plusieurs objectifs

BLOT Aymeric

Thèse préparée et soutenue publiquement le 21 septembre 2018,
en vue de l'obtention du grade de Docteur en Informatique.

Jury:

Mr.	BATTITI Roberto	University of Trento	*Referee*
Mr.	DE CAUSMACKER Patrick	KU Leuven	*Examiner*
Mr.	HOOS H. Holger	Leiden University	*Examiner*
Mrs.	JOURDAN Laetitia	University of Lille	*Supervisor*
Mrs.	KESSACI Marie-Éléonore	University of Lille	*Co-adviser*
Mr.	MATHIEU Philippe	University of Lille	*Examiner*
Mr.	SAUBION Frédéric	University of Angers	*Referee*
Mr.	STÜTZLE Thomas	Université Libre de Bruxelles	*Examiner*

Centre de Recherche en Informatique, Signal et Automatique de Lille
Université Lille 1 – Bâtiment M3 extension – Avenue Carl Gauss
59655 Villeneuve d'Ascq Cedex FRANCE

Acknowledgements

First of all I would like to thank the jury members and especially Laetitia Jourdan and Marie-Éléonore Kessaci, that supported me during the last three years on an every-day basis. Laetitia, I am very grateful for everything you have done since I met you for the first time six years ago, in particular for the many research opportunities in Lille but also in Nagano and in Vancouver. Marie-Éléonore, you taught me perseverance and thoroughness; without your advising this thesis would definitively not have happened. Many thanks to Patrick de Causmaecker, Philippe Mathieu, and Thomas Stützle for accepting to be part of my jury, and to Roberto Battiti and Frederic Saubion for also reporting on my manuscript; I am honoured for the interest you gave to my work.

I would like to thank my coauthors Patrick de Causmaecker, Holger H. Hoos, Manuel López-Ibáñez, and Heike Trautmann, for the work we carried out together during this thesis. Our discussions have brought so much and pushed this thesis much further than I could have done by myself. I also want to thank Hernán Aguirre and Kiyoshi Tanaka: I only stayed in Nagano three months but I will never forget them; they confirmed my passion for research and changed my life forever. どうもありがとうございました。

Generally speaking, thank you everyone from the ORKAD team, the former DOLPHIN team, and my colleagues from INRIA and the CRIStAL laboratory. I will also always keep very good memories from the years I spent at the ENS Rennes: they brought me my passion for computer science and research, and I could not have dreamt for a better formation.

I cannot thank enough the friends I made along the way, in Orsay, in Rennes, and in Lille. Thank you Grégoire, Thomas, Hugo, Nicolas, Lauriane, Mathieu, Lucien, Maxence, and Léonard; I was not always easy to put up with, but you supported me and I would not be where I stand today without you.

Finally, I am grateful to my family. To my parents, for their constant support during the last 26 years; and to my brother and sister and their companions for giving me three wonderful nieces.

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

> The journey of a thousand miles begins with a single step.
>
> Lao-Tzu

This thesis lays on the intersections between multi-objective combinatorial optimisation, local search algorithms, and automatic algorithm design. It was carried out in the ORKAD[1] team, that focuses on combining combinatorial optimisation and data mining to solve optimisation problems. Before its creation in 2017 as an independent research team in the CRIStAL[2] laboratory, the ORKAD team was a research group associated to the DOLPHIN[3] joint project team, collaboration between the CRIStAL laboratory and the Inria Lille-Nord Europe research institute.

Motivations

Optimisation problems are ubiquitous. Numerous real-world problems, such as planning schedules, building financial portfolios, routing vehicles, or predicting future patients at risk in healthcare, can be formulated by determining the best solution among a very large number of possible ones. For many optimisation problems, while evaluating the quality of a single solution is usually fairly easy and quick, solving them to optimality is much more computationally expensive as their difficulty increases at least exponentially with the size of the problem. The question of whether of not these problems, called $\mathcal{NP}$-hard, can theoretically be solved efficiently is at the core of one of the major unsolved problem of computer science: the $\mathcal{P}$ *versus* $\mathcal{NP}$ problem.

In order to obtain good solutions for $\mathcal{NP}$-hard problems that would be too large to be solved to optimality in reasonable time, or in general for any large-scale op-

[1] *Operational Research, Knowledge And Data*
[2] *Centre de Recherche en Informatique, Signal et Automatique de Lille* (UMR CNRS 9189)
[3] *Discrete multiobjective Optimization for Large-scale Problems with Hybrid dIstributed techNiques*

timisation problem, approximation algorithms such as metaheuristics have been proposed and are usually preferred. Local search algorithms are metaheuristics that focus on the structure of the problem to solve, in order to benefit from the relation between similar solutions and progressively and iteratively approach optimal solutions. They have been shown to be very efficient, either used as self-contained algorithms or hybridised into more complex metaheuristics.

Along with the other metaheuristics, local search algorithms are very generic approaches that can be applied to many combinatorial optimisation problems as long as few assumptions over the problem modelling are respected, such as a finite or at least countable number of solutions. However, it is to be expected that no single algorithm can perform the best on every problem, so metaheuristics usually involves many possible variants, using many alternative strategies, to improve performance on specific problems structures. Given a specific problem, automatically determining which of the many variants of the algorithm will be the most efficient, or, more broadly, automatically designing the optimal algorithm that use the most efficient strategies, is a recent but thriving research field.

Finally, optimisation problems as well as automatic design problems can involve more than a single criteria to optimise. If as single quality is to be maximised, or a single cost is to be minimised, the resolution process usually results in a single final optimal solution. However, considering multiple objectives usually involves a much richer context in which many incomparable compromise solutions are to be sought. If classical multi-objective optimisation problems are increasingly studied and understood, multi-objective concepts for algorithm design problems have only been considered recently.

Based on these observations, we investigate in this thesis the different intersections between combinatorial optimisation, local search algorithms, and automatic algorithm design, in the context of multi-objective optimisation. In particular, our focus is divided in one hand on multi-objective automatic algorithm design, i.e., the automatic design of algorithms relatively to multiple performance metrics, and on the other hand on the automatic design of multi-objective algorithms, using multi-objective local search algorithms. Finally, we investigate two aspects of adaptation using automatic algorithm design: first with a predictive viewpoint, where algorithms are configured with regard to performance on learning instances, and with an complementary adaptive viewpoint, where algorithms autonomously react during their execution to the instance being solved.

Outline

Following this general introduction, this thesis is organised in four successive parts, starting from general notions about multi-objective optimisation and presenting the state of the art of automatic algorithm design, then focusing on multi-objective local search (MOLS) algorithms, before focusing on automatically designing MOLS algorithms using *offline* algorithm design, and finally discussing two possible *online* extensions.

Part I: Multi-objective Optimisation and Algorithm Design

First, Part I lays the foundations of this thesis down, and presents the research fields of both combinatorial optimisation and algorithm design.

Chapter 1 details the multi-objective context that we use in this thesis. We give the general definitions and notions of multi-objective combinatorial optimisation, we discuss the performance assessment of multi-objective algorithms, and we present the three combinatorial problems that will be tackled in the experiments of Part III and Part IV.

Chapter 2 presents the research field of automatic algorithm design (AAD) with a proposition of a new taxonomy. We first introduce the foundations of our proposition, then we give a detailed overview of the existing related research fields and state of the art methods. Finally we present and motivate our taxonomy proposition by discussing existing works according to several general viewpoints: a temporal viewpoint: *"when does the automatic design take place?"*, a structural viewpoint: *"how much of the algorithmic design can be modified?"*, and finally a complementary complexity viewpoint, related to the available knowledge sources.

Part II: Multi-objective Local Search

Next, Part II focuses on the class of algorithms studied in this thesis, the multi-objective local search (MOLS) algorithms.

Chapter 3 provides a technical and historical background on MOLS algorithms. We first present their specific notions and philosophy, and conduct a chronological survey of the use of local search techniques in multi-objective algorithms. Then, we discuss the local search strategies found in the literature, and finally we propose a new unification of MOLS algorithms and detail how the major literature algorithms are instantiated.

Chapter 4 details the specific implementations of the MOLS algorithms that will be used in the following chapters, based on the unified structure presented in

Chapter 3. We first present a *classical* MOLS algorithm that exposes many parameters in order to automatically configure it in Chapter 6. Then, we discuss how we can involve generic mechanisms to dynamically control the value of some parameters of MOLS algorithms during their execution, and present the adaptive MOLS algorithm that is considered in Chapter 7. Finally, we present the notion of configuration scheduling that is investigated in Chapter 8.

Part III: Automatic Offline Design

Then, Part III investigates *offline* AAD approaches, and more specifically multi-objective algorithm configuration, when the algorithm configuration is optimised before its execution.

Chapter 5 introduces MO-ParamILS, a multi-objective automatic configuration framework based on MOLS techniques, as a dedicated approach for multi-objective configuration scenarios. First we formally define multi-objective algorithm configuration and detail some use cases. Then, we present ParamILS, a prominent and well-known single-objective algorithm configurator based on a single-objective local search, before proposing MO-ParamILS, that we based on a MOLS algorithm. Finally, we study the performance of the multiple variants of MO-ParamILS against approaches directly using ParamILS only on various use cases, to show the worth of using multi-objective approaches against classical single-objective approaches.

Chapter 6 deals with the automatic design of MOLS algorithms. In the course of three successive studies, the use of a multi-objective configurator is compared against the use a single-objective configurator. Three configuration approaches are compared: first a classical baseline of optimising the convergence of the MOLS algorithms, then an aggregated approach focusing on convergence while taking into account the distribution of solutions, and finally the simultaneous optimisation of both convergence and distribution independently. The first study provides comprehensive preliminary results on classical problems by limiting itself to a small subset of possible MOLS configurations. The second study provides conclusive results by tackling a much larger pool of configuration. Finally the third study validates our observations by tacking artificially constructed scenarios on which the correlation between objectives is controlled. To ensure fair comparisons, the multi-objective and single-objective approaches used are based on ParamILS and MO-ParamILS, as they are based on the same principles.

Part IV: Automatic Online Design

Last, Part IV discusses two extensions of MOLS automatic design, involving *online* elements, i.e., when modifications of the MOLS configuration occur during the execution.

Chapter 7 uses notions of parameter control to delay the prediction of the optimal configuration. Instead of only using the prediction resulting from the *offline* configuration process, we investigate how MOLS algorithms can benefit from generic control mechanisms by using multiple efficient strategies. Following the discussion of Chapter 4, we first survey some of the generic control mechanisms that can easily be integrated in our adaptive MOLS structure, before discussing the actual performance of the simplest of them.

Chapter 8 extends the configuration process investigated in Chapter 6 by considering schedules of configurations, rather than using a unique configuration during the entire execution. Following the discussion of Chapter 4, we investigate the automatic configuration of schedules dividing the execution between two and three different strategies.

Part I

Multi-objective Optimisation and Algorithm Design

If you can't criticise, you can't optimise.

Harry Potter and the Methods of Rationality
Eliezer Yudkowsky

Chapter 1

Multi-objective Metaheuristics

> In the beginning there was nothing, which exploded.
>
> *Lords and Ladies.*
> Terry Pratchet

In this chapter, we present multi-objective optimisation, its necessary definitions and notions, then give a short overview of multi-objective metaheuristics. We also present the performance indicators and the permutations problems that we will use in the following chapters.

1.1 Multi-objective Combinatorial Optimisation

1.1.1 Introduction

Optimisation problems arise in many fields of mathematics, computer science and engineering. They deal with finding the *best* solutions from all possible solutions. Optimisation problems comprise *continuous* optimisation problems, in which solutions are described using decision variables taking uncountable values, and *discrete* optimisation problem, in which all these variables necessarily take specific and countable values. In this thesis, we consider *combinatorial* optimisation problems: discrete optimisation problems in which the number of solutions is finite, although in practice often too high to be exhaustively enumerated in a reasonable computation time.

If possible solutions can naturally be ranked using a single scalar metric, such as for example a cost to minimise, or a reward to maximise, then the optimisation problem is denoted as *single-objective*. On the contrary, if the goal is to find the

solutions simultaneously optimising several metrics, then the problem is denoted as a *multi-objective* (or *multi-criteria*) optimisation problem. In that case, it usually involves a trade-off between multiple conflicting objectives.

1.1.2 Definition

In multi-objective optimisation (MOO), a set $\mathcal{D}$ of solutions is investigated regarding multiple criteria characterising their quality. A MOO problem (MOOP) involves optimising simultaneously a vector of n ($n \geqslant 2$) distinct functions $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ over the set $\mathcal{D}$, and can be formulated following Equation 1.1, where $x = (x_1, x_2, \ldots, x_m)$ is a vector of decision variables.

$$\text{(MOOP)} \quad \begin{cases} \text{optimise} & F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ \text{subject to} & x \in \mathcal{D} \end{cases} \tag{1.1}$$

The set $\mathcal{D}$ is also called the *search space*. Its image through F is called the *objective space*. A function f_k is either called a *criterion*, an *objective function*, or simply an *objective*.

In multi-objective combinatorial optimisation (MOCO) problems, the set $\mathcal{D}$ of solutions is finite and the domains of the decisions variables of x are all discrete. Each function f_k can be assumed without loss of generality to be minimised, as maximising or mixed MOO problems can be easily mapped to analogous minimising MOO problems using opposite functions $f'_i(x) = -f_i(x)$.

In the following sections and chapters, unless specified otherwise, every criterion will be supposed to be minimised.

1.1.3 Solution Comparison

The two main approaches used to deal with MOO problems are either to use an *a priori* approach, if preferences over the different objectives are known (e.g., scalarising the vector $F(x)$ to a single objective $f(x)$), or to use an *a posteriori* approach, optimising each objective simultaneously. There are also *interactive* approaches in which the preferences of a decision maker are taken in account during the optimisation process, but they are much less used due to the heavy cost of constant human interaction.

In the following, we present first the Pareto dominance, then some of its many a priori alternatives.

Pareto Dominance

A posteriori approaches are based on the concept of *Pareto dominance*, used to capture trade-offs between the criteria f_k. Pareto dominance (or *Pareto efficiency*) is originally an economical notion proposed by Pareto (1896), which has then be broadly applied in many contexts beyond economics such as mathematics, engineering, or life sciences.

A solution s_1 is said to dominate a solution s_2 (denoted as $s_1 \succ s_2$) if, and only if, (i) s_1 is better than or equal to s_2 according to all criteria, and (ii) there is at least one criterion according to which s_1 is strictly better than s_2 (Equation 1.2, when every criterion is to be minimised).

$$s_1 \succ s2 \iff \begin{cases} \forall\, k \in \{1, \dots, n\},\ f_k(s_1) \leqslant f_k(s_2), \text{ and} \\ \exists\, k \in \{1, \dots, n\},\ f_k(s_1) < f_k(s_2) \end{cases} \tag{1.2}$$

The Pareto dominance does not imply a complete order on the set of all possible solutions. If neither $s_1 \succ s_2$ nor $s_2 \succ s_1$, then the solutions s_1 and s_2 are said *incomparable*. A set S of solutions in which there are no $s_1, s_2 \in S$ such that $s_1 \succ s_2$ is called a *Pareto set* or a *Pareto front*. The goal when solving a MOOP is to determine the best Pareto set, i.e., the set $S^\star \subset \mathcal{D}$ such that there is no $s' \in \mathcal{D}$ that dominates any of the $s \in S^\star$; this set is referred to as the *Pareto optimal set*.

Weighted linear scalarisation

The most simple way to aggregate all criteria into a single function is to use a weighted sum of the different objectives. Given $W = (w_1, w_2, \dots, w_n)$ a weight vector of n coefficients, the goal is to optimise a scalar function $f(x)$ instead of optimising the vector $F(x)$ (Equation 1.3).

$$f(x) = \sum_{k=1}^{n} w_k f_k(x) \quad \text{with} \sum_{k=1}^{n} w_k = 1 \tag{1.3}$$

Using this approach, optimising a weighted sum of multiple objectives corresponds to searching for an optimal solution for a given MOOP in a specific direction in objective space. It is known that under certain circumstances (namely, when the Pareto optimal front $S^\star$ is not convex) some optimal solutions cannot be obtained in this manner. In such cases, Pareto-based multi-objective optimisation algorithms are usually preferred.

Weighted Chebyshev norm

Instead of minimising a linear aggregation of the different objective, the weighted Chebyshev norm associates the quality of a solution x to the worse of its component $f_k(x)$, using the distance to a given reference point z (Equation 1.4).

$$f(x) = \max_{1 \leqslant k \leqslant n} w_k \cdot |f_k(x) - f_k(z)| \quad \text{with} \sum_{k=1}^{n} w_k = 1 \tag{1.4}$$

While this approach pressures the algorithm to optimise each objective simultaneously, it consequently makes it impossible to find the extreme solutions of the Pareto set.

Lexicographical ordering

If the objective functions f_k can be ordered according to their order of importance, a lexicographical ordering can also replace the Pareto dominance (Equation 1.5).

$$s_1 \succ s2 \iff \exists\, k \in \{1, \ldots, n\}, \begin{cases} \forall\, i \in \{1, \ldots, k\},\ f_i(s_1) = f_i(s_2), \text{ and} \\ f_k(s_1) < f_k(s_2) \end{cases} \tag{1.5}$$

Multi-objective indicators

Finally, in addition to using the Pareto dominance, it is possible to use binary quality indicators, such as for example hypervolume (Zitzler and Thiele, 1999), to compare solutions to either other single solutions or to whole fronts of solutions. This approach has been successfully applied to many algorithms, leading to the indicator-based algorithm family including, not exhaustively, the indicator-based evolutionary algorithm (IBEA, Zitzler and Künzli, 2004), the indicator-based multi-objective local search algorithm (IBMOLS, Basseur and Burke, 2007), and the indicator-based ant colony optimisation algorithm (IBACO, Mansour and Alaya, 2015).

1.1.4 Multi-objective Metaheuristics

Metaheuristics are high-level algorithms designed to quickly find good solutions for a large range of optimisation problems too difficult for exact algorithms. Indeed, many combinatorial optimisation problems are $\mathcal{NP}$-hard with an number of possible solutions growing exponentially, therefore requiring approximation mechanisms in order to obtain high-quality solutions in a reasonable amount of time. However, approximation algorithms do not guaranty the optimality of the

final solutions. Exact algorithms can nevertheless be used to get optimal solutions, either on small instances or sub-problems, or after reduction of the problem size.

Metaheuristics can be classified into nature-inspired and local search algorithms. While the former generally involve evolution, culture, or group characteristics to simultaneously evolve multiple solutions together, the latter focus more on intensifying individual solutions by intensifying the search on similar solutions. In the following, we present some of the more common multi-objective metaheuristics.

Nature-inspired Algorithms

Nature-inspired algorithms, or *bio-inspired* algorithms, are generally inspired by biological processes, and based on abstract concepts such as evolution, environmental pressure, and natural selection (survival of the fittest), as well as on concrete observations such as animal behaviour modelling. The most well known include evolutionary algorithms (EA's) such as the genetic algorithm (GA, Holland, 1992), swarm algorithms such as the particle swarm optimisation algorithm (PSO, Kennedy and Eberhart, 1995) and ant colony optimisation algorithms (ACO, Dorigo et al., 1996).

As for multi-objective nature-inspired algorithms, the most popular are nowadays recent variants based on the MOEA/D (Zhang and Li, 2007), a multi-objective EA based on decomposition; NSGA-II (Deb et al., 2000, 2002), a non-dominated sorting GA; SPEA2 (Zitzler and Thiele, 1999; Zitzler et al., 2001), a strength Pareto EA; and IBEA (Zitzler and Künzli, 2004), an indicator-based EA. The reader is referred to Coello et al. (2007) or Deb (2001) for more in-depth presentations of many multi-objective population-based and evolutionary algorithms.

We note in particular the existence of the MOACO framework (López-Ibáñez and Stützle, 2010a,b), which specifically provides a general multi-objective ant colony optimisation framework to use with automatic design tools. It is able to instantiate most of the multi-objective ACO algorithms from the literature and many combinations of components yet never investigated.

Local Search Algorithms

Local search (LS) algorithms exploit the structure of the search space to iteratively find better and better solutions. They are based on the idea that small modifications in the representation of a solution may lead to either a small improvement or a small deterioration of its initial quality, leading to the notion of *neighbourhood*, that gives a structure to the search space by connecting *close* solutions. This notion is often called the *proximate optimality principle* (e.g., Glover and Laguna, 1997).

LS algorithms are originally very efficient metaheuristics designed for single-objective problems (Hoos and Stützle, 2004). They have been adapted for multi-objective problems in various ways, either directly extended from well-known and established single-objective algorithms (e.g., Serafini, 1994; Ulungu et al., 1995; Czyzak and Jaszkiewicz, 1996; Hansen, 1997), or hybridised with and within evolutionary algorithms (e.g., Ishibuchi and Murata, 1996; Knowles and Corne, 1999; Talbi et al., 2001).

A detailed chronological overview of multi-objective local search algorithms will be given in Chapter 3.

1.2 Performance Assessment

The use of Pareto-based multi-objective algorithms leads to fronts of final solutions. In order to compare the performance of such algorithms, it is then necessary to be able to quantify the quality of Pareto sets.

1.2.1 Overview

Several characteristics of Pareto sets can be measured. Through the use of the many performance indicators proposed in the literature (Knowles and Corne, 2002; Okabe et al., 2003), three main properties of Pareto can be assessed: accuracy, diversity, and cardinality.

Accuracy: the front of solutions is close to the theoretical Pareto optimal front, either by volume or distance.
Diversity: the solutions of the front are either well-distributed or well-spread.
Cardinality: the front contains a large number of high-quality solutions.

According to a recent survey (Riquelme et al., 2015), the most commonly used performance indicators in the literature are the following.

Hypervolume (HV): (accuracy, diversity) based on the volume of the search space that contains dominated solutions (Zitzler and Thiele, 1999);
Generational distance (GD): (accuracy) based on the distance of the solutions of the front to the solutions of a reference front (van Veldhuizen and Lamont, 2000);
Epsilon family (ε): (all) based on the minimum factor ε (additive or multiplicative) by which the front is worse than a reference front regarding all objectives (Zitzler et al., 2003);

Inverted generational distance (IGD): (accuracy, diversity) similar to GD, based on the distance of the solutions of a reference front to the solutions of the given front (Coello and Cortés, 2005);

Spread (Δ): (diversity) based on the distribution and spread achieved among the solutions (Deb et al., 2002);

Two set coverage (C): (all) based on the fraction of solutions of the front dominated by at least one solution of a reference front (Zitzler and Thiele, 1998).

It was shown that it is generally not possible to aggregate all of these properties into a single indicator; it is thus recommended to consider multiple performance indicators, preferably ones that complement each other, in order to assess the efficiency multi-objective optimisation algorithms fairly (Zitzler et al., 2003). In practice, the hypervolume indicator (Zitzler and Thiele, 1999) is by far the indicator the most commonly used in the multi-objective literature, while the other indicators are much less used.

Finally, multi-objective performance indicators are either binary metrics, that compare two different sets of solutions (e.g., one set of solution with a reference set or a reference point), or unary metrics, that are able to give independent quality assessment.

In the following, we present in more detail the hypervolume indicator and the Δ spread indicator, that will be used in the experiments of this thesis. These two specific indicators have been chosen first because they are unary performance indicators, a restriction of the current automatic algorithm configurators; they are also well known and used in the literature, and the spread enable to more explicitly consider diversity, as hypervolume is first and foremost an indicator focused on accuracy.

1.2.2 Hypervolume

Hypervolume (HV) is a performance indicator proposed by Zitzler and Thiele (1999); the idea is to compute the volume of dominated space in objective space. Assuming normalised objective values in $[0, 1]$, the unary hypervolume measures the volume between a given Pareto set of solutions and the point $(1, 1)$, as pictured on Figure 1.1.

Hypervolume needs to be maximised, with a normalised minimal value of 0 when the front is reduced to the point $(1, 1)$ and an optimal value of 1 when the front is reduced to the point $(0, 0)$. In the later chapters, in order to facilitate analysis, we use a minimising variant of hypervolume instead, computed as $1 - \text{HV}$.

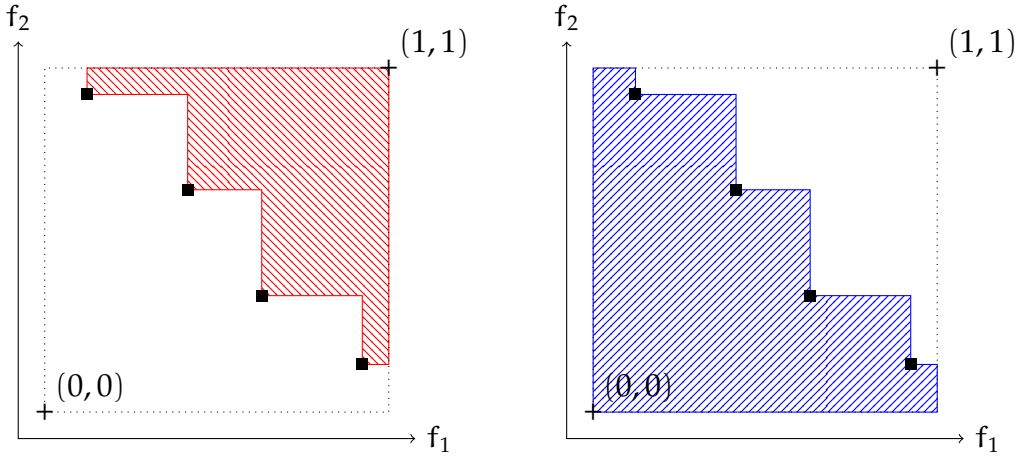


Figure 1.1 – Normalised unary hypervolume indicator (left: HV; right: $1 - \text{HV}$)

This variant can be seen as the indicator aiming to minimise the volume of non-dominating space, in contrast to the hypervolume which aims to maximise the volume of dominating space.

Finally, while primarily being an accuracy performance indicator, hypervolume also captures information about the diversity of the front of solutions, which is one of the reasons making the popularity of hypervolume.

1.2.3 Δ Spread

As a complementary indicator, we use a variant of spread to capture the distributional properties of the Pareto set. Figure 1.2 shows two sets of solutions: one well-distributed (squares) and the other unbalanced (circles).

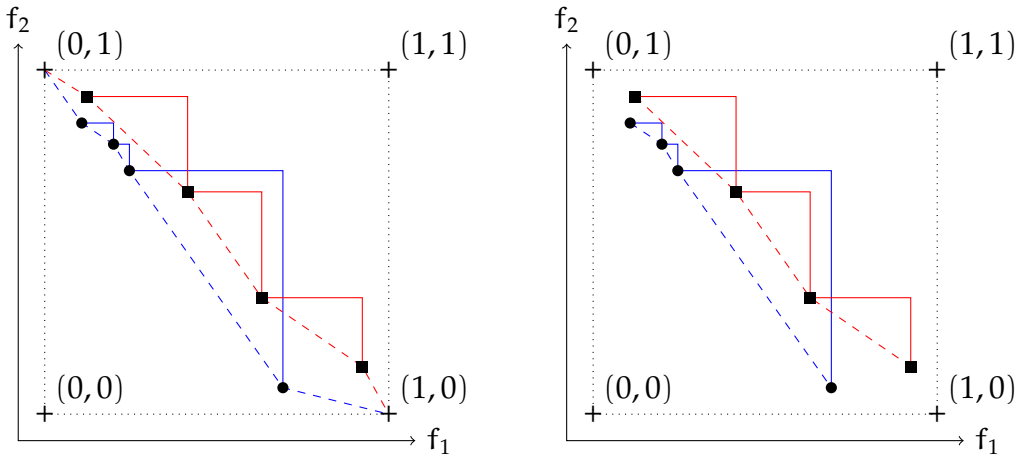


Figure 1.2 – Normalised (left) Δ and (right) Δ' spread indicators

The Δ spread indicator (Deb et al., 2002) is defined by Equation 1.6 for a given Pareto set S, ordered regarding the first criterion, where d_f and d_l are the Euclidean distances between the extreme positions $(1,0)$ and $(0,1)$, respectively, and the boundary solutions of S, and $\bar{d}$ denotes the average over the Euclidean distances d_i for $i \in [1, |S|-1]$ between adjacent solutions on the ordered set S.

$$\Delta := \frac{d_f + d_l + \sum_{i=1}^{|S|-1} |d_i - \bar{d}|}{d_f + d_l + (|S|-1) \cdot \bar{d}}, \tag{1.6}$$

This indicator is to be minimised; it takes small values for large Pareto sets with evenly distributed solutions, and values close to 1 for Pareto sets with few or unevenly distributed solutions.

In practice the distances between the extreme solutions of the set S of the points $(1,0)$ and $(0,1)$ are much bigger than the distances between consecutive solutions of S. This is especially true if the reference points of the normalisation are taken conservatively, which is the case in the current context of algorithm configuration where the normalisation needs to be fixed before the execution of the algorithm.

In consequence, we use the following variant instead, denoted as Δ', and defined simply by Equation 1.7, where the two distances d_f and d_l to the reference points $(1,0)$ and $(0,1)$ have been removed, thus removing spread information and making this variant a solely distance-based distribution performance indicator.

$$\Delta' := \frac{\sum_{i=1}^{|S|-1} |d_i - \bar{d}|}{(|S|-1) \cdot \bar{d}}, \tag{1.7}$$

This indicator keeps the property of Δ of having values independent to the problem instance being solved.

1.3 Permutation Problems

In this section, three permutation problems are presented; they will then be used in the following chapters. All three problems share the same solution representation (or *genotype*), a fixed-size permutation, which enables the analysis of the same algorithm and strategies on very diverse situations, as each problem lead to very different solution models (or *phenotypes*) and objectives.

1.3.1 Permutation Flow Shop Scheduling Problem

The permutation flow shop scheduling problem (PFSP) is a classical combinatorial optimisation problem, and one of the best-known problems in the scheduling literature since it models several typical problems in manufacturing. It involves a set of

n jobs $\{J_1, \ldots, J_n\}$ that need to be scheduled on a set of m machines $\{M_1, \ldots, M_m\}$. Each job J_k need to be processed sequentially on each of the m machines, with fixed processing times $(p_{k,1}, \ldots, p_{k,m})$. Finally, machines are critical resources that can only process a single job at a time. For the *permutation* flow shop scheduling problem, each machine process the jobs in the same order, so that a solution may be represented by a permutation of size n. The completion times $C_{i,j}$ for each job on each machine for a given permutation $\pi = (\pi_1, \ldots, \pi_n)$ are computed using Equation 1.8 to Equation 1.11.

$$C_{\pi_1,1} := p_{\pi_1,1} \tag{1.8}$$

$$C_{\pi_1,j} := C_{\pi_1,j-1} + p_{\pi_1,j} \quad \forall j \in \{2, \ldots, m\} \tag{1.9}$$

$$C_{\pi_i,1} := C_{\pi_{i-1},1} + p_{\pi_i,1} \quad \forall i \in \{2, \ldots, n\} \tag{1.10}$$

$$C_{\pi_i,j} := \max(C_{\pi_{i-1},j},\ C_{\pi_i,j-1}) + p_{\pi_i,1} \quad \forall i \in \{2, \ldots, n\} \quad \forall j \in \{2, \ldots, m\} \tag{1.11}$$

The completion time C_k of a job J_k is then simply its completion time on the last machine $C_{k,m}$. An illustration of a small permutation flow shop instance ($n = 3$ jobs, $m = 4$ machines) is given in Figure 1.3. It features examples of *waiting time*, e.g., on machine M_2 as job J_1 is still being processed on machine M_1, and an example of *idle time* for job J_2 on machine M_3 as the processing of job J_1 is not yet completed. The corresponding completions times of the three jobs J_1, J_2 and J_3 are 21, 23 and 11, respectively.

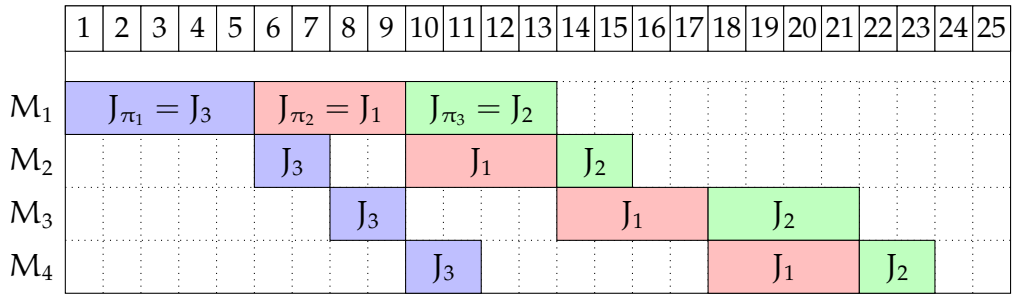


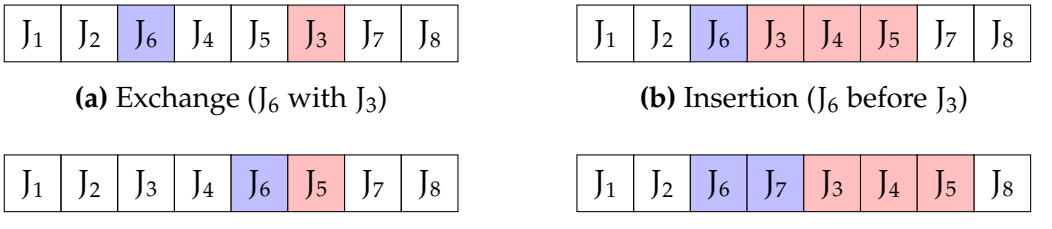
Figure 1.3 – Example of PFSP schedule for $n = 3$ jobs, $m = 4$ machines, and $\pi = (3, 1, 2)$

The most common objective to minimise on flow shop scheduling problem is the makespan (i.e., the total completion time of the schedule $C_{\pi_n,m}$, here 23 in Figure 1.3). Other classical objectives include the total flow time (i.e., the sum of completion times, and consequently their average), or when due dates are introduced, the maximum or total tardiness (Lawler et al., 1993). Weighted variants of these objectives are also common (Dubois-Lacoste et al., 2011b). In the following, we will study two bi-objective PFSP, with first a classical combination of

makespan and total flow time (Dubois-Lacoste et al., 2011c; Bezerra et al., 2014). Because the makespan and total flow time objective are quite correlated, we will also study a second bi-objective PFSP, obtained by considering a combination of two makespan objectives computed with hand-tuned correlated processing times (Kessaci-Marmion et al., 2017).

The classical literature PFSP instances are given by Taillard (1993). They are randomly generated with independent processing times following the uniform distribution $\mathcal{U}[1;99]$. There are 110 Taillard's instances, with number of jobs $n \in \{20, 50, 100, 200, 500\}$ and number of machines $m \in \{5, 10, 20\}$, 10 instances being available for each valid combination (n, m).

Classical PFSP neighbourhoods include the exchange neighbourhood, where the positions of two jobs are exchanged, and the insertion neighbourhood, where one job is reinserted at another position in the permutation. It was shown that for multi-objective local search algorithms the hybrid neighbourhood defined as the union of the exchange and insertion neighbourhoods lead to better performance than considering a single neighbourhood (Dubois-Lacoste et al., 2015). Other common operations on schedules include the adjacent swap, special case of exchange when the two jobs are necessary adjacent – thus highly reducing the computational cost but also the interest of such an operation – or the *block-move*, generalisation of both the insertion and the adjacent swap when the positions of two adjacent *subsets* of jobs are exchanged. Block-moves are less commonly used as they induce much bigger neighbourhoods. These operations are illustrated in Figure 1.4 starting from an ordered permutation.



(a) Exchange (J_6 with J_3)

(b) Insertion (J_6 before J_3)

(c) Adjacent swap (J_6 with J_5)

(d) Block-move ($\{J_6, J_7\}$ with $\{J_3, J_4, J_5\}$)

Figure 1.4 – Common PFSP schedule operations on an ordered permutation

1.3.2 Travelling Salesman Problem

The travelling salesman problem (TSP) is one of the most widely studied combinatorial optimisation problems, optimising the tour of an hypothetical salesman

needing to visit once each of the n cities of a given set $\{C_1, C_2, \ldots, C_n\}$. The TSP can be defined by a complete weighted graph G whose n nodes represent the cities, while edges corresponds to direct paths between cities. In the symmetric TSP, this graph is undirected, and edge weights correspond to distances between cities. Given a TSP instance G, the goal is to determine a tour passing through every city exactly once, such that the total distance travelled is minimised, i.e., a minimum-weight Hamiltonian cycle in G. This cycle corresponds to a permutation of the n cities. There is no real meaning of the "beginning" or "direction" of the tour – e.g., for an instance with 4 cities the solutions $(1, 2, 3, 4)$ and $(2, 1, 4, 3)$ both map to the same tour – so permutations may need to be normalised (e.g., requiring to begin the tour with C_1 and to visit C_2 before C_3) so that each possible tour has a unique representation. An illustration is given in Figure 1.5 for a small instance of $n = 8$ cities.

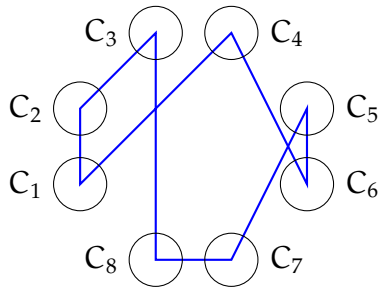


Figure 1.5 – Example of TSP tour for $n = 8$ cities and $\pi = (1, 2, 3, 8, 7, 5, 6, 4)$

Multi-objective TSP instances can easily be obtained by considering either correlated additional costs, such as distance and travel time, or simply multiple independent uncorrelated costs. A benchmark set of Euclidean instances (available online[1]) has been widely used in the literature to assess the performance of bi-objective TSP algorithms. These instances were constructed by combining two independently generated distance matrices, the two objectives being therefore uncorrelated. In addition to these instances, we will also consider variably correlated instances, by first generating a set of cities, then duplicating it and slightly moving each city, to obtain two correlated matrices of Euclidean distances (Kessaci-Marmion et al., 2017).

A classical neighbourhood for the travelling salesman problem is the 2-*opt* (or *pairwise exchange*) neighbourhood, where two tours are neighbours if, and only if, one can be obtained from the other by removing two non-adjacent edges reconnecting the resulting tour fragments by two other edges. It has the visual property

[1] https://eden.dei.uc.pt/~paquete/tsp/#Exp2

of repairing routes that cross themselves. An illustration is given by Figure 1.6, in which edges (C_1, C_4) and (C_3, C_8) are removed and reinserted, reordering the tour of cities $\{C_1, C_2, C_3\}$ to remove the crossing. Note that the 2-opt method is a special case of the more general k-opt method (or *Lin–Kernighan* method, Kernighan and Lin (1970); Lin and Kernighan (1973)), but that using $k > 2$ usually leads to a much bigger neighbourhood size and thus far higher computational cost. The permutation neighbourhoods (e.g., exchange, insertion; see Figure 1.4) can also be used on the TSP, albeit much less used and efficient than the k-opt methods.

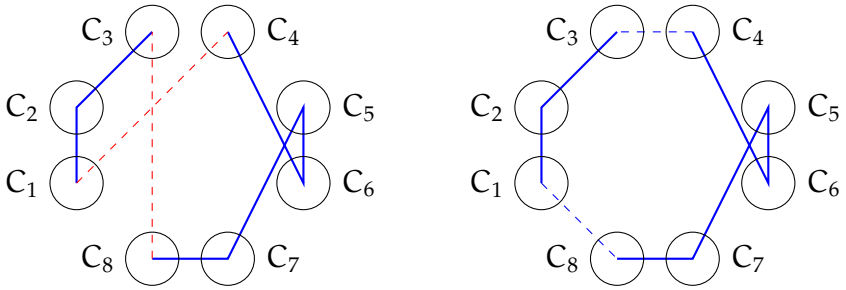


Figure 1.6 – Example of 2-opt recombination (left: removal; right: reinsertion)

1.3.3 Quadratic Assignment Problem

The quadratic assignment problem (QAP) involves assigning a set of n facilities $\{F_1, F_2, \ldots, F_n\}$ to a set of n given locations $\{L_1, L_2, \ldots, L_n\}$, minimising a cost function that depends both on the distance between locations and the flow between the facilities assigned to these locations. A solution is a permutation $\pi = (\pi_1, \ldots, \pi_n)$, where each location L_k is associated to the facility F_{π_k}. The objective is then to minimise the cost C associated to the solution, defined by Equation 1.12 for a given permutation π, with $w_{i,j}$ the flow between facilities F_i and F_j, and $d_{i,j}$ the distance between locations L_i and L_j.

$$C := \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} d_{\pi_i,\pi_j} \tag{1.12}$$

Figure 1.7 shows two solutions of a small QAP instance ($n = 8$), in which for clarity most of the flow is supposed equal to 0 and is not represented (otherwise the graph would necessarily be complete). This figure highlights that the locations are fixed, the permutation only changing the mapping according to which facilities are associated to locations.

Similarly to the TSP, multi-objective QAP instances can be obtained by considering either correlated additional costs or multiple independent uncorrelated

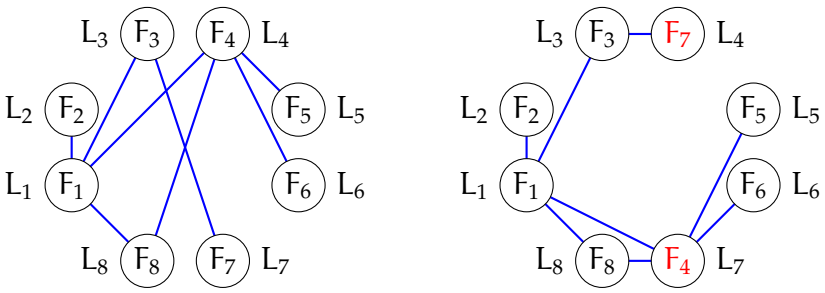


Figure 1.7 – Examples of QAP pairings for $n = 8$ locations (thus facilities); left: $\pi = (1, 2, 3, 4, 5, 6, 7, 8)$; right: $\pi = (1, 2, 3, 7, 5, 6, 4, 8)$

costs. To our present knowledge, there are publicly available multi-objective QAP instance generators (Knowles and Corne, 2003) but no widely recognised multi-objective QAP benchmarks in the literature. To obtain bi-objective instances, we consider two correlated flow matrices, both tied to a single distance matrix. As for both previous problems, it enables for the correlation between the two objectives to be manually adjusted (Kessaci-Marmion et al., 2017).

The neighbourhood commonly used on QAP is the exchange neighbourhood (see Figure 1.4). Indeed, while other neighbourhood such as insertions or k-opt operations can be used, they have here no real meaning as positions in the permutation are not related in any way to an ordering of the facilities, but solely their mapping to the different locations, which stay fixed.

Chapter 2

Automatic Algorithm Design

> If I have seen further it is by standing on the shoulders of Giants.
>
> Isaac Newton

In this chapter, we present the research field of *automatic algorithm design* (AAD). After some necessary preliminary definitions, we give an overview of the different research fields of the literature, such as automatic algorithm control, algorithm selection, hyper-heuristics, and state of the art methods that are related to AAD. Then, we propose and discuss a new taxonomy of AAD to better group under a single label these research fields.

In the process of solving problems and obtaining solutions, one generally needs to go through many analysis, decisional and computational steps, steps that may be automatised with the use of *algorithms*. The usual procedure is to: (i) formally define the problem to solve, (ii) either choose an existing algorithm or design a new specific one to solve the problem, (iii) run the algorithm on the particular input of interest, to finally (iv) obtain relevant solutions. These steps describe an algorithm, for the theoretical higher-order *"problem solving"* problem. More particularly, the second step is related to answering the question "which algorithm should I use to solve my problem", which can also be better worded as "what is the best algorithm to solve this problem". While these questions are generally left to human expertise, they can be tackled automatically, through what we call *automatic algorithm design*.

2.1 Preliminaries

In order to better contextualise the automatic algorithm design research field and better compare the different algorithm design approaches, we first give the necessary definitions and notions regarding to algorithms and design choices.

Problem: a description, semantic, and formal definition of the problem and the possible solutions (e.g., the travelling salesman problem (TSP)).

(Problem) instance: the particular data, relative to a given problem, over which an algorithm is used to obtain final solutions (e.g., the graph of distances corresponding to a list of cities for the TSP).

(Problem) instance class: a subset of problem instances sharing common characteristics (e.g., instances of the same size, sharing a similar structure, or originating from the same source).

(Problem) instance feature: a measurable property or characteristic of the instance. Note that selection or extraction of instance features are in themselves very hard machine learning problems.

Algorithm: a complete and unambiguous description of how to obtain solutions for a given problem instance; in the following, we also identify an algorithm to the decisional schedule it induces.

Parameter: a decision point in an algorithm reflecting a design choice.

Parameter value: the value associated to a given parameter.

Parameterised algorithm: an algorithm exposing design choices as parameters; a set of default parameter values is usually available.

(Algorithm) configuration: the set of parameter values necessary to run a parameterised algorithm, by specifying a setting to each of its parameters.

Virtually all algorithms are based on a succession of design choices that enables them to successfully run and achieve results. If some of these design choices may be left to the user discretion in the form of parameters, most of them are statically defined in the algorithm following early decisions of the algorithm designer. Each of these design choices ultimately heavily impact the performance of the algorithm. Nowadays the tendency is hopefully to propose frameworks more open in their design. Indeed, with more available parameters, they can potentially reach far better performance if adequately configured. Eventually, almost every design choice could potentially (and probably should) be automatically optimised (Hoos, 2012).

Parameters are usually classified into three categories:

Categorical parameters, which have a finite number of unordered discrete values, often used to select between alternatives mechanisms (e.g., to select a specific strategy, or to enable or disable a mechanism);

Integer parameters, which have discrete and ordered domains (e.g., to specify a number of iterations or the size of a set of solutions); and

Continuous parameters, that take numerical values on a continuous scale (e.g., to set a probability or percentage threshold).

The distinction between *structural parameters* (i.e., categorical parameters, or integer parameter with high impact on the comportment of the algorithm) and *behavioural parameters* (i.e., other integer and continuous parameters) is also common, under many different appellations: *qualitative/quantitative*, *symbolic/numeric*, *categorical/numerical*, *component/parameter*, *nominal/ordinal*, or *categorical/ordered* (see Eiben and Smit, 2012). In addition, some *conditional parameters* may only be used depending on the setting of other parameters, and some combinations of parameters may be forbidden when they are known to lead to incorrect or undesirable algorithmic behaviour.

2.2 Overview

Offline approaches are usually opposed to *online* approaches (Hamadi et al., 2012), following the taxonomy of Eiben et al. (1999) of *parameter setting* (Figure 2.1), which divides *parameter tuning* approaches, which aim to find good values for the parameters *before* the run of the algorithm, and *parameter control* approaches, which start the run with initial parameter values that are then controlled and adapted *during* the run.

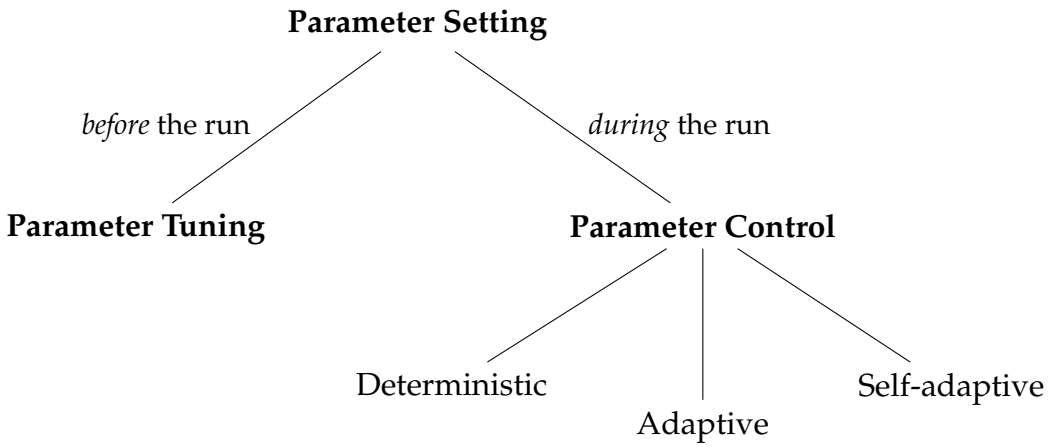


Figure 2.1 – Eiben et al. (1999) parameter setting taxonomy

Offline approaches (parameter setting) focus on getting the best possible algorithm prior of its actual use on the input data; once the algorithm is fixed it runs following its specification. They can therefore be seen as *prediction*-based approaches. Conversely, online approaches (parameter control) do not predict the best possible algorithm but rather focus on optimising its configuration during its execution. In other words, online approaches try to adapt the schedule of decisional and computational steps of the algorithm using its impact on the input data, while offline approaches try to predict the entire fixed schedule. Of course, if this theoretically leads to much more efficient algorithms, general automatic adaption in algorithms is orders of magnitude more complex than *just* optimising over all possible static algorithms. This distinction between offline and online approaches deeply shaped the research on automatic algorithm design.

Finally, to quote Karafotias et al. (2015) on the tuning and control of evolutionary algorithms: 'From a practical perspective, tuning is an absolute 'must' [...] Meanwhile, parameter control is more of a *neat-to-have* than a *need-to-have*. '

In the following, we present and discuss many of the research fields related to AAD, and how they differ between each others. While fields such as algorithm configuration and parameter control directly falls under Eiben taxonomy, we observe than many others such as algorithm selection or hyper-heuristics also closely relates to the same problematic: automatically devising better algorithms for given problems.

2.2.1 Algorithm Selection

Algorithm selection focuses on understanding the relation between algorithm performance and problem instance features. The basis is that, for a given set of problem instance classes, there is a corresponding set of complementary algorithms that can be used to improve overall performance.

Formally (Rice, 1976), the algorithm selection problem consists in, given a portfolio $\mathcal{P}$ of algorithms $A \in \mathcal{P}$, a set of instances $\mathcal{I}$, and a cost metric $o : \mathcal{P} \times \mathcal{I} \to \mathbb{R}$, optimising a mapping $s : \mathcal{I} \to \mathcal{P}$ across all instances of $i \in \mathcal{I}$, as given in Equation 2.1.

$$\begin{cases} \text{optimise} & \sum_{i \in \mathcal{I}} o(s(i), i) \\ \text{subject to} & s : \mathcal{I} \to \mathcal{P} \end{cases} \tag{2.1}$$

Because this problem optimises the performance on each instance of the set independently, algorithm selection is also sometimes called *per-instance* algorithm

selection. A simplified general workflow of algorithm selection is given in Figure 2.2., in which a selection tool construct the final mapping by iteratively providing an algorithm A and an instance i to a runner, whose role is to simply returns the subsequent performance.

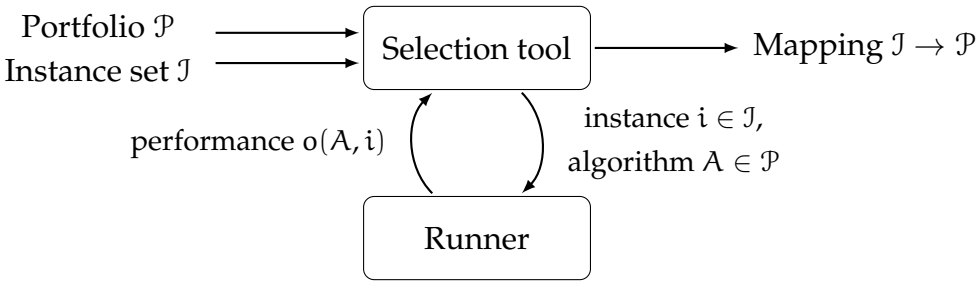


Figure 2.2 – Algorithm selection general workflow

Some of the most prominent algorithm selection tools include SATzilla (Xu et al., 2008), ISAC (Kadioglu et al., 2010), 3S (Kadioglu et al., 2011) and CSHC (Malitsky et al., 2013). A recent extensive survey on algorithm selection for combinatorial search problem can by found in Kotthoff (2016), which also keeps an up to date online literature summary on algorithm selection literature[1].

One extension of the traditional per-instance algorithm *selection* problem is per-instance algorithm *scheduling*, that associates to each instance not a single algorithm anymore, but a schedule of different algorithms. This extension allows to increase robustness, in particular regarding instances for which multiple algorithms might perform well. The algorithm schedules can be optimised globally for all instances (Hoos et al., 2015), determined for each instance relatively to algorithm performance on similar instances (Amadini et al., 2014), or used statically as a pre-solving mechanism before traditional algorithm selection (Kadioglu et al., 2010, 2011; Hoos et al., 2014). These approaches have been shown to be very efficient and to achieve strong performance on many algorithm selection scenarios (Lindauer et al., 2016).

2.2.2 Algorithm Configuration / Parameter Tuning

Algorithm configuration (or parameter tuning) focuses on getting the best performance of a given algorithm on a given distribution of instances, through modifications of its parameters. The algorithm being optimised is called the *target algorithm*, while the algorithm optimising the parameters of the target algorithm is

[1] https://larskotthoff.github.io/assurvey/

called the *configurator*. It can be seen as the automatic process to find the best "default" parameters for a given algorithm on a given instance class. As algorithm selection, algorithm configuration is an offline process. In the machine learning community, this problematic is also referred to as *hyperparameter optimisation* (e.g., Bergstra and Bengio, 2012).

Formally, the algorithm configuration problem consists in, given a parameterised target algorithm A, the space Θ of configurations of A, a distribution of instances $\mathcal{D}$, a cost metric $o : \Theta \times \mathcal{D} \to \mathbb{R}$, and a statistical population parameter E, optimising the aggregated performance of the target algorithm A across all instances $i \in \mathcal{D}$, as given in Equation 2.2 (in which A_θ denotes the algorithm obtained by associating the configuration θ to the target algorithm A).

$$\begin{cases} \text{optimise} & E[o(A_\theta, i),\ i \in \mathcal{D}] \\ \text{subject to} & \theta \in \Theta \end{cases} \tag{2.2}$$

Algorithm configuration supposes that the limit implied by Equation 2.2 exists and is finite. In practice, a machine learning approach is taken, by considering a finite set of training instance $\mathcal{I}$ instead, and validating the performance of the final configuration on a separate set of instances. The most commonly used statistical population parameter is the simple average of the performance of the target algorithm. A simplified general workflow of algorithm configuration is given in Figure 2.3.

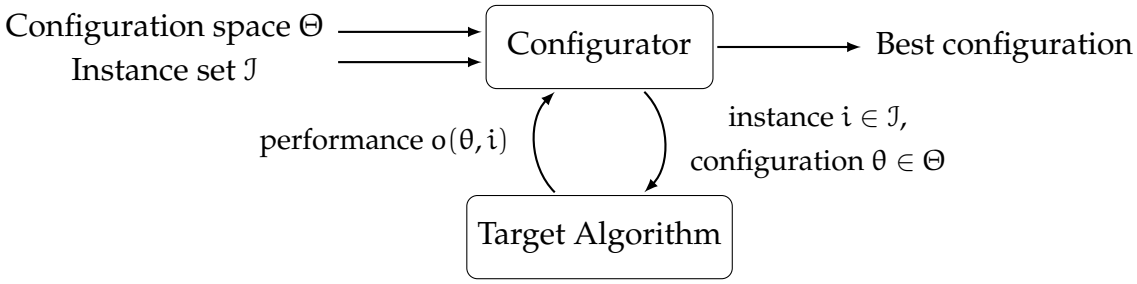


Figure 2.3 – Algorithm configuration general workflow

While in practice the terms *algorithm configuration* and *parameter tuning* have been in the past commonly used interchangeably, more recently the former is preferred when the parameter space mostly contains categorical parameters and the latter when it mostly contains numerical parameters.

Many different types of automatic configuration tools can be found in the literature. For example, irace (López-Ibáñez et al., 2016), one of the most popular configurator, uses statistical racing (Birattari et al., 2002; Balaprakash et al., 2007)

to find efficient configurations while discarding the one statistically outperformed. CALIBRA (Adenso-Díaz and Laguna, 2006), as well as ParamILS (Hutter et al., 2007, 2009), are based on iterative search. SPO (Bartz-Beielstein et al., 2005), or SMAC (Hutter et al., 2011), are other examples of configurators that build and refine a model for the parameter values (see also Bartz-Beielstein and Markon, 2004). Finally, GGA++ (Ansótegui et al., 2015) is a model-based configurator based on GGA (Ansótegui et al., 2009), an anterior configurator itself based on genetic algorithms.

2.2.3 Parameter Control

Parameter control focuses on adapting the parameter values of the running algorithms during its execution, rather than only using initial values that would otherwise stay fixed during its whole run. It is based on the observation that there is no reason for a specific parameter value to be and stay optimal the entire execution of the algorithmic process.

The classical taxonomy of parameter control algorithms is given by Eiben et al. (1999, 2007), and categorises three types of algorithms in which adaptation takes place: deterministic, adaptive, and self-adaptive algorithms. While it originally only applies on evolutionary algorithms, it is now used much more broadly.

Deterministic algorithms: parameters values are altered using deterministic rules. In other words, the adaptation is independent of some feedback from the search process, and only uses predetermined schedules (e.g., based on the number of iteration, or the running time elapsed).

Adaptive algorithms: parameters values are altered using some form of feedback from the search process. Decisions may involve credit assignment based on the performance linked to the parameter values.
An additional subdivision of adaptive algorithms into *functionally-dependant* algorithms and *self-adjusting* algorithms can also be found, based on whether the parameter values can be decided looking at the current state of the algorithm only, or if they depend on the success of previous iterations (Doerr and Doerr, 2015).

Self-adaptive algorithms: parameters values are encoded into solution genotypes and evolved naturally during the search process.

Recent and extensive surveys on parameter control in evolutionary algorithms can be found in Karafotias et al. (2015); di Tollo et al. (2015); Aleti and Moser (2016). While many methods have been proposed in the literature, most of them are tied

to specific algorithms (e.g., evolutionary algorithms) or specific parameters of specific algorithms (e.g., controlling the population size of an evolutionary algorithm). In this thesis, we are interested in generic, parameter independent, adaptive control mechanisms, among which four types, increasingly sophisticated, of generic methods can be distinguished.

Formula and rules: albeit typically strongly tied to specific contexts, formula and rules can in practice be devised regardless of the specific parameter or algorithm. Rules can be based on time, number of iterations, or any feedback from the search; they may be based on theoretical results or on intuition; they can have absolute effects (i.e., using specific values) or relative effects (i.e., modifying the parameter with regard to its current value (e.g., the $^1/_5$th rule; Schumer and Steiglitz, 1968).

Probability-based decisions: using feedback from the search, it becomes possible to associate to each parameter value a reward. Mechanisms such that the probabilistic rule-driven adaptive model of Wong et al. (2003), probability matching (Thierens, 2005), and adaptive pursuit (Thierens, 2005) all base their decisions on probabilities computed from search feedback.

Multi-armed bandits: similarly to probability-based decisions mechanisms, multi-armed bandit approaches can be used to take decisions by considering each parameter value as a distinct arm, thus transforming the problem into a known probability theory one. The dynamic multi-armed bandit of Costa et al. (2008); Maturana et al. (2009) and the adaptive range parameter selection mechanism of Aleti and Moser (2011) are examples of such multi-armed bandits algorithms applied to automatic algorithm design.

Reinforcement learning: finally, reinforcement learning (Sutton and Barto, 1998) can also be used by superimposing an additional layer of *states* in order for the decisions to better take into account the environment (e.g., Eiben et al., 2006), again enabling the use of known machine learning mechanisms.

Generic online mechanisms will be detailed in more depth in Chapter 4.

2.2.4 Hyper-heuristics

The term *hyperheuristic* originates from Cowling et al. (2000) in which it is used to describe 'heuristics to choose heuristics'. Burke et al. (2010) (see also Burke et al., 2013) proposed a much broader taxonomy for hyper-heuristic approaches, founded on two dimensions: the source of feedback during learning and the nature of the heuristic search space, detailed hereafter.

Offline learning: knowledge is gathered from a set of training instances, and hopefully generalises to unseen instances.
Online learning: learning is done while the algorithm is solving one particular instance of the problem.
No learning.

Heuristic selection: methodologies for choosing or selecting existing heuristics.
Heuristic generation: methodologies for generating new heuristics from components of existing heuristics.

Burke et al. (2010) also follow Hoos and Stützle (2004) to refine this second dimension according to the search paradigm of the hyper-heuristic and whether it uses either a constructive or a perturbative search process.

Construction heuristics: the search process considers complete candidate solutions and alters them by modifying one or more of their components.
Perturbation heuristics: the search process considers incomplete candidate solutions, in which one or more components are missing, and iteratively extends them.

2.2.5 Other Fields and Taxonomies

Other research fields are also related to AAD, while others taxonomies have already been proposed.

Battiti et al. (2008) proposed definitions of both *reactive search* and *intelligent optimisation*. While the latter broadly corresponds to applications of machine learning strategies in heuristics, the former focuses on integration of machine learning techniques in local search heuristics for solving complex optimisation problems.

Autonomous search systems (Hamadi et al., 2012) already encompass most of the others research fields and taxonomies. Most importantly, they generalise the online–offline distinctions of Eiben et al. (1999) to fields outside parameter control. The taxonomy focuses around building singular monolithic systems, able without outside expert knowledge to find solutions to any problem, internally using one or many algorithm design tools and techniques. In other words, an autonomous search system is an easy-to-use interface for end-users, that aims to minimise technical interaction and knowledge by internally automatising configuration and solving processes.

Finally, *genetic programming* and *genetic improvement* are related research fields that apply traditional optimisation techniques (e.g., genetic algorithms) to software

engineering problems in order to improve existing software (Langdon, 2015; Petke et al., 2017). Examples of goals include optimising performance (e.g., quality, running time, memory or energy consumption), but also fixing program behaviour.

2.2.6 Multi-objective Automatic Design

All of the notions presented before were originally proposed as single-objective tools, to optimise the performance of single-objective algorithms. In order to apply such tools to multi-objective algorithms, one would need to use a single performance indicator, such as the hypervolume resulting of the final set of solutions (e.g., Dubois-Lacoste et al., 2011a).

However, the last few years have seen the development of many different tools either based on multi-objective optimisation or specifically designed for multi-objective optimisation. Among others, we can cite MOSAP (Horn et al., 2016), a multi-objective algorithm selection framework, S-Race and SPRINT-Race (Zhang et al., 2015, 2016, 2018), extending statistical racing for model selection according to multiple criteria, and MO-ParamILS (Blot et al., 2016), a multi-objective configurator for multi-objective configuration we proposed. MO-ParamILS will be presented and studied in depth in Chapter 5.

2.3 Overall Automatic Design Taxonomy Proposition

In this section, we propose a new taxonomy of automatic algorithm design. It follows and expends the temporal viewpoint already present in the Hamadi et al. (2012) taxonomy, while proposing a second, transverse, viewpoint based on the algorithmic structure of the elements being optimised. An important motive for this taxonomy is to propose a taxonomy for researchers focused on the design tools and techniques themselves, as autonomous search systems focus more to giving an autonomous black-box to non-technical end-users.

2.3.1 Temporal Viewpoint

The first point of view of our taxonomy is based on *when* are applied the algorithm design tools. Eiben et al. (1999), then Hamadi et al. (2012), separate tools that are applied *before* and *after* the first actual computational step on the given problem instance. Tools that are used to choose an algorithm and its parameters are classified as *offline* tools, while techniques used to adapt the algorithm or its strategies during the execution are classified as *online* techniques.

We propose to distinguish three phases in this timeline, rather than only two. First, we have tools that only require a description of the problems and the classes of instances that will be tackled, such as the learning process of algorithm selection and parameter tuning. More precisely, they do not require to know the specific instance that will be solved in the future, and they are used to obtain generalisations and to give predictions. We say that they only use *problem features*.

Then, we have tools or recommendations that use the specific features of the instance to solve. For example, the application of the mapping found using algorithm selection, or more generally the choice of parameter values following the algorithm designer recommendations (e.g., parameters with recommended values depending of the instance size). We say that they use *a priori features*.

Finally, following Eiben et al. (1999), the last phase begins when the first computational step starts the search. *Search features*, directly linked to the algorithm progression and the solutions considered, are then available to be used.

One of the advantages of separating the *online* phase into two distinct phases is first to better compare algorithm configuration to algorithm selection, as it splits the latter into two parts the machine learning first, then the application of the resulting mapping. Furthermore, it also enables to better include recommendations and rules based on manual preliminary experiments or simply intuition, that wouldn't otherwise be considered as not suitable as generic design tools.

2.3.2 Structural Viewpoint

The second point of view of our taxonomy is based on *what* is manipulated by the algorithm design tools, and more precisely *how much* of the solving process can be modified. On the one hand, parameter tuning mostly deals with numerical parameters, for which it is usually expected to imply very small and controlled variations in performance. Indeed, categorical parameters usually lead to much more stronger variation in performance. On the other hand, when selecting different algorithms, they can possibly be based on entirely different techniques and induce completely different performance profiles (which is incidentally the main motivation of algorithm selection).

These differences in expectation explain why some similar research fields were developed based on different assumptions and leading to very distinct principles. For example, why algorithm selection require problem features to compute a mapping of which algorithm is better on which type of instances, or why some efficient parameter tuning techniques can use local search techniques. However, both ends of the spectrum ultimately deal with the same problem of how to obtain what is

predicted to be the best algorithm to use on a given instance.

Indeed, parameter tuning, algorithm configuration, and algorithm selection can all be generalised into a broader problem, given in Equation 2.3, in which $\mathcal{D}$ is a distribution of instances, $\mathcal{A}$ is the algorithm space (e.g., obtained by a description of a given algorithm parameters values, or given by a portfolio), E is a statistical population parameter, and o is the (possibly multivariate) cost metric ($o : \mathcal{A} \to \mathbb{R}^n$, with $n \geqslant 1$).

$$\begin{cases} \text{optimise} & E[o(s(i), i),\ i \in \mathcal{D}] \\ \text{subject to} & s : \mathcal{D} \to \mathcal{A} \end{cases} \tag{2.3}$$

Equation 2.2 (parameter tuning; algorithm configuration) is a specialisation of Equation 2.3 when the mapping is supposed to be constant (i.e., a single configured algorithm A_θ is sufficient to achieve optimal performance over the distribution $\mathcal{D}$ of instances). To our present knowledge, most parameter tuning and algorithm configuration scenarios either suppose that the distribution of instances is homogeneous to a single class of instances, or nevertheless aim for a single "general" configuration. Similarly, Equation 2.1 (algorithm selection) is a specialisation of Equation 2.3 when the distribution of instances is reduced to a simple set of instances. In both cases, either all instances are supposed to be part of a single instance class, or each individual instance is optimised separately; the task of figuring out the instances classes is left to the expert knowledge of the end-user. Tools able to simultaneously both analyse the distribution to determine instances classes and optimise a mapping over these classes are, to the best of our knowledge, yet to be proposed.

Equation 2.3 can in theory be further refined into Equation 2.4 by considering that (i) the algorithm output itself is to be minimised, without the need of an additional cost metric (i.e., to optimise directly $a(i)$ instead of $o(a, i)$, with $a \in \mathcal{A}$ an algorithm) and that (ii) there is no need to explicitly first select an algorithm to then apply it on the instance (i.e., optimising $f(i) = s(i)(i)$).

$$\begin{cases} \text{optimise} & E[f(i),\ i \in \mathcal{D}] \\ \text{subject to} & f : \mathcal{D} \to \mathbb{R}^n \end{cases} \tag{2.4}$$

This definition is closer to the idea of autonomous search systems proposed in Hamadi et al. (2012) as it see the entire problem as a black-box system and does not expose the underlying selected algorithm to the end-user.

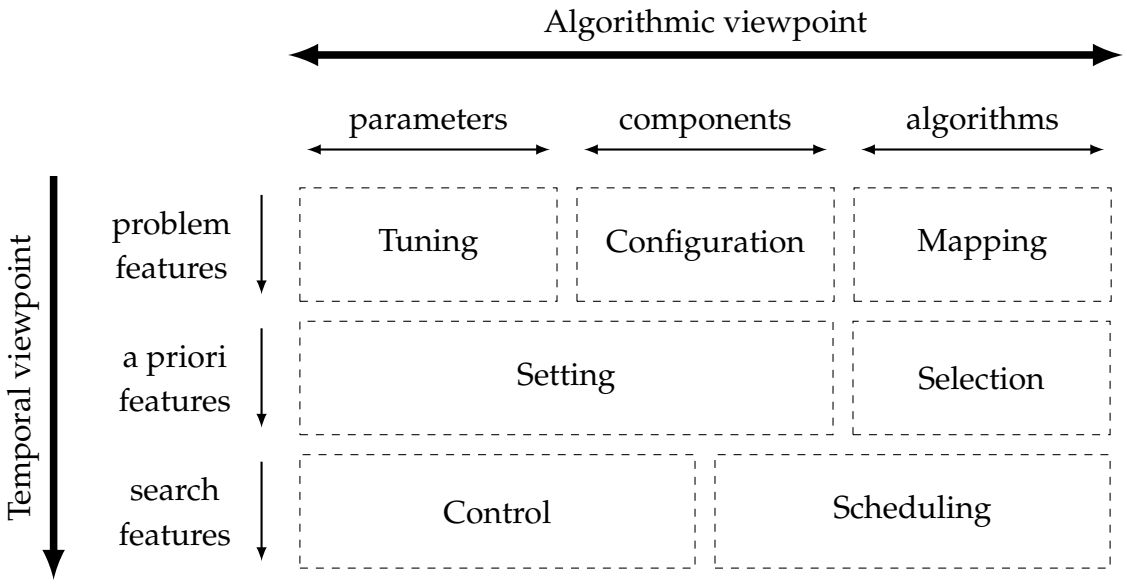


Figure 2.4 – Algorithm design overview

This viewpoint can rightfully be seen as more or less artificial. Indeed, to give some examples: a tool designed to deal with categorical parameters can be applied on continuous parameters after discretisation; very different algorithms can be seen as a single algorithm with a main, rather drastic, categorical parameter; and categorical parameters can be arbitrarily ordered and seen as integer parameters. However, it can be expected that in these cases the overall performance of the algorithm design tool will be worse than if the more suitable tool was used instead.

Under certain circumstances these different approaches can be used simultaneously and achieve even increased performance: e.g., the AutoFolio (Lindauer et al., 2015) selection tool relying on the automatic configurator SMAC (Hutter et al., 2011) to seed the portfolio of claspfolio 2 (Hoos et al., 2014), or Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2017) simultaneously selecting the WEKA[2] learning algorithm and optimising its parameters.

2.3.3 Overview

Figure 2.4 illustrates our taxonomy proposition and describes several sub-problems of the algorithm design problem, clarified hereafter.

First, we describe the offline sub-problems, that correspond to both problem and a priori features.

Tuning: the algorithm is already known, but it needs to be adapted for the given

[2] https://www.cs.waikato.ac.nz/ml/weka/

instance class of interest. Most parameters are either continuous or behavioural. Performance is usually predictable between similar configurations.

Configuration: the chosen algorithm has multiple available strategies or heuristics, that need to be set before the execution. Configuration includes categorical parameters that makes predictions less relevant, as a single parameter change can lead to very different (even possibly improper and unfeasible) algorithm behaviour and performance. Tuning may also be necessary, either during or after configuration.

Mapping: the algorithm is yet to be chosen, usually because not enough information on the future instances is known and they may be different enough to warrant having to study instances classes and to map each of them to an optimal algorithm. Evidently, configuration and tuning may also be necessary.

These three sub-problems all aim to *predict* what will be the best algorithm on the possible future problem instances. They should all be seen as machine learning process, with training data on past or artificial instances and testing generalisation on distinct unseen instances.

Setting: now that the instance of interest is known, the different parameters can be set using either the recommended configuration, basic rules (e.g., parameter values based on the instance size, or the presence or absence of some features), or obviously using the prediction resulting of specific tuning and configuration.

Selection: likewise, either following a choice based on literature recommendation or a previously obtained mapping, the algorithm and its parameters is chosen for execution.

As mentioned before, these distinctions are mainly there to easily include manual design processes that result of standard algorithm development or preliminary analyses. The only difference between *setting* and *selection* is that in latter also selects the algorithm, while it is already known in the former. This also enables us to keep the term *selection*, as *mapping* was preferred to designates the learning phase of *algorithm selection*.

Then, we describe the online sub-problems which can also use search features.

Control: starting from their initial values, continuous parameters can now be adapted during the execution of the algorithm, to better match their values with regard of the particular search context encountered. Specific control mechanisms can also handle generic categorical parameters, strategies, and heuristics

It enables many possible improvements, such as for example combining the strength of multiple strategies, delaying the strategy choice to after the start of the resolution.

Scheduling: the solving process can use different strategies at different moments of the search. Multiple distinct algorithms can now be used sequentially on the instance in order to multiply the chances of using one able very well adapted to quickly solve the instance, with the drawback of generally not being able to efficiently use the knowledge collecting a previous algorithm.

Note that if online mechanisms are not taken into account during the preliminary design phases, it is likely that the parameter values obtained are only optimal in average during the run, and not specifically optimal as initial parameter values for the online adaptation.

2.3.4 Additional Complexity Viewpoint

Our taxonomy proposition presented in Figure 2.4 does not include the classical parameter control taxonomy of algorithms proposed by Eiben et al. (1999). Instead, we propose the following definitions to update the original one and complement our taxonomy while also matching the learning categorisation of Burke et al. (2010).

Static algorithms: the description of the algorithm is not altered during the execution; every parameter that would was been considered for tuning or configuration has a fixed, final, value.

Dynamic algorithms: parameter values can change, but only according to random, predictive, and deterministic events, such as elapsed running time or number of iterations.

Adaptive algorithms: the algorithm can use explicit feedback from the search process to modify itself through control mechanisms, such as machine learning or credit assignment procedures.

The main differences between this taxonomy and the one of Eiben et al. (1999) are as follow. First, we include static algorithms into the taxonomy. Then, we use the term "*deterministic* algorithm" instead of "*dynamic* algorithm"; Eiben et al. (1999) already raised concerns that "the term 'deterministic' control might not be the most appropriate" and mentioned "'fixed' parameter control" as a possible preferred alternative. The adaptive definition is directly taken from the original one. Finally, as for now the self-adaptive distinction does not really seems to have meaning outside the scope of evolutionary algorithms and is therefore not included.

Finally, these three categories of algorithms directly correspond to the three categories of the learning dimension in Burke et al. (2010): static algorithms use no learning, dynamic algorithms use offline learning (i.e., knowledge from previous instances), and adaptive algorithms use online learning (i.e., knowledge from the current instance).

Part II

Multi-objective Local Search

> Equipped with his five senses, man explores the universe around him and calls the adventure Science.
>
> Edwin Hubble

Chapter 3

Unified MOLS Structure

> Everything must be made as simple as possible. But not simpler.
>
> Albert Einstein

In this chapter, we investigate multi-objective local search (MOLS) techniques in the literature.

We first give a detailed overview of local search techniques in multi-objective metaheuristics, following their historical development. Then, we survey and discuss the different MOLS strategies. Finally, we propose a unification of MOLS techniques into a general framework.

This chapter provides the necessary materials that are used in Chapter 4 to instantiate the different MOLS algorithms used in the following chapters: the static MOLS algorithm of Chapter 6, the adaptive MOLS algorithm of Chapter 7, and finally the MOLS configuration schedules of Chapter 8.

This chapter contributions are closely linked to the following publication:

- **Blot, A.**, Kessaci, M., and Jourdan, L. (2018b). **Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation.** *Journal of Heuristics*.

3.1 Preliminaries

3.1.1 Definitions

In this thesis, we focus on multi-objective combinatorial optimisation problems in which the search space is a *finite set* of solutions. These problems are generally $\mathcal{NP}$-

hard, making exhaustive numeration of the search space is generally not feasible for large instances.

For solving such problems, approximation algorithms that exploit the structure of the search space to iteratively find better and better solutions are used, such as local search algorithms. The idea behind local search algorithms is that small modifications in the representation of a solution may lead to either a small improvement or a small deterioration of its initial quality. The notion of *neighbourhood* is defined from this idea. A *neighbourhood operator* is a function that modifies part of a given solution that produces a new solution called a *neighbour*. The set of neighbours that can be generated from a given solution x defines the neighbourhood $\mathcal{N}(x)$ of x. This concept of neighbourhood gives a structure to the search space by connecting close solutions.

Several definitions result from this concept. When the neighbourhood $\mathcal{N}(x)$ of a solution x contains no improving neighbour, x then constitutes a *local optimum*. Note that a solution x can be a local optimum for a neighbourhood $\mathcal{N}_1$ without being a local optimum for another neighbourhood $\mathcal{N}_2$. Furthermore, there is no guarantee for a local optimum to be a global optimum, and it is generally not one. On the contrary, a global optimum is always a local optimum regardless of the neighbourhood considered since it is of the best possible quality.

Local search algorithms iteratively use neighbourhood operators to reach better and better solutions. Unfortunately, local optima are fundamentally detrimental to local search algorithms as there is usually no means to distinguish between local and global optima. Thus, to continue the search after having converged to a local optima, researchers have proposed multiple techniques, including either performing multiple random moves over the search space called kicks (e.g., Iterated Local Search, Lourenço et al. (2010)), accepting to move temporarily to worse solutions (e.g., Simulated Annealing, Kirkpatrick et al. (1983), doing a Tabu Search, Glover (1989); Glover and Laguna (1997)) or switching between several neighbourhood structures (e.g., variable neighbourhood search, Mladenović and Hansen (1997)).

In a multi-objective combinatorial optimisation context, this notion is extended by defining *Pareto local optima* (PLO) as the solutions whose neighbourhood contains no dominating neighbour, i.e., the solutions whose every neighbour is either dominated or incomparable. Likewise, there is no guarantee for a PLO to be a Pareto optimum, although Pareto optima are always PLO regardless of the neighbourhood considered. Furthermore, many, if not all, of the above techniques used to escape local optima in a single-objective context have been extended to multi-objective ones to deal with PLO.

3.1.2 Historical Development

Historically, local search algorithms have been initially designed to solve single-objective combinatorial optimisation problems and thus are themselves single-objective algorithms (Hoos and Stützle, 2004). Multi-objective local search (MOLS) algorithms are used on the same combinatorial problems, e.g., multi-objective travelling salesman problems, multi-objective scheduling problems (Jaszkiewicz, 2002; Basseur and Burke, 2007; Liefooghe et al., 2012; Dubois-Lacoste et al., 2015), and bioinformatics problems (Abbasi et al., 2015). The majority of the literature works focuses on bi-objective and tri-objective problems, while very fewer works tackle more than three objectives simultaneously. This is due to the nature of the induced search space; indeed, in these *many-objective* problems (Ishibuchi et al., 2008) solutions are much more often incomparable to each others, thus majorly hindering the neighbourhood exploration of MOLS algorithms.

The development of MOLS algorithms simultaneously occurred following two different directions. On the one hand, they were directly extended from some of the well-known and established single-objective algorithms (e.g., Serafini, 1994; Ulungu et al., 1995; Czyzak and Jaszkiewicz, 1996; Hansen, 1997). On the other hand, they were also either integrated into evolutionary algorithms as inner components or used as post-processing algorithms (e.g., Ishibuchi and Murata, 1996; Knowles and Corne, 1999; Talbi et al., 2001). Nowadays, the prominent MOLS algorithms in the literature have grown into the PLS algorithms, which are derived from the second type of MOLS algorithms.

In the following, we detail chronologically the development of these two algorithmic families before summarising their common characteristics.

Extensions of Single-objective Algorithms

Since local search algorithms have been originally designed for single-objective optimisation, they are single-trajectory algorithms, meaning that they follow a single solution: the *current* solution. Unsurprisingly, the first MOLS algorithms were extensions of these single-objective local search algorithms.

Simulated Annealing (SA) (Kirkpatrick et al., 1983) is a local search procedure that optimises a single solution, using a decreasing parameter, the temperature, to slowly converge to the global optimal solution. Serafini (1994) and Ulungu et al. (1995, 1999) have independently proposed the same algorithm, Multi-Objective Simulated Annealing (MOSA). Like in the original single-objective algorithm, a single *current* solution is considered and moved through the search space, while

a subsidiary set is used to store the potential Pareto optimal solutions. The current solution is updated by evaluating a single random neighbour and potentially accepting it with regard to rules based on probabilities, which are themselves based on whether the neighbour dominates, is dominated by or is incomparable to the current solution, and on weighted projections of the fitness function. Czyzak and Jaszkiewicz (1996, 1998) proposed the Pareto Simulated Annealing (PSA), in which, rather than a single current solution, a set of current solutions is used to converge into multiple optima at the same time. The diversity of having multiple current solutions is also used to guide the parallel searches in diverse directions. Engrand (1998) and then Suppapitnarm and Parks (1999) proposed other MOSA variants, in which the current solution is periodically replaced by one of the archived solutions (SMOSA). Other variants also include the Pareto Archived Simulated Annealing (PASA) by Suresh and Mohanasundaram (2004), the Archived Multi-Objective Simulated Annealing (AMOSA) by Bandyopadhyay et al. (2008) and those based on both Pareto dominance (PDMOSA) and weights (WMOSA) proposed in literature reviews by Suman (2003); Suman and Kumar (2006).

Tabu Search (TS) (Glover, 1989; Glover and Laguna, 1997) is a local search algorithm that uses an auxiliary set of solutions, the *tabu list*, to guide the search and escape local optima by preventing a *backward* move on the search space by banning the acceptance of neighbours too similar to recent considered solutions. In a TS local search, when a solution is explored, each of the solution's neighbours is evaluated and the best non-tabu one is selected to replace the current solution, even if it has a worse quality. The first multi-objective algorithm based on TS is the Multi-Objective Tabu Search (MOTS) proposed by Hansen (1997). It uses a set of *current* solutions and independently explores their neighbourhoods using an aggregation of the multiple objectives. After a given number of iterations, a *drift* strategy is applied: the set of solutions is updated by replacing one of the solutions by another one, both uniformly selected at random in it, to explore the whole front and not merely to focus on one part of the objective space. Other TS algorithms have been proposed, such as the one by Baykasoglu et al. (1999), which is based on a local search with an intensification memory to restart from when no more improving move is available, or the two algorithms proposed by Jaeggi et al. (2004, 2008), based on the Hooke and Jeeves move (MOTS) and path-relinking (PRMOTS), respectively. Multi-objective variants of scatter search using TS and path-relinking have also been proposed (Beausoleil, 2001; Molina et al., 2007).

Greedy Randomised Adaptive Search Procedure for Maximum Independent Set (GRASP) is a procedure originally presented by Feo et al. (1994) for single-objective optimisation problems. It has been extended by Vianna and Arroyo (2004) for multi-objective optimisation problems (GRASP-MULTI). Both algorithms

are multi-start metaheuristics that alternate two phases, the first being the construction of an initial solution, and the second being the iterative improvement of that solution through a local search procedure; the best overall solution (or set of solutions, for the multi-objective version) is then returned. The local search procedure simply replaces the current solution by any improving neighbour until there is no more improving neighbour. In the case of the multi-objective version, the different local search iterations use different aggregation weights and a list of all potential Pareto optimal solutions is automatically kept up to date. An extensive survey of multi-objective GRASP can be found in Martí et al. (2015), in which the authors propose a multi-objective GRAPS with path-relinking.

Lastly, Variable Neighbourhood Search (VNS) (Mladenović and Hansen, 1997) is a local search algorithm that solves the *local* optima problem by simply considering multiple other neighbourhoods. Indeed, a PLO regarding a given neighbourhood may have dominating neighbours regarding other neighbourhoods. Geiger (2008) proposed the Multi-Objective Variable Neighbourhood Search (MOVNS) based on PLST (Talbi et al., 2001) and the VNS methodology. Like in PLS, an archive is used to store all potential Pareto optimal solutions. In every iteration, both a solution and a neighbourhood, if they have not been explored yet, are selected uniformly at random, and then the solution is entirely explored and the Pareto set is updated using all the neighbours. Arroyo et al. (2011) proposed an interesting alternative to the MOVNS algorithm by adding a *shaking* mechanism: instead of generating the neighbourhood of a solution of the Pareto set, their algorithm generates the neighbourhood of a neighbour of the solution of the Pareto set.

In Evolutionary Algorithms

In addition to the single-objective local search algorithms, the development of MOLS algorithms also occurred at the same time jointly with the multi-objective evolutionary algorithms.

Evolutionary Algorithms (EAs) constitute a class of metaheuristics based on the iterative improvement of a set of solutions (namely, the *population*), which is often used to tackle multi-objective optimisation problems. EAs usually iterate both crossover and mutation techniques to improve the population. There are two types of hybridisation of multi-objective EAs with local search mechanisms. The first type integrates the local search inside the EA, either complementing or replacing the mutation, by using a local search on every solution of the population at each iteration. The local search is then generally a single-objective algorithm, based on an aggregation of the different objectives. The second type uses a MOLS

as a post-processing procedure at the end of the EA.

Ishibuchi and Murata (1996, 1998) first proposed the Multi-Objective Genetic Local Search (MOGLS), which hybridises a genetic algorithm with a single-objective local search by performing a local search on every solution generated at every iteration. During the local search, at most k neighbours of the current solution are produced and any improving neighbour is accepted. The crossover and the mutation strategies generate new solutions where the local search is applied using a new aggregation, i.e., the weights are randomly chosen. A *cellular* variant, called C-MOGLS, was proposed by Murata et al. (2000), which divides solutions into cells associated with weight vectors to guide the selection and local search procedures of the MOGLS. Jaszkiewicz (2002) proposed another Genetic Local Search (GLS) where the main differences are the way the solutions are selected for recombination and the local search aggregation, which uses a weight vector selected at random from a set of possible weight vectors. Knowles and Corne (1999, 2000a) proposed the Pareto Archived Evolutionary Strategy (PAES), an EA without crossover that only relies on local search techniques. PAES was presented as "the simplest non-trivial approach to a multi-objective local search procedure", and three versions were introduced. All versions maintain a single *current* solution to be explored, while the population takes the role of a Pareto set. In the simple (1+1)-PAES, the current solution is explored by generating a single neighbour. The neighbour replaces the current solution either if it dominates the latter or if it is in a less crowded region of the population. The population itself is updated in such a way that any dominated solution is discarded and the solutions of less crowded spaces replace the solutions of more crowded spaces. The (1+λ)-PAES variant generates λ neighbours of the current solution at every iteration, which are all considered for updating the population and replacing the current solution. The (μ+λ)-PAES variant replaces the current solution with a list of μ current solutions, one of which is explored, selected using a binary tournament. Knowles and Corne (2000b) also proposed the memetic-PAES (M-PAES), a variant periodically employing a crossover.

Talbi et al. (2001) also proposed a genetic algorithm hybridised with a MOLS procedure. The execution of the algorithm is divided into two separate steps, beginning with the execution of a genetic algorithm. Once the genetic algorithm finishes, every solution of the final population is then explored by generating and archiving every possible non-dominated neighbour, a procedure that is then iterated until the end of the algorithm. This second step of the algorithms is sometimes referred to as PLS-2 due to its similarities to the Pareto local search algorithms (see the following). To avoid confusions due to the numbering, we will refer to it as PLST instead.

Similarly, Moslehi and Mahnam (2011) proposed a hybridisation of a multi-objective particle swarm optimisation algorithm with a single-objective local search procedure (MOPSO+LS).

Pareto Local Search Algorithms

Following the steps of algorithms such as PAES (Knowles and Corne, 1999) and PLST (Talbi et al., 2001), which are based on Pareto dominance and use the population of the EA as a Pareto set, many more algorithms solely based on local search techniques have been designed that are not simple extensions of known single-objective local search algorithms. These simple extensions can still be seen as either single-trajectory algorithms or *multiple*-trajectory algorithms, as multiple solutions are simultaneously iteratively improved. On the contrary, the notion of trajectory is more blurred in the following algorithms, which are based more on improving iteratively the full archive (originally the evolutionary population) than on focusing on single solutions.

Paquete et al. (2004) and Angel et al. (2004) simultaneously proposed the first standalone local search algorithms: the Pareto Local Search (PLS) and the Bi-criteria Local Search (BLS). Both algorithms are very similar and are both known as the original PLS algorithm. Unlike in the previous EAs, in PLS algorithms, a *population* is called an *archive* and always consists of a Pareto set. At every iteration of the PLS algorithm, a single solution not yet considered is taken from the archive to explore its neighbourhood and all of its neighbours are used to update the archive.

Aguirre and Tanaka (2005) proposed the multiple multi-objective random bit climbers (moRBC) algorithm, which also follows a local search scheme. At each iteration, all the possible moves of the neighbourhood are generated. They are all successively applied to the current solution, which can be immediately replaced when a dominating neighbour is found. These authors proposed multiple versions of moRBC wherein incomparable neighbours may be accepted and a separate archive may be used for crowding or restarting purposes.

Using the idea of employing a separate standalone procedure to generate the initial solutions of the local search, Paquete and Stützle (2003) proposed the Two-Phase Local Search procedure (TPLS) for bi-objective optimisation problems. First, an initial solution is generated (originally, using a local search), considering the first objective only. Then, a local search is performed, starting from the resulting solution, but using an aggregation of the objectives slightly more oriented towards the second objective. This step is then repeated until the final local search considers the second objective only. Many variants of the TPLS procedure have been proposed, among which is the 2-Phase Pareto Local Search (2PPLS) procedure by

Lust and Teghem (2010), which hybridises the first step by constructing *potentially extreme supported efficient solutions* as the initial set of a PLS algorithm and an adaptive version of TPLS, likewise hybridised with a PLS algorithm, by Dubois-Lacoste et al. (2011b).

Instead of using the Pareto dominance or an aggregation-based comparison, the Indicator-Based Multi-Objective Local Search (IBMOLS) (Basseur and Burke, 2007; Basseur et al., 2012) accepts neighbours that are better than any solution of the population, by using a binary multi-objective indicator, such as the hypervolume indicator (Zitzler and Thiele, 1999). The population size of IBMOLS is fixed, the worse solution being replaced as soon as a new neighbour is accepted. The authors also proposed an iterative version of IBMOLS, in which the new initial Pareto set is obtained by applying random noise to a given number of solutions of the Pareto set. If the Pareto set is not big enough, additional solutions randomly generated are considered.

Drugan and Thierens (2012) proposed a multi-restart version of PLS with the Iterated PLS (IPLS). IPLS follows the PLST algorithm, but associates with every solution a Boolean flag that is turned off after the solution neighbourhood is explored. When all solutions are flagged, the search first restarts from a new solution that is randomly generated. After a given number of PLS runs, instead of considering a new solution, IPLS uniformly selects at random a solution from the archive, applies a mutation and restarts from the resulting solution.

Two separate generalisations of the PLS algorithms have since been independently proposed: the Dominance-based Multi-objective Local Search (DMLS) of Liefooghe et al. (2012) and the Stochastic Pareto Local Search (SPLS) of Drugan and Thierens (2012). The DMLS generalisation uses an archive of solutions and includes multiple strategies related to the selection of solutions to explore and to the exploration of the neighbourhood. $\mathrm{DMLS}(\alpha\cdot\beta)$ denotes that the DMLS uses the selection strategy α (with $\alpha \in \{1, \star\}$ for the selection of a single random solution and all solutions, respectively) and the exploration strategy β (with $\beta \in \{1, 1_{\not\prec}, 1_{\succ}, \star\}$ for the acceptance of a single neighbour at random, a single non-dominated neighbour at random, a single dominating neighbour at random and all neighbours, respectively). The SPLS generalisation also uses an archive of solutions from which at each iteration a solution is selected uniformly to be explored. Similarly, multiple exploration strategies are discussed. Furthermore, like in IPLS, a Boolean flag is associated with each solution to avoid exploring it multiple times and to enable faster termination and restarts when the exploration is not performed exhaustively. Indeed, the aforementioned authors also proposed a more generic process to restart PLS algorithms, together with a hybrid genetic PLS algorithm. Moalic et al. (2013) also proposed the Fast Local Search (FLS), which behaves like SPLS

in that the exploration of a solution neighbourhood stops as soon as a neighbour not dominated by the archive is found. Tricoire (2012) also proposed the *multi-directional local search* (MDLS), loosely based on PLS, in which at every iteration a solution from the archive is taken at starting point of a subsidiary local search, before merging the resulting archives by filtering dominating solutions.

The anytime behaviour of PLS algorithms has been investigated by Dubois-Lacoste et al. (2012); Dubois-Lacoste et al. (2015), who proposed variants that optimise not just the quality of the final archive only, but also the quality of intermediate archives. They proposed the optimistic hypervolume improvement (OHVI), an alternative mechanism for selecting the solution of the archive whose neighbourhood will be explored, and, more importantly, they showed that changing the exploration strategy during the search could improve the performance of the PLS algorithm.

Finally, Inja et al. (2014) proposed the Queued Pareto Local Search (QPLS), another restart scheme using a queue to avoid premature convergence. Starting from the initial solutions, QPLS recursively explores every solution of the queue by using dominating neighbours to finally obtain a single final solution. If this final solution is not dominated by the archive, it is merged and k incomparable neighbours are added to the queue. The authors also proposed the Genetic Queued Pareto Local Search (GQPLS), which hybridises genetic algorithm techniques to update the queue.

3.1.3 Condensed Literature Summary

All of the MOLS algorithms outlined above share a common structure, in which a Pareto set of solution is iteratively improved by considering either a solution or a set of solutions as *current*, which is then explored to merge some or all of their neighbouring solutions to the Pareto set. Table 3.1 summarises the main local search algorithms in the literature, according to the five following local search attributes.

Current solutions: A single current solution or a current set of multiple solutions is used by the local search.

Archive: The local search keeps track of a separate current set, or the current solutions can be directly selected from the archive.

Neighbourhood exploration: A single neighbour, the full neighbourhood or only a subset of the neighbourhood (if a stopping criterion is used) is evaluated.

Acceptance criterion: Incomparable and dominated neighbour may be accepted and returned after the neighbourhood exploration, either as the stopping criterion of the exploration or in addition to the final neighbour.

Table 3.1 – Condensed literature summary

"X": the algorithm possesses the given characteristic
"C": the algorithm may be configured to possess the given characteristic

	Current		Archive		Neighbours			Acceptance Criterion		Quality		Reference	
Algorithm	Single current solution	Multiple current solutions	Separate archive and current solution	Current solution(s) from the archive	Single neighbour explored	Partial neighbourhood exploration	Full neighbourhood exploration	Accept if incomparable	Accept if dominated	Aggregation	Pareto dominance	Compare to current solution	Compare to Pareto set
MOSA	X		X		X			X	X	X		X	
PSA		X	X		X			X	X	X		X	
MOTS		X	X				X	X		X		X	X
MOVNS	X			X			X	X			X		X
MOGLS	X			X	X			X		X			X
PAES	X	C	X		C	C		X		X		X	
PLST		X		X			X	X			X		X
PLS	X			X			X	X			X		X
moRBC	X		X	X		X		C			X	X	
IBMOLS		X		X		X		X	X				X
DMLS	C	C		X	C	C	C	C			X	C	C
SPLS	X			X	C	C	C	C			X	C	C
FLS	X			X	X			X			X		X

MOSA (Serafini, 1994; Ulungu et al., 1995); PSA (Czyzak and Jaszkiewicz, 1996); MOTS (Hansen, 1997); MOVNS (Geiger, 2008); MOGLS (Ishibuchi and Murata, 1996); PAES (Knowles and Corne, 1999, 2000a); PLST (Talbi et al., 2001); PLS (Paquete et al., 2004; Angel et al., 2004); moRBC (Aguirre and Tanaka, 2005); IBMOLS (Basseur and Burke, 2007); DMLS (Liefooghe et al., 2012); SPLS (Drugan and Thierens, 2012); FLS (Moalic et al., 2013)

Quality: The comparison of the quality of two neighbours is done by considering either an aggregation or the Pareto dominance.

Reference: During the neighbourhood exploration, neighbours are compared either to the current solution or to other solutions such as the full Pareto set.

In Table 3.1, an "X" means that the algorithm possesses the corresponding characteristic, possibly depending of the context during the resolution (e.g., SA algorithm accepting dominated solutions by means of the temperature), whereas a "C" means that the characteristic is only present in some particular variant of the algorithm (e.g., the DMLS structure is able to instantiate many different local search algorithms).

3.1.4 Analysis and Discussion

Table 3.1 shows a trend between the two algorithmic families of the MOLS algorithms, where extensions of single-objective local search algorithms generally separate the archive and the current solutions and use aggregations, and the family of the PLS algorithms, which generally directly select the current solutions from the archive and use Pareto dominance.

One of the apparent weakness of MOLS algorithms relates to the possible number of solutions included in the archive and thus the size and shape of the optimal Pareto front. Indeed, if a MOLS algorithm does not use any mechanism to bound the size of its archive, exploration of too many solutions (and furthermore exhaustive neighbourhood explorations) can become prohibitively computationally expensive slowing the convergence of the algorithm to a halting point (Liefooghe et al., 2012). MOLS are similarly much weakened when using too large neighbourhood, especially when explorations are performed exhaustively. Another current weakness of MOLS algorithms is that there is usually no explicit handling of the intensification/diversification trade-off. If some works focus on preserving diversity at the cost of some convergence speed (Blot et al., 2015), in most of the MOLS algorithms only intensification is rewarded and diversification is delegated as a side-effect of the archiving process. Furthermore, some variants of MOLS algorithms may require long computational time to reach high-quality approximations of the Pareto fronts and result on poor solutions if stopped early. Anytime mechanisms for MOLS algorithms have been proposed to deal with this particular limitation (Dubois-Lacoste et al., 2015).

The two DMLS and SPLS generalisations can be configured to instantiate a large range of PLS strategies, but are not compatible with many extensions of single-objective strategies (and do not claim to be). The first fundamental limitation is that these generalisations do not use an explicit set of current solutions

that is conveyed through the iterations of the local search, but instead select new current solutions from the archive every iteration. This also implies that the current solutions are always non-dominated. They can, through the use of an activation/deactivation scheme, emulate to some extent some trajectory-based strategies by keeping track of the selection of the previous iteration, but without the flexibility of keeping a separate set of current solutions, which allows, for example, to easily perform explorations outside their current archive (e.g., to explore dominated neighbours or when the algorithm allows some deterioration of the current solutions). The second main limitation is that the use of an archive as the main set of solutions leads to the use of the Pareto dominance (or a weakened version) for quality comparison, which leaves out the use of scalar-based comparisons in the exploration procedure.

To overcome these limitations, to allow more flexibility and to incorporate more diverse strategies, we propose a new MOLS generalisation, which is detailed in the following sections. Its main characteristics are the use of two explicit sets of solutions (namely, the set of current solutions and the archive), the separation of acceptance and stopping criteria in the exploration strategy, the possibility of using a simple set and not a Pareto set for the set of current solutions, the possibility of using scalar-based acceptance criteria and, finally, the use of an explicit reference during neighbourhood comparisons.

3.2 MOLS Strategies

In this section, we describe different sets and strategies of the MOLS algorithms through examples from the literature review of the previous section. They are the basic components of our unification of MOLS that will be presented in Section 3.4.

3.2.1 Set of Potential Pareto Optimal Solutions (Archive)

The *archive* is the Pareto set at the core of all MOLS algorithms. It holds potential Pareto optimal solutions, i.e., solutions not yet dominated by any other found solutions. This is the set of solutions finally returned by the procedure.

Depending on the problem considered, the size of the archive can become very large. Unless this size is kept unbounded, a mechanism such as a diversity criterion (e.g., crowding, relaxed dominance) or a basic filtering mechanism may be used to remove the less important potential Pareto optimal solutions once a given size is reached (Liefooghe et al., 2012).

3.2.2 Set of Current Solutions (Memory)

In addition to the archive, the *current set*, a second set of solutions, is used to keep all the solutions whose neighbourhood may be explored. These solutions are taken either from the archive or from previous iterations and may possibly be dominated by some solutions of the archive. To avoid using the same term (i.e., *current*) for both the current set and the current solutions it contains, we propose to call this set *memory*.

We identify three categories of strategies concerning the usage of the memory. First, as a direct extension of the single-objective local search algorithms, the memory can contain a single current solution (e.g., MOSA algorithm (Ulungu et al., 1999)). Iteration after iteration, the current solution is explored, potentially replaced by one of its neighbours, while the archive is automatically updated. If the current solution appears to be a PLO, a restart can then be performed from one of the other potential Pareto optimal solutions. However, considering a single current solution means focusing on a single trajectory in the search space, whereas the multi-objective setting requires optimising the whole Pareto front. Thus, the second category of strategies includes algorithms that keep a set of multiple *current* solutions and explores it sequentially, with the direct consequence of an improved diversity since each of the separate trajectories can then focus on the subset of the Pareto front (e.g., PSA (Czyzak and Jaszkiewicz, 1998), MOTS algorithms (Hansen, 1997)). Finally, the third category includes algorithms that do not keep track of the trajectory, but rather directly select and explore solutions from the archive (e.g., PAES (Knowles and Corne, 1999), PLS (Paquete et al., 2004), DMLS algorithms (Liefooghe et al., 2012)).

Note that, like in the archive, the size of the memory may become very large, and, therefore, the same bounding mechanisms may be used. However, as such mechanisms were proposed for algorithms in which the memory and the archive were joined, it may be advantageous to bound only the memory and keep the archive unbounded.

We may envision a new exploration strategy where multiple solutions could be explored at the same time by combining their neighbourhoods. In that case, without loss of generality, the *current* object would be itself a set of solutions and the *memory* would be a set of sets of solutions.

3.2.3 Exploration Strategies

The exploration of the current solution consists in the construction of its neighbourhood, i.e., the generation of its neighbours.

Like in the single-objective case, two types of exploration strategy are distinguished: the *best improvement strategy* and the *first improvement strategy*. The *best* strategies compare every neighbour to the current solution or to the reference so that only the best non-dominated neighbours are accepted. On the contrary, the *first* strategies generate neighbours one by one and stop when a given stopping criterion is reached. Of course, the latter strategies are not limited to stopping after a single accepted neighbour. In both the *best* and the *first* strategies, the exploration procedure generates some neighbours, accepting some of them, and then returns the set of *accepted* neighbours. For each of these neighbours, three questions arise: (i) Should it be included into the archive? (ii) Should it replace the current solution? (iii) Should the exploration continue or stop in regard to its quality?

The quality of a neighbour can be a function of either the current solution or a part or the totality of the archive. Figure 3.1 shows how the objective space is divided into dominating solutions (a), incomparable solutions (b) and dominated (c) solutions, regarding (left) a single solution x and (right) multiple solutions x, u, v and w. Solutions in the (c) space are dominated by the current solutions and are generally ignored, whereas exploration strategies usually consider solutions in the (a) or (a+b) spaces. Considering the neighbouring solutions enables to make better-informed decisions, e.g., distinguishing between the (α), (β), and (γ) spaces; the main drawback, however, is the added cost (e.g., computational time) of an overall more expensive exploration procedure. An alternative to using the Pareto dominance criterion is to aggregate the objectives, to obtain a scalar value subsequently used to either rank neighbours or compute probabilities. The weights of the aggregation can be either globally set, associated with the current solution or updated automatically in regard to the state of the archive.

The archive (the set of potential Pareto optimal solutions) can be updated directly either during the exploration of a current solution or after the exploration of all current solutions has been performed. In the direct update, the explorations of the remaining current solutions may be impacted, i.e., the reference set is modified on the fly. Similarly, the memory (the set of current solutions) can be updated during the exploration to replace the current explored solutions (e.g., in trajectory-based local search algorithms (Serafini, 1994; Ulungu et al., 1995; Czyzak and Jaszkiewicz, 1996; Hansen, 1997)) or to include promising new neighbours directly (Blot et al., 2015). If the memory contains multiple solutions, they are all explored before the search continues unless an early stopping criterion is met. Note that, if multiple solutions are explored and either the memory or the archive is updated during the exploration, the order in which the solutions of the memory are explored can strongly impact the performance.

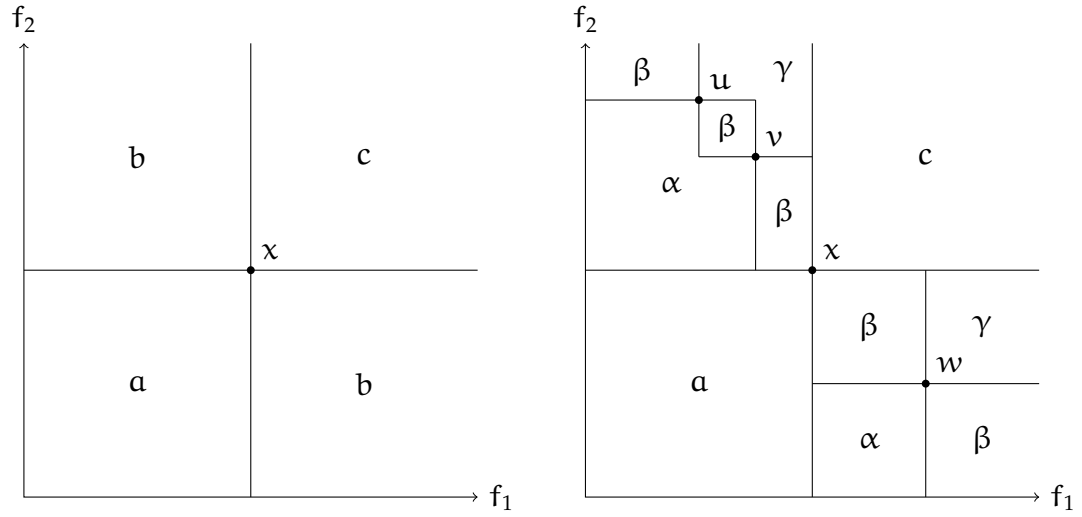


Figure 3.1 – Objective space around x, without (left) and with (right) taking into account surrounding solutions u, v and w

(a), (b) and (c): objective space partitions in which the solution respectively dominates, is incomparable with and is dominated by the solution x.
(α), (β) and (γ): subdivisions of the (b) objective space partition in which the solution respectively dominates, is incomparable with and is dominated by the solution u, v or w.

3.2.4 Selection Strategies

After the exploration step has been completed, the solutions of the memory will have been explored and the archive will have been updated with the accepted neighbours. The memory has to be updated for the next iteration. Generally, the solutions are taken from the archive (e.g., randomly, with regard to a crowding or sharing property (Deb, 2001), to an individual contribution (Dubois-Lacoste et al., 2012) or to the order of insertion in the archive (Blot et al., 2017a)). However, in trajectory-based local search algorithms, the memory is unchanged since it has been updated during the previous exploration step.

3.2.5 Termination Criteria

The local search has a natural termination criterion, which is reached when the memory becomes empty, meaning that no more solution is to be explored. Such an event generally means that every solution of the archive is a PLO. This situation also arises when the algorithm intentionally removes partially explored solutions from the memory, for example, to force a quick convergence or ensure diversification. Other commonly used termination criteria include the whole computational time; the total number of iterations, explorations or evaluations; and the number of successive iterations without improvement.

3.3 Escaping Local Optima

In single-objective optimisation, local search algorithms are generally trapped in local optima. However, various mechanisms (e.g., SA, TS) can be used to converge further towards a global optimum. Likewise, the basic instantiations of the procedures detailed in this paper will generally be trapped in sets of PLO. Likewise, the same various mechanisms can be and have been adapted for MOLS procedures to converge further towards the set of Pareto optima.

First, a temperature can be used to compute the probabilities of accepting neighbours of lesser quality (Serafini, 1994; Ulungu et al., 1995; Czyzak and Jaszkiewicz, 1996). This temperature can be either a global parameter of the local search or a specific temperature that can be associated with each and every solution of the memory when the local search follows a set of solutions of fixed size. The Tabu paradigm can also be used to drive the search out of the PLO (Hansen, 1997). Similarly, a global tabu list or a set of tabu list can be used for each followed solution. Finally, it is possible to use an iterated local search scheme (Drugan and Thierens, 2012) to stop the local search early, before reaching a true set of PLO. In this

Procedure 3.1: LS(`memory`, `archive`)

Input: `memory`, a set of solutions to generate neighbourhoods
Input: `archive`, a Pareto set of solutions
Output: the updated `archive` set

```
until local search stopping condition is met
  or memory = ∅ do
    candidates ← ∅;
    until iteration stopping condition is met
      or every current ∈ memory has been considered do
        let current ∈ memory;
        ref ← REFERENCE( current, memory, archive,
         candidates );
        accepted ← EXPLORE( current, ref, archive );
        memory ← UPDATE( memory, current, accepted );
        candidates ← candidates ∪ accepted;
    archive ← COMBINE( archive, candidates );
    memory ← SELECT( memory, archive, candidates );
return archive;
```

case, a convergence condition is defined as, for example, a threshold in the convergence rate or a stagnation criterion. The search can then restart either from the new solutions selected uniformly in the search space or from the solutions in the close neighbourhood of the current or the best solutions, using a *kick*. In the single-objective case, a kick consists in taking a solution, either the current one or the best one, and performing a given number of random moves over the search space. In the multi-objective case, some solutions are selected (either a single one, a fixed number or a ratio of solutions, or all of them) from the memory or the archive; a single-objective kick is performed on each of them, and the resulting solutions are included in a new Pareto set, and then the algorithm restarts from it.

3.4 MOLS Unification Proposition

3.4.1 Main Loop

Procedure 3.1 (LS) describes the main loop of the local search. This procedure takes an initial current set and an archive as input and returns the updated archive. It consists in iterating three steps (the names in parentheses are the names of the

Procedure 3.2: EXPLORE(`current`, `ref`, `archive`)

Input: `current`, a solution to generate the neighbourhood
Input: `ref`, a set of solutions to compare neighbours with
Input: `archive`, a Pareto set of solutions
Output: `accepted`, the set of accepted solutions
Side effect: modifies the `archive` set

```
accepted ← ∅;
```
until *exploration stopping condition is met*
  or *every* `neighbour` $\in \mathcal{N}$(`current`) *has been considered* **do**
  let `neighbour` $\in \mathcal{N}$(`current`);
  `accepted` ← ACCEPT(`accepted`, `neighbour`, `ref`);
  `current`, `ref`, `archive` ← UPDATE(`ref`, `accepted`, `current`, `archive`, `neighbour`);

return `accepted`;

sub-procedures described as they appear in Procedure 3.1).

1. First, the solutions of the memory are explored one by one: for each, a reference is chosen to compare the neighbours with (REFERENCE), then some or all of the neighbours are accepted as candidates (EXPLORE), and, finally, the memory may be updated with the neighbours (UPDATE).
2. When all the current solutions have been explored, or when an early stopping condition is met, all accepted neighbours are used to update the archive (COMBINE). Note that it is possible to update the archive during the exploration, in which case the COMBINE procedure can still be used to bound its size.
3. Finally, the memory is set up with the new solutions to explore.

These three steps are iterated until the memory is empty or as soon as a given stopping condition is met.

3.4.2 Local Search Exploration

The exploration mechanism (EXPLORE) is described in Procedure 3.2. This procedure handles how neighbouring solutions are generated and accepted, and how the reference set is updated. It takes as input a solution to explore, which is used to generate the neighbourhood; a reference set to compare the neighbours with; and the archive of the local search. It returns a set of accepted neighbours of the input solution and possibly modifies the archive as a side effect.

Procedure 3.3: ITER(`archive`)

Input: `archive`, a Pareto set of solutions
Output: the updated `archive*` set

```
archive ← LS(archive);
archive* ← archive;
until global stopping condition is met do
    memory, archive ← PERTURB(archive, archive*);
    archive ← LS(memory, archive);
    archive* ← COMBINE(archive, archive*);
return archive*;
```

The neighbours of the current solution are generated one by one, and for each new neighbour, the set of accepted neighbours is updated (ACCEPT). To implement some local search algorithms from the literature, it is possible to immediately update the current solution, the reference set and the archive (UPDATE). Neighbours are generated until every possible neighbour of the current solution has been generated or as soon as a given stopping condition is met.

3.4.3 Iterated Local Search Algorithm

The local search of Procedure 3.1 (LS) can eventually stop because either the archive contains only PLO or an early stopping condition has been met. One of the possible mechanisms to iterate the local search (LS) and continue the search is described in Procedure 3.3 (ITER). It follows the iterated local search (ILS) scheme (Lourenço et al., 2003; Drugan and Thierens, 2010, 2012) where the final archive given by the local search is slightly modified and given again as input to the local search procedure.

First, the local search is performed once, which sets up `archive*`, the Pareto set that contains the overall best non-dominated solutions across local search iterations. Then, until the global stopping condition is met, new initial memory and archive are generated (PERTURB), subsequent local search are performed and the two archives are combined to update `archive*` (COMBINE).

3.5 Literature Instantiation

Following the unification presented in Section 3.4, Table 3.2 and Table 3.3 detail how the main literature algorithms are instantiated in Procedure 3.1 and Proced-

ure 3.2, respectively, of our structure. In Table 3.2, k designates a constant of the algorithm set beforehand. In Table 3.3, the "$*$" symbol means that the memory size is variable.

Table 3.2 shows that many of the MOLS algorithms in the literature use the current solution as a reference. However, recent studies increasingly encourage the use of the archive as a reference since it leads to improved results (Blot et al., 2017a,c). The recombination column highlights that the recombination only makes sense when the exploration step returns a new Pareto archive; for trajectory-based local search algorithms, such a step is directly performed during the exploration, when a neighbour replaces the current solution in the memory. Not mentioned here is the possible bounding of the archive size, which is also performed on some problems after Pareto filtering (e.g., Liefooghe et al., 2012). The selection column mainly differentiates between trajectory-based algorithms, for which such a step is likewise irrelevant, and algorithms that do not use a memory mechanism but recreate the set of new solutions every iteration.

Lastly, Table 3.3 shows that, if the first MOLS algorithms predominantly accepted improving neighbours, newer MOLS algorithms have shown that considering incomparable neighbours leads to improved results.

Table 3.2 – Condensed literature instantiation (LS Procedure)

imp.: improving (implies underlying scalarisation)
dom.: dominating
ndom.: non-dominated (either dominating or incomparable)

Algorithm	REFERENCE	UPDATE	COMBINE	SELECT	Miscellaneous
MOSA	current solution	replace if *imp.*	irrelevant	do nothing	
PSA	current solution	replace if *imp.*	irrelevant	do nothing	
MOTS	irrelevant	always replace	irrelevant	drift if long enough	
MOVNS	irrelevant	remove current if fully explored; add *ndom.* neighbours; filter	Pareto dominance	all solutions	
MOGLS	current solution	replace if *imp.*	Pareto dominance	do nothing	stop after a given number of explorations without improvement
$(1+1)$-PAES	current solution	replace if *dom.* or less crowded	Pareto dominance	do nothing	
$(1+\lambda)$-PAES	current solution	replace if *dom.* or less crowded	Pareto dominance	do nothing	
$(\mu+\lambda)$-PAES	current solution	replace if *dom.* or less crowded	Pareto dominance	do nothing	reference chosen via binary tournament
PLST	archive	do nothing	Pareto dominance	all solutions	
PLS	irrelevant	replace if *dom.*	Pareto dominance	single unexplored	
moRBC	current solution	replace if *imp.*	irrelevant	irrelevant	restart on local optima
IBMOLS	memory	do nothing	irrelevant	all solutions	
DMLS $(1\cdot)$	current solution	do nothing	Pareto dominance	single unexplored	
DMLS $(\star\cdot)$	current solution	do nothing	Pareto dominance	all solutions	
SPLS	current solution	do nothing	Pareto dominance	single non-flagged	
FLS	current solution	always replace; filter	Pareto dominance	do nothing	

MOSA (Serafini, 1994; Ulungu et al., 1995); PSA (Czyzak and Jaszkiewicz, 1996); MOTS (Hansen, 1997); MOVNS (Geiger, 2008); MOGLS (Ishibuchi and Murata, 1996); PAES (Knowles and Corne, 1999, 2000a); PLST (Talbi et al., 2001); PLS (Paquete et al., 2004; Angel et al., 2004); moRBC (Aguirre and Tanaka, 2005); IBMOLS (Basseur and Burke, 2007); DMLS (Liefooghe et al., 2012); SPLS (Drugan and Thierens, 2012); FLS (Moalic et al., 2013)

Table 3.3 – Condensed literature instantiation (EXPLORE Procedure)

imp.: improving (implies underlying scalarisation)
dom.: dominating
ndom.: non-dominated (either dominating or incomparable)

Memory size: 1: single solution; k: constant; $*$: variable; other: other constant

Algorithm	Memory size	ACCEPT	UPDATE	Miscellaneous
MOSA	1	if *imp.*	update both	
PSA	$*$	if *imp.*	update both	
MOTS	$*$	best non-tabu neighbour	update both	
MOVNS	1	if *ndom.*	irrelevant	use an unexplored neighbourhood
MOGLS	$*$	first *imp.*	do nothing	stop after first *imp.* neighbour or k neighbours
$(1+1)$-PAES	1	if *dom.* or less crowded	do nothing	stop after 1 neighbour
$(1+\lambda)$-PAES	1	if *dom.* or less crowded	do nothing	stop after λ neighbours
$(\mu+\lambda)$-PAES	μ	if *dom.* or less crowded	do nothing	stop after λ neighbours
PLST	$*$	if *ndom.*	do nothing	
PLS	1	if *ndom.*	do nothing	
moRBC$(1+1)$	1	if *dom.*	replace `ref` if accepted	neighbours generated using the current reference
moRBC$(1+1)^*$	1	if *ndom.*	replace `ref` if accepted	neighbours generated using the current reference
moRBC$(1+1)^A$	1	if *dom.* or less crowded *ndom.*	replace `ref` if accepted	neighbours generated using the current reference
IBMOLS	$*$	if not worst w.r.t. the indicator	remove worst from `ref`	
DMLS$(\cdot 1)$	k	if *dom.* or *ndom.*	do nothing	stop after 1 neighbour
DMLS$(\cdot 1_{\not\prec})$	k	if *dom.* or *ndom.*	stop after 1 *ndom.*	
DMLS$(\cdot 1_{\succ})$	k	if *dom.* or *ndom.*	do nothing	stop after 1 *dom.*
DMLS$(\cdot\star)$	k	if *dom.* or *ndom.*	do nothing	
SPLS	1	if *dom.* or *ndom.*	do nothing	
FLS	$*$	if *ndom.*	do nothing	stop after first *ndom.* neighbour

MOSA (Serafini, 1994; Ulungu et al., 1995); PSA (Czyzak and Jaszkiewicz, 1996); MOTS (Hansen, 1997); MOVNS (Geiger, 2008); MOGLS (Ishibuchi and Murata, 1996); PAES (Knowles and Corne, 1999, 2000a); PLST (Talbi et al., 2001); PLS (Paquete et al., 2004; Angel et al., 2004); moRBC (Aguirre and Tanaka, 2005); IBMOLS (Basseur and Burke, 2007); DMLS (Liefooghe et al., 2012); SPLS (Drugan and Thierens, 2012); FLS (Moalic et al., 2013)

Chapter 4

MOLS Instantiations

> A computer program can modify itself but it cannot violate its own instructions – it can at best change some parts of itself by *obeying* its own instructions.
>
> *Gödel, Escher, Bach: An Eternal Golden Braid*
> Douglas Hofstadter

In this chapter, following the discussions of Chapter 3, we discuss the specific multi-objective local search (MOLS) algorithms that will be used for experiments in Chapter 6, Chapter 7, and Chapter 8, together with their implementation.

First, we describe a highly configurable MOLS algorithm, resulting from the unification proposition of Chapter 3. We discuss its possible parameters and configuration space. This MOLS algorithm is thoroughly analysed in Chapter 6, and serves as basis for all MOLS algorithms used in this thesis.

This first MOLS algorithm is *static*: once set, its parameters are used during the entire execution and cannot be changed. In order to study MOLS algorithms in which the value of the parameters are adapted during its execution, we then discuss the steps necessary to obtain an *adaptive* MOLS algorithm. We also examine the most impactful parameter of the static MOLS algorithm, and discuss how generic control mechanisms can be integrated. The resulting *adaptive* MOLS algorithm is analysed in Chapter 7.

Then, we discuss the possibility of considering schedules of MOLS configurations, as an intermediary proposition between *offline* and *online* design of MOLS algorithms. Parameters values and strategies can change during the execution of the MOLS algorithm, but only according to a *static* schedule. These schedules of MOLS configurations are investigated in Chapter 8.

Finally, we present AMH (Adaptive MetaHeuristics), the C++ framework in which all the instantiations presented in this chapter have been implemented. It was designed according to the experimental needs of the following chapters, in order to facilitate the automatic construction, control, and integration into automatic design tools.

While the publications regarding the different MOLS instantiations are described in the chapters in which they are respectively studied, the AMH framework is specifically linked to the following publication:

- **Blot, A.**, Kessaci-Marmion, M., and Jourdan, L. (2017b). **AMH: a new framework to design adaptive metaheuristics.** In *12th Metaheuristics International Conference, MIC 2017. Proceedings*, pages 586–588.

4.1 Static MOLS Algorithm

Chapter 3 proposed a unification of MOLS strategies into a general framework. In this section, we describe the specific instantiation that will be used in Chapter 6 together with the parameters that comprise its configuration space.

4.1.1 Algorithm

Algorithm 4.1 describes a basic static MOLS algorithm. It is based on an iterated local search (see Procedure 3.3) and the DMLS algorithm of Liefooghe et al. (2012). The inner MOLS loop iterates three successive steps: the selection step in which some solutions of the current set are selected, the exploration step in which the neighbourhood of every selected solution is investigated, and the archive step in which the resulting neighbours are merged into the current set of solutions. The outer MOLS loop simply performs a perturbation of the current archive, calls the inner MOLS loop on the perturbed set of solution, merges the resulting set with the current archive, and iterates until the global termination criterion is met.

This instantiation differs from the unification of Chapter 3 by the set of strategies we choose to focus on. In particular, it only features dominance-based strategies, while strategies based on aggregation are not taken into account. Other differences include a selection step *before* the exploration step, at the beginning of the main loop, rather than at its end, a complete exploration of every solution selected, and an explicit combination mechanisms that first remove dominated solutions before bounding the size of the archive. Finally, as in the DMLS algorithm, the "mem-

Algorithm 4.1: Static Iterated Multi-Objective Local Search

Input: `archive`, a Pareto set of solutions
Output: the updated `archive` set

```
current ← archive;
/* Inner MOLS loop */
until inner termination criterion is met do
    /* Selection */
    memory ← SELECT(current);
    /* Exploration */
    candidates ← ∅;
    for solution ∈ memory do
        ref ← REFERENCE(solution, current);
        accepted ← EXPLORE(solution, ref);
        candidates ← candidates ∪ accepted;
    /* Archive */
    current ← bound(pareto(current ∪ candidates));
archive ← pareto(archive ∪ current);
/* Outer MOLS loop (ILS) */
until termination criterion is met do
    /* Perturbation */
    current ← PERTURB(archive);
    /* Inner MOLS loop, again */
    until inner termination criterion is met do
        /* Selection */
        memory ← SELECT(current);
        /* Exploration */
        candidates ← ∅;
        for solution ∈ memory do
            ref ← REFERENCE(solution, current);
            accepted ← EXPLORE(solution, ref);
            candidates ← candidates ∪ accepted;
        /* Archive */
        current ← bound(pareto(current ∪ candidates));
    archive ← pareto(archive ∪ current);
return archive;
```

ory" is not updated during the exploration, as it is discarded at the end of every iteration.

Finally, Figure 4.1 and Figure 4.2 illustrate the *inner* and *outer* loops of the MOLS algorithm and the chronological succession of the MOLS and the ILS iterations.

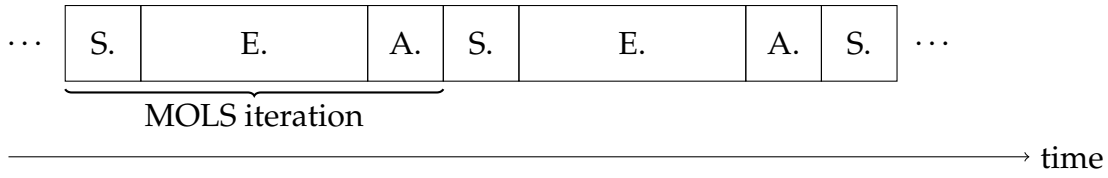


Figure 4.1 – Inner MOLS loop (labels: "S.": selection, "E.": exploration, "A.": archive)

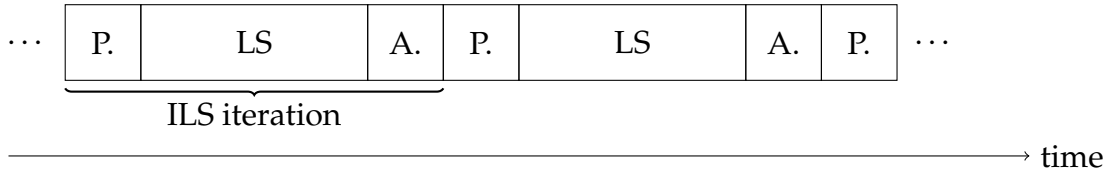


Figure 4.2 – Outer MOLS loop (labels: "P.": perturbation, "LS": local search, "A.": archive)

4.1.2 Configuration Space

The complete configuration space of Algorithm 4.1 considered in this thesis is summarised in Table 4.1. It features five main categorical parameters, that are tied to the different choices of strategies of the MOLS algorithm, and five integer parameters, that each enable fine tuning of one strategy.

We shortly describe hereafter the meaning of every parameter and every parameter value. They are discussed more deeply in Chapter 3.

`select-strat`: the selection strategy: with the value `all`, every solution of the archive will be explored; otherwise, with either of the values `rand`, `newest`, or `oldest`, only some solutions, respectively uniformly chosen at random from the archive, or chosen within the latest or oldest solutions included in the archive will be explored.

`select-size`: the (strictly positive) number of solution selected from the archive.

Table 4.1 – Considered parameter space

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`select-size`	$\mathbb{N}^+$
Exploration	`explor-strat`	{`all`, `all-imp`, `imp`, `imp-ndom`, `ndom`}
Exploration	`explor-ref`	{`sol`, `arch`}
Exploration	`explor-size`	$\mathbb{N}^+$
Archive	`bound-strat`	{`unbounded`, `rand`, `replace`}
Archive	`bound-size`	$\mathbb{N}^+$
Perturbation	`perturb-strat`	{`kick`, `kick-all`, `restart`}
Perturbation	`perturb-size`	$\mathbb{N}^+$
Perturbation	`perturb-strength`	$\mathbb{N}^+$

`explor-strat`: the exploration strategy: with the values `all` or `all-imp`, every neighbour is evaluated and the non-dominated and dominating neighbours, respectively, are returned; with the values `imp`, `imp-ndom`, or `ndom`, the neighbours are iteratively evaluated until a sufficient number of dominating, dominating and non-dominated neighbours, respectively, are found and returned. Finally, with the value `imp-ndom`, the non-dominated neighbours does not contribute to the *number* of neighbours selected but are also returned.

`explor-ref`: the reference of the exploration, either the current explored solution (with the value `sol`), or the current archive (with the value `arch`).

`explor-size`: the number of neighbours selected in the `imp`, `imp-ndom`, and `ndom` exploration strategies.

`bound-strat`: the bounding strategy after Pareto dominance. With the value `unbounded`, it returns the current archive. With the value `rand`, solutions chosen uniformly at random are discarded from the archive as long as the size of the archive is too large; with the value `replace`, it uses the DMLS strategy of removing newly accepted solutions if they did not replaced at least one solution of the archive.

`bound-size`: the maximum number of solution in the archive above which the bounding strategy will apply.

`perturb-strat`: the perturbation strategy: with the value `kick`, some solutions of the archive will be selected uniformly at random and then iteratively replaced by one of their neighbours; with the value `kick-all`, the `kick` strategy applies for every solution of the archive; with the value `restart`, some solution of the search space, selected uniformly at random, will be con-

sidered instead.

`perturb-size`: the number of solutions considered in the `kick` and `restart` strategies.

`perturb-strength`: the number of kick iterations in the `kick` and `kick-all` strategies.

4.2 Control Mechanisms Integration

The philosophy of integrating online mechanisms in static algorithms is to make the most of multiple strategies, mechanisms, or parameter values, during a single execution of the algorithm, whereas the static algorithm would only have used a single one.

In this section, we discuss how to integrate generic control mechanisms into the static MOLS algorithm introduced in the previous section. This discussion is divided into five successive steps, each answering a different question of the integration process:

Parameter analysis: *what alternative* strategies can be used?
Knowledge exploitation: *how to use* knowledge to select the strategy to use?
Knowledge extraction: *what information* from the search process can be used?
Knowledge modelling: *how to store* extracted knowledge?
Decisional schedule: *when* and *how frequently* extract and use knowledge?

Similar discussions are for example described in Karafotias et al. (2012) as first considering the parameters ("*what* is to be controlled"), then a set of observables ("what *evidence* is used"), and the algorithm making decisions ("*how* the control is performed"); or in di Tollo et al. (2015) as the following steps: *aggregation criteria computation*, *reward computation*, *credit assignment*, and *operator selection*.

4.2.1 Parameter Analysis

The parameters of our static MOLS algorithm, described in Table 4.1, are either categorical and numerical parameters. For each phase of the MOLS algorithm, the categorical parameters defines the strategy that will be applied, while the numerical parameters are used to further fine-tune the behaviour of the algorithm and are common to all strategies.

In order to use a control mechanism, a choice of which parameter will be controlled is necessary. Several possibilities naturally arise. First, the control mechanism can be limited to a single parameter of the MOLS algorithm, either a categorical or a numerical one, while all the other parameters values are fixed during

the run. The control mechanisms could also deal with multiple parameters, either independently or through the use of combinations of parameter values. Finally, multiple control mechanisms could theoretically be used independently on multiple parameters. To keep the our adaptive MOLS algorithm simple and to ensure its viability we will use standard literature control mechanisms and focus on controlling a single parameter.

The most impactful phase of the static MOLS algorithm is without doubt the exploration phase, that exposes up to nine possible exploration strategies (with the combination of the `explor-strat` and `explor-ref` parameters). We choose to focus on the strategy categorical parameter, rather than on the fine-tuning numerical parameter, as we expect its control to have a more significant impact of the performance of the MOLS algorithm.

4.2.2 Knowledge Exploitation

From the choice of the selected parameters to be controlled follows the choice of the control mechanism itself. While a large number of possible control mechanism can be found in the literature (see Chapter 2), the majority of them are fundamentally tied to a given algorithm or a given parameter, and cannot simply be extended to other algorithms.

However, as we focus on a single, categorical, parameter, a class of generic control parameters including probability based mechanisms, multi-armed bandits, and also reinforcement learning can still conveniently be applied. An overview of these generic control mechanisms will be given in the following section.

4.2.3 Knowledge Extraction

The two main types of algorithm in which parameter control is used are deterministic and adaptive algorithms, that differ in their use of feedback from the search process (see Chapter 2). While control mechanisms in deterministic algorithm only use predictable feedback such as elapsed time or number of iterations, control mechanisms of adaptive algorithms requires feedback that is based on the search process itself, the current instance and the solutions evaluated so far. This feedback is essential to fuel the decisions that enable the control mechanism to select efficient strategies during the search. Hence follows the following crucial question: *what information* from the search process can be used as feedback?

Knowledge that can be assessed or computed during the execution of the algorithm is called an *observable*. A possible set of simple observable would for ex-

ample for evolutionary algorithms be genotypic diversity, phenotopic diversity, fitness standard deviation, fitness improvement, or also stagnation counters (Karafotias et al., 2014).

For our adaptive MOLS algorithm, we choose to use the fitness improvement as observable, computed through the mean of the hypervolume (see Chapter 1). While hypervolume is a by itself a very good and widespread multi-objective accuracy performance indicator, and incidentally one of the two quality indicators that will be used in the following chapters, it was shown to be a efficient measure to feed control mechanisms in multi-objective optimisation (Moffaert et al., 2013).

To motivate our choice of hypervolume as feedback, preliminary observations on other possible observables have been conducted, namely based on direct objective values, number, distribution of solutions in the archive, and distance between successive and extreme solutions. None of these investigated observables was deemed a reasonable alternative to the use of hypervolume as feedback to the control mechanism.

4.2.4 Knowledge Modelling

Having chosen the hypervolume as feedback for the control mechanism, the second question to answer is *how to store* the extracted knowledge.

As a result of the well-separated and categorical nature of the MOLS exploration strategy parameter, that we choose to focus on, and the ensuing simple control mechanisms, we decided to model each possible exploration strategy by an individual arm, and to associate to every arm a single value associated to the predicted feedback of the arm.

Note that the use of reinforcement learning control mechanisms would require to also model the possible states of the search and then to associate a single prediction value to every combination of an arm and a possible state. Furthermore, some other control mechanisms not considered in the experiments could possibly require to have access to a more detailed version of the whole history of feedback, rather than a single aggregated value, e.g., to use a sliding window of the history (Maturana et al., 2009).

4.2.5 Decisional Schedule

Most importantly, after the questions of *what strategies* can be used, *how to select* them dynamically, *what knowledge* can be used to that end, and finally *how to store* said knowledge, comes the final, central, question of *when to take decisions*.

Of course, extracting knowledge, store it, and use it to drive a control mechanism has an inherent cost that slows down the algorithm, compared to the static version that always use the same statically determined configuration and set of strategies. However, this cost is necessary to accurately use the best strategies on every instance rather than only relying the a static offline prediction made for all instances. It is nevertheless important to not impair computation time too much in favour of the additional learning overhead. Additionally, while some strategies may require some time before reaching their peak efficiency, the knowledge sources may also require sufficient time between extractions for the feedback to be relevant and useful.

In our MOLS algorithm (Algorithm 4.1), there are at least two meaningful places where a control mechanism could be integrated. First, it could be placed inside the inner loop of the MOLS, meaning that it would take the decision of which exploration mechanism to use before its actual application at the beginning of the MOLS iteration, and could gather the feedback of its performance at the end of the iteration. This possibility is illustrated in Figure 4.3.

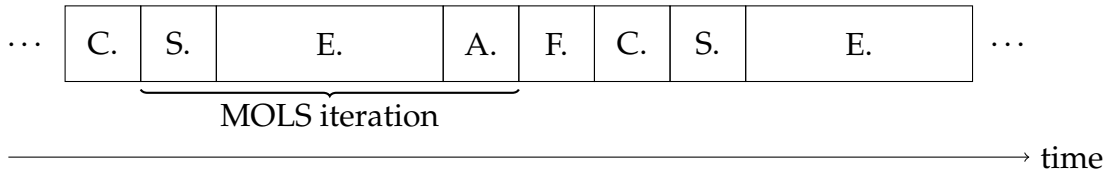


Figure 4.3 – Control integration in the inner MOLS loop (labels: "C.": control, "S.": selection, "E.": exploration, "A.": archive, "F.": feedback)

This would probably be the most natural approach, if not for the following issues. First, not all explorations use the same computation time, as exhaustive explorations are naturally much more time-consuming that partial explorations. This could however be alleviated by scaling the following feedback value using the elapsed computation time. On the other hand, a second problem is that if a single exploration is ineffective, the feedback associated to that exploration strategy will instantaneously be affected negatively. To easily allow more time for the exploration strategy to be effective, the selection and feedback steps of Figure 4.3 could be performed once every k MOLS iterations, where k is an additional parameter that should be determined experimentally. However, the third problem, which is that our MOLS algorithm is based on an iterated local search algorithm, and as such occasionally performs perturbations, after which the previous feedback may not relevant to chose the exploration strategy.

In view of these problems, a second possibility was considered instead, illustrated in Figure 4.4. This time, the control mechanism is placed inside the *outer* loop of the MOLS instead, i.e., in the ILS loop. Decisions about which exploration to use are then taken before the call the the inner MOLS procedure, that then use the same exploration until it finishes, and the feedback is computed after the archive resulting of the MOLS procedure is merged into the final archive.

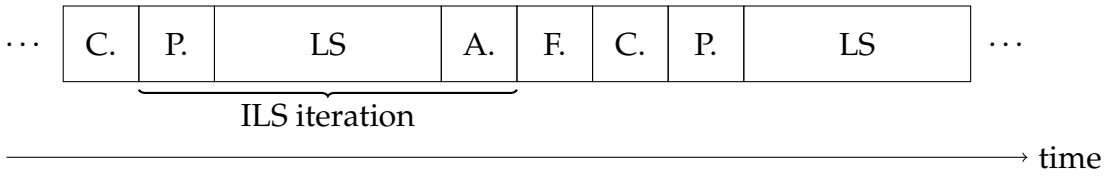


Figure 4.4 – Control integration in the outer MOLS loop (labels: "C.": control, "P.": perturbation, "LS": local search, "A.": archive, "F.": feedback)

4.3 Adaptive MOLS Algorithm

Having discussed in the previous section the integration into Algorithm 4.1 of control mechanisms, we present in this section the adaptive MOLS algorithm that will be used in Chapter 7 along with the generic control mechanisms that could be integrated into it.

4.3.1 Algorithm

Algorithm 4.2 describes the resulting adaptive MOLS algorithm used in this thesis. It is based on Algorithm 4.1 and adds the necessary structure to control the exploration strategy of the iterated inner MOLS (Algorithm 4.3). All other strategies are fixed since they were shown to be less impactful MOLS components on the PFSP (see Chapter 6).

Note that both the static Algorithm 4.1 and the adaptive Algorithm 4.2 become strictly identical when the INIT_ARM and CONTROL_ARM procedures always returns the same exploration procedure.

4.3.2 Related adaptive MOLS Algorithms

Other adaptive versions of MOLS algorithms have already been proposed. The most related to our adaptive algorithm is the Pareto Autonomous Local Search presented in Veerapen and Saubion (2011), in which several possible exploration

Algorithm 4.2: Adaptive Iterated Multi-Objective Local Search

Input: `archive`, a Pareto set of solutions
Output: the updated `archive` set

```
current ← archive;
/* Initialise all rewards */
INIT_REWARDS();
/* Select initial exploration strategy */
exploration ← INIT_ARM();
/* Apply the MOLS algorithm */
current ← MOLS(current, exploration);
/* Merge resulting archive and update rewards */
tmp ← pareto(archive ∪ current);
UPDATE_REWARDS(exploration, current, tmp);
archive ← tmp;
until termination criterion is met do
    /* Select exploration strategy */
    exploration ← CONTROL_ARM();
    /* Perturbation */
    current ← PERTURB(archive);
    /* Apply the MOLS algorithm */
    current ← MOLS(current, exploration);
    /* Merge resulting archive and update rewards */
    tmp ← pareto(archive ∪ current);
    UPDATE_REWARDS(exploration, current, tmp);
    archive ← tmp;
return archive;
```

Algorithm 4.3: Inner Multi-Objective Local Search (`mols`)

Input: A set of solutions, an exploration strategy
Output: A set of solutions

```
archive ← initial set of solutions;
until termination criterion is met do
    /* Select a random solution */
    selected ← select_1_rand(archive);
    /* Apply the given exploration strategy */
    reference ← archive;
    accepted ← exploration_strat(current, reference);
    /* Update archive with accepted neighbours */
    archive ← bounded_pareto(archive, accepted);
return archive;
```

strategies are controlled on two permutation problems, the QAP and the TSP, using a probability matching control mechanism. The main differences follow.

First, they focus controlling operators on solutions, directly in the MOLS iteration loop, and the MOLS is not iterated. On the contrary, we focus on improving the entire set of solutions, and the control is performed outside the MOLS iteration loop. Then, they use both the the relative change in quality (of a solution), and distance between the solutions, while we only use the relative change in hypervolume to capture both aspect of quality and diversity. Finally, they use a sliding window mechanism, when we will prefer in the experiments a more simple reinforcement learning mechanism (see Equation 7.5).

4.4 Configuration Scheduling

In the previous sections, we presented the instantiations of two MOLS algorithms: a static one (Algorithm 4.1), in which the parameters values are fixed before the search, and an adaptive one (Algorithm 4.2), in which a control mechanism modifies the exploration strategy during the search.

In this section, we propose a hybrid approach, that enables both the modification of parameter values during the search and the use of offline parameter configuration.

4.4.1 Proposition

Parameter configuration deals with a single target algorithm and its configuration space; its goal is to predict the configurations of the search space that are optimal on a given distribution of possible instances. Once a seemingly optimal configuration is found, selected, and finally used on an instance, this configuration is obviously used for the entirety of the run of the target algorithm. Automatic parameter configuration of MOLS algorithms is tackled in depth in Chapter 6.

On the other hand, parameter control enables to start the run from an initial configuration (possibly given by an automatic configurator) and to progressively adapt the configuration of the running algorithm to better fit the instance and ultimately use parameter values that are optimal for the given instance, rather than optimal for the entire possible distribution of instance. However, adaptation time is usually rather limited, restraining control to very few parameters and parameter values. Additionally, the optimal configuration may also depend of the state of the search, making strategies stronger in the initialisation phase or on the contrary better fitted for the final steps of the algorithm. Automatic parameter configuration of MOLS algorithms is tackled in Chapter 7.

Our proposition is to consider schedules of configurations. Instead of considering a single configuration for the entire run of the algorithm, or trying to controlling parameter values automatically, we will use a static timed sequence of multiple configurations. There are two major benefits of manipulating a static time-based sequence of configurations are. It uses no feedback from the search process, making it a *deterministic* parameter control approach (see Chapter 2) and avoiding to implement an online reward-based decision mechanism. It also enables the use of classical AAC tools with no restrictions on the number of parameters of being modified during the execution: indeed online mechanisms generally control very few parameters simultaneously, usually a single one and very rarely more than two at the same time.

4.4.2 Definitions

Given a configurable algorithm $\mathcal{A}$ and it associated configuration space Θ, let us first denote by $\mathcal{A}_{\theta,T}$ the algorithm obtained that use the specific configuration $\theta \in \Theta$ for a given time budget T. Then, let us define, given a maximal schedule length $K \geqslant 1$, $k \in \{1,\dots,K\}$ configurations $(\theta_1,\theta_2,\dots,\theta_k) \in \Theta^k$ of $\mathcal{A}$, and k time budgets $(T_1,T_2,\dots,T_k)$, the dynamic algorithm framework $\mathcal{F}_{(\theta_i)^k,(T_i)^k}$ obtained by sequentially applying in sequence every configuration θ_i using the respective time budget T_i, i.e., $(\mathcal{A}_{\theta_1,T_1},\mathcal{A}_{\theta_2,T_2},\dots,\mathcal{A}_{\theta_k,T_k})$. The total time budget of the dynamic algorithm framework is then $T = \sum_{i=1}^{k} T_i$. Note that algorithm $\mathcal{A}$ is *not* restarted

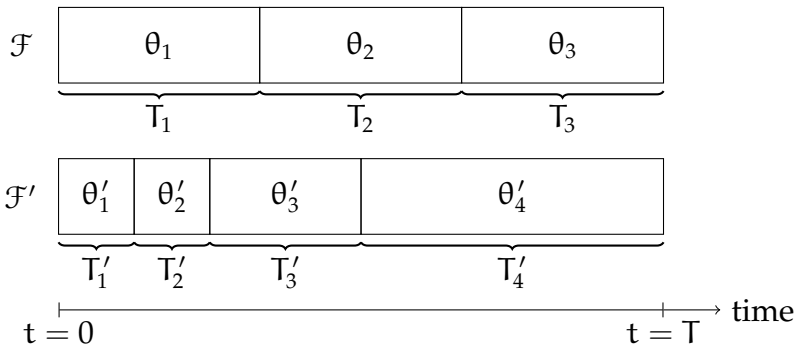


Figure 4.5 – Examples of two configuration schedules $\mathcal{F}$ and $\mathcal{F}'$

when the configuration is modified: at time $t = T_i$ the run seamlessly continues by switching from the configuration θ_i to configuration θ_{i+1}, in an online way.

The configuration space of the resulting framework is directly function of Θ, the configuration space of the original configurable algorithm. Indeed, if every possible parameter can be modified during the run, the resulting configuration space is directly related to the Cartesian product Θ^k. This exponential growth implies that considering both a long schedule length k and a large associated configuration space Θ is not recommended.

Two examples of dynamic algorithm frameworks $\mathcal{F}$ and $\mathcal{F}'$ are given in Figure 4.5. While the framework $\mathcal{F}$ uses $k = 3$ configurations to divide the total time budget into three interval of equal length, the framework $\mathcal{F}'$ use $k = 4$ configurations, using in quick succession two configurations in the beginning of the run and then using for more time the two other configurations as the search progresses.

4.4.3 Related Approaches

In addition to being related to parameter control or hyper-heuristics, as it enables modifications in the algorithm configuration during its execution, this approach of scheduling configurations is also similar in design to per-instance algorithm scheduling (see Chapter 2, e.g., Amadini et al., 2014; Hoos et al., 2015; Lindauer et al., 2016). There are however major differences. First, per-instance algorithm scheduling is related to algorithm selection, and deals with a portfolio of distinct, hopefully complementary algorithms. Configuration scheduling is more related to parameter configuration, as it deals with a single algorithm and its configuration space. Then, in per-instance algorithm scheduling the search is restarted every times to algorithm changes, while in our configuration scheduling approach simply continues the search with the updated configuration as in parameter control. Finally, the goal of per-instance algorithm scheduling is *robustness*: for every

instance at least one of the algorithm of the schedule has to be efficient on the instance. The goal of configuration scheduling is in the contrary *control*, to improve the performance compared to using a single configuration for the entire run.

4.5 AMH: Adaptive MetaHeuristics

Algorithm 4.1, together with the other MOLS implementations of this chapter, has been implemented in C++ in AMH (Adaptive MetaHeuristics), a framework specifically designed during this thesis.

This framework enables the automatic construction of algorithms, given a functional description of their components, adapting their structures to the parameter values given at the beginning of their execution. For the need of the other MOLS instantiations, this C++ framework has also been designed to enable the seamless redesign of the algorithm structures during their executions.

4.5.1 Motivation

This framework was designed according to the experimental needs of the following chapters, in order to facilitate the automatic construction, control, and integration into automatic design tools. Indeed, the experiments of the following chapters have strong, different, requirements, that may be difficult to achieve using frameworks that are not dedicated to automatic algorithm design.

First, automatic configurators require an easy command line interface to provide a parameter value to each parameter of the target algorithm. The target algorithm should be able to easily use this description of the parameter values to construct the correct algorithm to subsequently execute it. If automatically using numerical values from the command line may be easy, automatically construct the flow of the algorithm with regard to design choices only available only at runtime may not be straightforward. On the other hand, hyper-heuristics and parameter control techniques, as well as our scheduling approach, require a flexible structure in which every parameter, every strategy or the algorithm itself may be modified and adapted during its execution.

ParadisEO [1] (C++) and jMetal [2] (java) are both well-known frameworks of the literature. They are dedicated to the design of metaheuristics and provide many tools to algorithm designers to write *static* metaheuristics and algorithms. They can easily accommodate the use of automatic algorithm configuration tools and

[1] http://paradiseo.gforge.inria.fr/
[2] https://jmetal.github.io/jMetal/

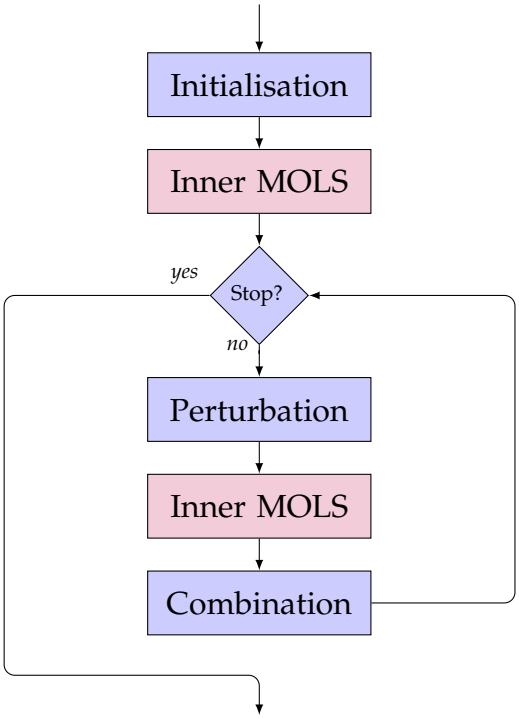


Figure 4.6 – Execution flow of an iterated MOLS algorithm

use parameters to configure the metaheuristic at runtime. However, their have not been designed to take into account the possibility to fully modify an algorithm during its execution, which, even if possible, remains very difficult and left as exercise to the algorithm designer.

4.5.2 Philosophy

Metaheuristic can very often be viewed as a succession of individual steps, such as for example for our MOLS algorithm the selection, exploration and archive steps. Therefore, we can associate a metaheuristic to its specific flow of execution. This flow can be as simple as linear, can involve Boolean branching (e.g., to create loops), and involve routines using other flows of execution. In the case of adaptive algorithms, this flow can be temporally or definitely rewritten.

Figure 4.6 illustrates the execution flow of an the iterated MOLS algorithm of Algorithm 4.1. Every component of the execution flow, and therefore of the algorithm, can be seen as a function taking as input and output a set of solutions. Any coherent part of the algorithm: a single component, a sequence of components, as well as the entire loop, or the entire algorithm, can ultimately be seen as such a function. As example, the *Inner MOLS* component actually represent a similar loop-based execution flow.

Numerical parameter usually does not interfere or influence the execution flow of the algorithm. On the other hand, categorical parameters such as strategy ones directly determine the execution flow. For a static metaheuristic, the construction of the execution flow is done before the run and not modified during the execution, whereas the execution flow of an adaptive metaheuristics may be fully modified.

AMH is designed to automatically construct the execution flow before the execution and of control it during the execution of the metaheuristic. The graph representing the structure of the execution flow, with all its components and branching, can easily be instantiated using parameters provided as input of the algorithm, thus facilitating algorithm configuration. This structure being manually managed and traversed, rather than irredeemably compiled, it becomes possible to modify it during its own execution, allowing a natural control over its components, and more generally speaking, allowing its adaptation.

4.5.3 Design and Implementation

The AMH framework is implemented in C++. It handles the execution flow of a given algorithm by encapsulating algorithmic operations into components and describing their temporal interactions. All algorithms implemented in AMH inherit from a base function class – a single class representing a function – i.e., a delimited part of the execution having defined input and output types, which are specified at compile-time using templates. Moreover, AMH also provides a large range of *execution flow primitives* such as conditions and loops, in order to easily connect all parts of an implemented algorithm.

The core design of AMH is to only focus on the flow of execution, and not on the solving methods. Indeed, the algorithm designer may provide any solution representations and solving mechanisms, as long as they are can represent types and functions. The types of every input and output, (i.e., usually solution representations) are used in template at compile-time, thus validating the correction of the final execution flow using the C++ compiler. Solving mechanisms need to be encapsulated, either as static classes inheriting from the base AMH function class, or dynamically as native C++ functions. In particular, it means that existing C++ algorithm implementations (e.g., metaheuristics implemented under ParadisEO) can benefit from AMH just by defining atomic components and encapsulating them. Finally, an expected consequence of such a functional approach is that hybridisation of algorithms using the same solution representation is immediate.

4.5.4 Execution Flow Examples

In the following, we present three examples of execution flows, using the MOLS algorithms of the following chapters.

First, Figure 4.7 illustrates the execution flow of the dynamic Algorithm 4.2 investigated in Chapter 7, while focusing on the facility to create "complex" execution flows. It integrate control and feedback components that share the information of the rewards associated to each execution path. The path to chosen regarding the choice made during the control step.

Then, Figure 4.8 also illustrates the execution flow of the dynamic Algorithm 4.2, but highlighting another feature of AMH: during the control step the execution flow is updated by detaching one of the component (the MOLS step of the previous iteration) and replacing it by a new one, dynamically constructed.

Finally, Figure 4.9 highlights how easily a schedule of MOLS algorithm can be constructed, simply by chaining different components constructed using the different configurations of the schedule.

4.6 Perspectives

In this chapter, we presented and discussed three different instantiations of MOLS algorithms: a highly parameterised static MOLS algorithm, an adaptive MOLS algorithm that enable the integration of generic parameter control mechanisms, and finally schedules of MOLS algorithms; together with the framework we designed to implement them.

In the following, we detail two perspectives related to these implementations.

More complete static MOLS algorithm. Many of the strategies that have been presented in Chapter 3 have not yet been integrated in the search space of the algorithms described in Chapter 4, and could enrich the Table 4.1 with many new original combinations of strategies.

In particular, all the considered strategies are based on Pareto dominance, while the aggregation-based strategies have been set aside. The selection and bounding strategies have also been restricted to very simple mechanisms, while other, more complex and time consuming, could also have been included (e.g., strategies based on the distribution of the solutions in the archive). Furthermore, fine-tuning parameters such as `select-size`, `explor-size`, and `perturb-size` have been used as absolute values, while alternative parameters considering percentages of

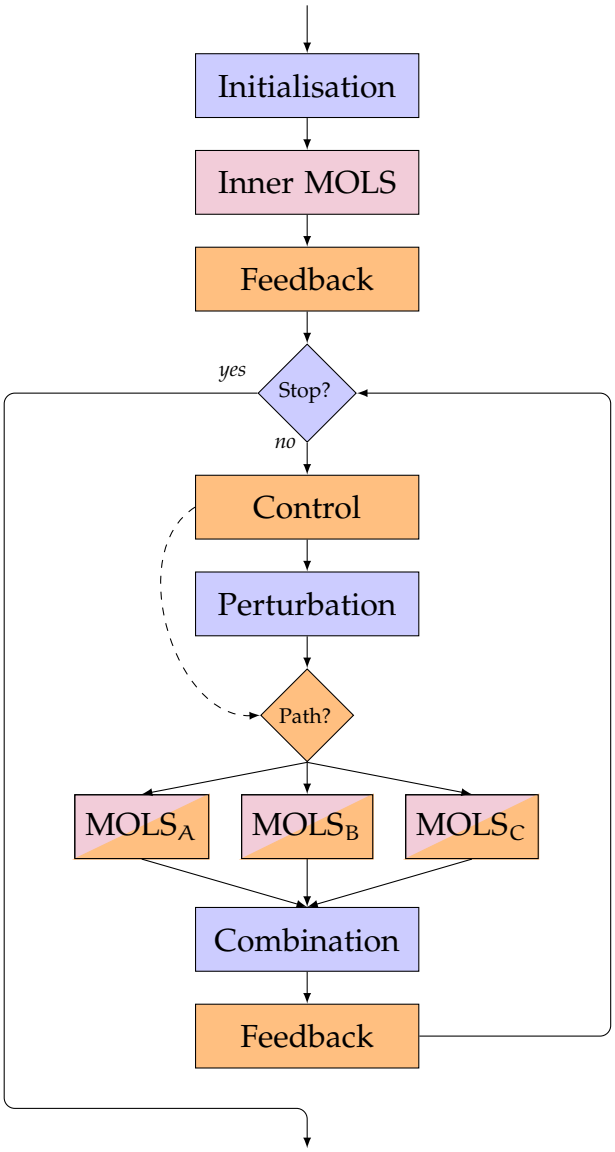


Figure 4.7 – Execution flow of an adaptive algorithm using multiple paths

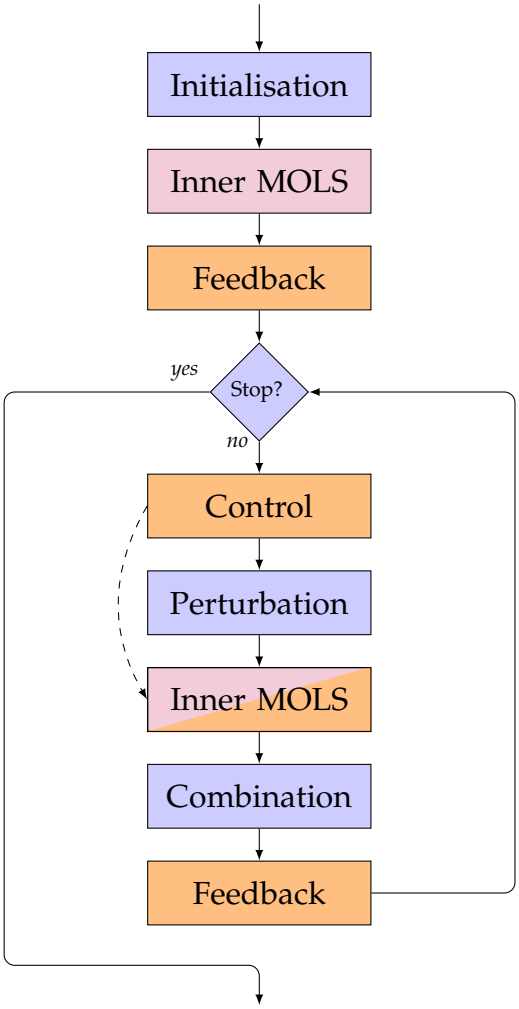


Figure 4.8 – Execution flow of an adaptive algorithm using reconstruction

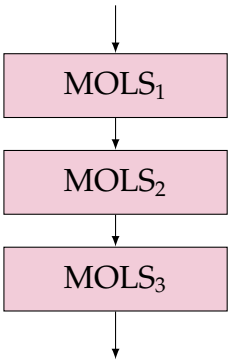


Figure 4.9 – MOLS schedule

the current size of the archive or the size of the neighbourhood have not been investigated. Finally, complex exploration stopping criteria, such as limiting the size of the explored neighbourhood or using hybrid conditions on neighbours also not have been investigated.

Other decisional schedules. In Chapter 4, we only considered very basic decisional schedules, in which the decisions are decided, and the reward are updated, every iteration, or after a set number of iterations.

More complex decisional schedules can also be considered, and involve for example multiple learning and exploitation phases, that alternate periods specifically dedicated to update the predicted quality of every strategies, and periods that only use the resulting predicted best strategies.

Part III

Automatic Offline Design

> Design is not just what it looks like and feels like. Design is how it works.
>
> Steve Jobs

Chapter 5

MO-ParamILS

> Study the past if you would define the future.
>
> Confucius

In this chapter, we introduce MO-ParamILS, a multi-objective automatic algorithm configurator. First, we discuss the motivations of multi-objective automatic configuration. Then, we present ParamILS, a prominent single-objective algorithm configurator. After discussing the possible uses of ParamILS in a multi-objective context, we present MO-ParamILS, our extension of ParamILS specifically designed for multi-objective configuration. Finally, we validate our multi-objective framework on multiple configuration scenarios.

This chapter contributions are closely linked to the following publication:

- **Blot, A.**, Hoos, H. H., Jourdan, L., Kessaci-Marmion, M., and Trautmann, H. (2016). **MO-ParamILS: A multi-objective automatic algorithm configuration framework.** In Festa, P., Sellmann, M., and Vanschoren, J., editors (2016). *Learning and Intelligent Optimization – 10th International Conference, LION 10. Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*, pages 32–47. Springer.

5.1 Multi-objective Automatic Configuration

5.1.1 Definition

Multi-objective automatic algorithm configuration (MO-AAC) naturally arises when the performance of an algorithm is not or can not be summed up into a single value such as running time or solution quality.

Formally, a multi-objective configuration problem consists in a direct extension of the single-objective configuration problem (Equation 2.2) when the original performance indicator is a vector of performance indicators. Using the notations of Chapter 2, given a parameterised target algorithm A, the space Θ of configurations of A, a distribution of instances $\mathcal{D}$, and a statistical population parameter E, with A_θ denoting the association of the parameterised algorithm A with the configuration θ, Equation 2.2 becomes Equation 5.1 when the original performance indicator $o : \Theta \times \mathcal{C} \to \mathbb{R}$ becomes $O : \Theta \times \mathcal{C} \to \mathbb{R}^n$, i.e., a vector of n performance indicators $O(A, i) = (o_1(A, i), o_2(A, i), \ldots, o_n(A, i))$.

$$\begin{cases} \text{optimise} & E[O(A_\theta, i),\ i \in \mathcal{D}] \\ \text{subject to} & \theta \in \Theta \end{cases} \tag{5.1}$$

As in Equation 2.2, the supposition is made that the limit implied by Equation 5.1 exists and is finite. Therefore, every component of the cost vector can be optimised independently in a multi-objective fashion as the problem becomes a standard multi-objective optimisation problem (Equation 5.2).

$$\begin{cases} \text{optimise} & (E[o_1(A_\theta, i),\ i \in \mathcal{D}], \ldots, E[o_n(A_\theta, i),\ i \in \mathcal{D}]) \\ \text{subject to} & \theta \in \Theta \end{cases} \tag{5.2}$$

Note that we will prefer to use the term *performance indicator* rather than the previously used term *cost metric* as it does not strictly relate to the mathematical definition of a metric. We will nevertheless continue to suppose that every indicator is a cost to be minimised.

As mentioned in Chapter 2, note that in practice, automatic configuration deals with a finite set of training instances rather than a distribution of instances, meaning that the quality $E[O(A_\theta, i),\ i \in \mathcal{D}]$ of a configuration θ over the entire distribution $\mathcal{D}$ will be approximated rather than really computed. We denote by *level of detail* of a configuration the number of instances on which the configuration was evaluated. It is of course supposed that the set of training instances is sufficiently large and representative of the underlying distribution so that the approximated quality is reliable when the level of detail is high enough.

5.1.2 Use Cases

Single-objective automatic algorithm configuration (SO-AAC) optimises the quality of a given target algorithm according to a single performance indicator. There are two main use cases: either the end-user has a fixed budget of computation time

and is interested in having the best solution quality, or a given solution quality is targeted and the running time to achieve it is optimised instead.

Use cases of MO-AAC are more diverse. The most straightforward use case is the simultaneous optimisation of both the running time and the solution quality of the target algorithm. That is, the end user is interested by both achieving the best solution quality and achieving it in the shortest amount of time. Auxiliary indicators such as the memory or energy consumption can also be considered in addition to other performance indicators. While the performance of multi-objective algorithms may be assessed using a single multi-objective indicator, with MO-AAC several multi-objective indicators can be used simultaneously, enabling multi-objective target algorithms to be configured according to multiple, complementary, indicators, instead of having to rely on a single one. Finally, any number of performance indicators may be optimised simultaneously without the need to aggregate them.

5.2 Single-objective ParamILS

ParamILS is an automatic configurator proposed by Hutter et al. (2007, 2009). While it was presented in the primary context of optimising the running time of the target algorithm, we present it here in the more general case of optimising an unknown performance indicator.

5.2.1 Core Algorithm

The core algorithm of ParamILS is given by Algorithm 5.1. It is based on an iterated local search (Lourenço et al., 2003; Hoos and Stützle, 2004), in which the best solution is iteratively improved by mean of both local search and perturbation mechanisms. Three parameters are exposed: the number r of initial random configurations, a restart probability p_{restart}, and the number s of random search steps performed in each perturbation phase.

ParamILS starts by considering r random configurations, in order to compare the initial (usually default) configuration to a few others to make sure of its relevance. Then, it applies a local search procedure (see Procedure 5.2), which is based on the *one-exchange neighbourhood*, i.e., modifying a single parameter value at a time. A tabu mechanism is also used to ensure that the configurator is never stuck. Between iterations, there is a p_{restart} chance to restart the search from a new configuration, chosen uniformly at random from the search space. Otherwise, a perturbation of s random steps is performed.

Algorithm 5.1: Single-objective ParamILS

Exposed parameters: r, $p_{restart}$ and s
Input: Initial configuration
Output: The incumbent, i.e., the overall best configuration found
Side effect: Updates the cache and the incumbent

```
/* Initialisation */
current_config ← initial configuration;
for i ← 1...r do
    tmp ← random configuration;
    update(tmp, current_config);
    current_config ←compare(current_config, tmp);

/* Iterated local search */
until termination criterion is met do
    /* Perturbation */
    if first iteration then
        tmp ← current_config;
    else
        with probability p_restart then // Restart
            /* incumbent is not forgotten */
            current_config ← random configuration;
            tmp ← current_config;
        otherwise // Random walk
            tmp ← current_config;
            for i ← 1...s do
                tmp ← random neighbour of tmp;

    /* Local search */
    tmp ← local_search(tmp);
    current_config ← compare(current_config, tmp);
return incumbent;
```

Procedure 5.2: localsearch(config)

Input: Initial configuration
Output: The best configuration found
Side effect: Updates the cache and the incumbent

```
current_config ← initial configuration;
tabu_set ← {current_config};
repeat
    foreach neighbour ∈ randomised neighbourhood of current_config
     do
        if neighbour ∈ tabu_set then
            next ;
        else
            tabu_set ← tabu_set ∪ {neighbour};
        update(neighbour, current);
        if compare(current, neighbour) = neighbour then
            current_config ← neighbour;
            break ;
until every neighbour of current_config is tabu;
return current_config;
```

Two essential elements of ParamILS are missing from Algorithm 5.1: the handling of both the global cache of runs and the incumbent. Indeed, due to the usually very high cost of evaluating configurations of the given target algorithm, ParamILS maintains a global cache of the results of all target algorithms performed during the search, thus avoiding to repeat superfluous costly algorithm runs.

The incumbent is the best configuration that was found by ParamILS, which is here handled completely implicitly. Both the incumbent and its quality are updated automatically every time a configuration is evaluated. Finally, the incumbent is not reset when the current configuration of the iterated local search is restarted.

The auxiliary functions used in Algorithm 5.1 and Procedure 5.2 are described hereafter.

`update(config, reference)` enforces that the configuration `config` can be compared to the configuration `reference`. This function is specified according the particular version of ParamILS used: BasicILS or FocusedILS.

`compare(config, challenger)` compares a *current* configuration `config` to another configuration `challenger` and returns the new *current* configuration, i.e., the later one if it deemed more promising or the former one otherwise. In the current ParamILS implementation, the challenger is accepted if its level of detail is at least equal, and if it has a better or equal quality (Procedure 5.3).

`detail(config)` returns on how many instances the configuration `config` has been run, according to the global cache. The instances being always considered incrementally, this number is sufficient to know on which instances the configuration has been evaluated.

`quality(config, insts)` returns the mean quality resulting of the runs of the configuration `config` on the instances `insts`.

There are two versions of ParamILS, that differ on how the procedure `update` is handled. While BasicILS uses a fixed set of instances to evaluate the performance of every configuration, FocusedILS uses a variable set of instances, performing less runs on configurations of poorer quality to focus on the most promising ones. These two versions are detailed in the following. A presentation of capping mechanisms, able to further improve the performance of ParamILS, immediately follows.

Procedure 5.3: compare(config, challenger)

Input: Configurations `config` and `challenger`
Output: A configuration

if *`detail(challenger)`* $<$ *`detail(config)`* **then**
    return `config`;
`insts` $\leftarrow$ the `detail(config)` first instances;
if *`quality(challenger, insts)`* $\leqslant$ *`quality(config, insts)`* **then**
    return `challenger`;
else
    return `config`;

Procedure 5.4: update(config, reference)

Input: Configurations `config` and `reference`
Exposed parameters: n
Side effect: Updates the cache regarding `conf` and `reference`

`insts` $\leftarrow$ the n instances;
foreach *instance* *`i`* $\in$ *`insts`* **do**
    `cache[reference,i]` $\leftarrow$ performance of `reference` over `i`;
    `cache[config,i]` $\leftarrow$ performance of `config` over `i`;

5.2.2 BasicILS, FocusedILS

BasicILS is the most simple version of ParamILS. It uses a single parameter, n, which specifies how many instances are needed to compare configuration performance. This allows for every configuration to have exactly the same level of detail, because the procedure `update` (Procedure 5.4) always use the same set of instances. This set of instances is selected uniformly at random (without replacement) from the given training set at the beginning of the algorithm.

However, BasicILS has major issues, mainly due to the difficulty of choosing the value of the parameter n. Chosen too small, solution quality can be inaccurate, leading to poor generalisation of the final configuration to unseen test instances. Chosen too large, much effort will be wasted on evaluating poor performing configurations, compromising the efficiency of the search process. Additionally, if after n instances the quality of two configurations seems almost identical there is no possibility to further refine them to take an adequate decision.

To overcome these disadvantages, a more advanced version of ParamILS, FocusedILS, was simultaneously proposed. The key idea behind FocusedILS is to avoid the potential problems arising from the use of a fixed number of instances for evaluating configurations by starting comparisons between configurations on a small initial set of instances and then increasing the number of instances as better and better configurations are found.

Procedure 5.5 outlines the `update` procedure of FocusedILS. Compared to the original version of Hutter et al. (2007), we propose to introduce two optional parameters n_{min} and n_{max} to better control the level of detail of the configurations; values of $n_{min} = 0$ and $n_{min} = \infty$ effectively disable them. This procedure follows three steps. First, a minimum level of detail n_{min} is enforced. Then, until a choice between the two configurations can be made according to the `compare` procedure, effectively when one of the two configurations dominates the other, their levels of detail are increasingly updated. Finally, if the two configurations have the same level of detail, the resulting configuration go through an intensification procedure.

In Procedure 5.6, we propose an alternative, more efficient variant of the original intensification mechanism, which performs a variable number of runs based on the time spent since intensification was last performed. The proposed intensification takes a configuration in input and performs new runs until its new quality dominates the previous one. To avoid spending too much time in the intensification procedure, a maximum number of evaluation could be specified. Another alternative intensification mechanism, less efficient, might also simply perform a fixed number of new runs.

5.2.3 Adaptive Capping Strategies

Adaptive capping strategies enable to stop evaluating a configuration when it becomes clear that the configuration will be discarded, in order to avoid wasting time evaluating poor configurations.

Indeed, with sufficient knowledge of the possible performance of the target algorithm (in particular, the minimal value of the performance indicator), it is possible to compute a lower bound of the performance approximation on a given number of instance. By comparing this lower bound to the quality of the configuration being compared to, the `update` procedure (Procedure 5.4; Procedure 5.5) can then be stopped early. This improvement, denoted as *trajectory-preserving capping* in Hutter et al. (2009), has the property of not modifying the result of a given local search of ParamILS due to its tabu mechanism. This property generalises to the full trajectory of BasicILS runs; however not for FocusedILS as it may impact further comparison to a capped configuration.

Procedure 5.5: update(config, reference)

Input: Configurations `config` and `reference`
Exposed parameters: $n_{min} = 0$, $n_{max} = \infty$
Side effect: Updates the cache regarding `conf` and `reference`

```
/* Pre-comparison */
insts ← the n_min first instances;
foreach instance i ∈ insts do
    cache[config,i] ← performance of config over i;
    cache[reference,i] ← performance of reference over i;

/* Comparison */
repeat
    n_config ← detail(config);
    n_reference ← detail(reference);
    if n_config > n_reference then
        inst ← the (n_config + 1)th instance;
        cache[config,inst] ← performance of config over inst;
    else if n_config < n_reference then
        inst ← the (n_reference + 1)th instance;
        cache[reference,inst] ← performance of reference over
         inst;
    else if n_config = n_max then
        break ;
    else
        inst ← the (n_config + 1)th instance;
        cache[config,inst] ← performance of config over inst;
        cache[reference,inst] ← performance of reference over
         inst;
until compare(config, reference) = reference or
 compare(reference, config) = config;

/* Post-comparison */
if detail(config) = detail(reference) then
    if compare(config, reference) = reference then
        intensify(reference);
    else
        intensify(config);
```

Procedure 5.6: intensify(config)

Input: Configuration `conf`
Side effect: Updates the cache regarding `conf` and `reference`

loop do
  q ← `quality(conf);`
  `inst` ← the (`detail(config)`+1)th instance;
  `cache[config,inst]` ← performance of `config` over `inst`;
  if *`quality(conf)`* $\leqslant$ q **then**
    break ;

A second capping mechanism was also introduced, denoted as *aggressive capping*, in which the incumbent is used in comparison to the lower bound instead of the current best solution of the local search. An additional parameter is then required to allow some slack in the accepted performance values, without which the local search would stale until a neighbour of the current best solution actually improve the overall incumbent.

Finally, in the case of optimising the running time of the target algorithm, both capping strategies can be further extended to enable early termination of the target algorithm, at the possible cost of having to rerun it on an instance on which it had been capped later in the search.

5.2.4 Configuration Protocol

In order to use ParamILS, it is required to follow a machine learning protocol. This protocol comprises either two steps (namely: training and test) or three steps (namely: training, validation, and test). In both cases, two sets of instances are required: a set of *training instances* and a set of *test instances*, which should be distinct from the first set and kept unseen during training. Training instances are used to build a prediction of the best configuration of the target algorithm, while test instances are used to verify that the quality of the predicted configuration actually generalises to other instances in order to avoid over-fitting.

Training: ParamILS is used on the training instances in order to predict the best configuration. Generally, only a subset of instances is used because there are more training instances than the expected level of detail for configurations. For three reasons: (i) ParamILS is a stochastic algorithm, (ii) the sample subset of training instances might have a great impact of the final configuration, and (iii) since the order of the instances itself may compromise the efficiency

of FocusedILS search strategy, it is recommended to perform multiple, independent, runs of ParamILS using different subset and ordering of the training instances. Note that these multiple training runs can typically be conducted in parallel.

Validation: Two problems arise at the end of the training step. First, if there are sufficiently enough training instances, the different subsets used during the training steps may be different. Furthermore, in the case of FocusedILS, the quality of the final configurations may not have been approximated using the same number of instances. For these reasons, in order to fairly compare the performance of all ParamILS runs, the quality of every final configuration should be reassessed on the same subset of training instances. The reassessment of each configuration may again be performed independently in parallel. This step may be skipped in two cases: if there is few enough training instances so that every final configuration was assessed on the exact same instances, or if the level of detail of each final configuration is high enough so that they are considered a very good approximation of their quality over the complete distribution of instances.
At the end of this step, only the best configuration is considered.

Test: Finally, to verify that the configuration resulting from the validation step actually generalises over previously unseen instances, its quality is reassessed once more, this time on a subset of the test instances. Again, the reassessment of each configuration may be performed independently in parallel.

5.3 Multi-objective ParamILS

In this section, we propose MO-ParamILS, a multi-objective extension of the single-objective configurator ParamILS, able to simultaneously optimise the performance of the target algorithm with regard to multiple performance indicators.

5.3.1 Motivations

The main motivation behind MO-ParamILS is to propose an efficient inherently multi-objective configurator that could be used without requiring any additional specific knowledge, such as for example an a priori aggregation of the different performance indicators.

To obtain a multi-objective configurator, we choose to extend a very efficient and well-known single-objective configurator, ParamILS. Indeed, ParamILS relies on a single-objective iterated local search procedure, and literature shows a large number of examples of efficient multi-objective local search algorithms (see

Chapter 3). Other options would have included extending other well known single-objective configurators such as for example SMAC (Hutter et al., 2011), or irace (López-Ibáñez et al., 2016), for which works for multi-objective contexts has also recently been carried out (see Zhang et al., 2015, 2016, 2018).

5.3.2 Core Algorithm

We now describe our multi-objective extension of the ParamILS framework, outlined in Algorithm 5.7. The main difference with ParamILS (Algorithm 5.1) lies in the use of a multi-objective iterated local search process (Procedure 5.8), in which an *archive* (i.e., set of non-dominated configurations) is iteratively modified rather than a single configuration of the given target algorithm. This search process is directly related to the multi-objective local search algorithms discussed in Chapter 3. Likewise, rather than a single incumbent, an archive of incumbent is updated during the whole configuration process. MO-ParamILS exposes the same three parameters as ParamILS: the number r of initial random configurations, the number s of random search steps performed in each perturbation phase and the restart probability p_{restart}.

The initialisation of the search process does not conceptually change, except that an initial set of default configurations can be provided and is combined, with the r randomly chosen configurations, into an archive. We ensure that whenever we add a new configuration to an archive, all Pareto-dominated configurations are discarded (see Function 5.9), so that the archive always contains only non-dominated configurations.

MO-ParamILS make use of the same auxiliary functions than ParamILS, with some minor differences. The `quality` function now returns a vector of qualities. The `compare` function now compares configurations using Pareto dominance: if ParamILS version accepted configurations of better or equal quality (see Procedure 5.3), the MO-ParamILS version similarly accepts configurations *that are not of worse quality*, i.e., configurations with either better, equal, or incomparable quality.

The restart mechanism mostly remains unchanged: it now replaces the current archive with one containing a single configuration chosen uniformly at random from the entire configuration space Θ. As for the perturbation mechanism, the original perturbation of ParamILS is used on a single configuration from the current archive, chosen uniformly at random, to obtain a single configuration stored as a new archive and used as the starting point of the subsequent local search phase (Geiger, 2008).

The subsidiary local search process used in MO-ParamILS is outlined in Procedure 5.8. From the wide range of existing multi-objective local search (MOLS)

Algorithm 5.7: Multi-objective ParamILS

Exposed parameters: r, $p_{restart}$ and s
Input: Initial archive of configurations
Output: The archive of incumbents, i.e., the overall best configurations found
Side effect: Updates the cache and the archive of incumbents

```
/* Initialisation */
current_arch ← initial archive;
for i ← 1...r do
    tmp ← random configuration;
    update(tmp, current_arch);
    current_arch ← archive(current_arch, tmp);

/* Iterated local search */
until termination criterion is met do
    /* Perturbation */
    if first iteration then
        tmp ← current_arch;
    else
        with probability p_restart then // Restart
            /* incumbents are not forgotten */
            current_arch ← { random configuration };
            tmp ← current_arch;
        otherwise // Random walk
            config ← current_config;
            for i ← 1...s do
                config ← random neighbour of tmp;
            tmp ← { config };
    /* Local search */
    tmp ← local_search(tmp);
    foreach config ∈ tmp do
        update(config, current_arch);
        current_arch ← archive(current_arch, config);
return the archive of incumbent;
```

Procedure 5.8: localSearch(init_arch)

Data: Initial archive of configurations
Result: Best archive of configurations found
Side effect: Change or update the incumbent if necessary

```
current_arch ← initial archive;
tabu_set ← current_arch;
repeat
    /* Selection */
    candidate_set ← ∅;
    foreach current_config ∈ current_arch do
        foreach neighbour ∈ randomised neighbourhood of
         current_config do
            /* Exploration */
            if neighbour ∈ tabu_set then
                next ;
            else
                tabu_set ← tabu_set ∪{neighbour};
            update(neighbour, current_config);
            if compare(neighbour, current_config) =
             neighbour then
                candidate_set ← candidate_set ∪{neighbour};
                break ;
            else if compare(current_config, neighbour) =
             neighbour then
                candidate_set ← candidate_set ∪{neighbour};
    /* Archive */
    foreach conf ∈ candidate_set do
        current_arch ← archive(current_arch, conf);
until candidate_set = ∅;
return current_arch;
```

Function 5.9: archive(arch, challenger)

Input: Archive `arch`, configuration `challenger`
Output: Updated archive `arch`

```
foreach config ∈ arch do
    if compare(challenger, config) = challenger then
        arch ← arch \ { config };
    else if compare(config, challenger) = config then
        return arch;
arch ← arch ∪ { challenger };
return arch;
```

procedures (see Chapter 3), it is based on a MOLS algorithm that is conceptually simple and resembles the original subsidiary local search procedure used in ParamILS; this MOLS algorithm in particular has also been previously shown to be very efficient (Blot et al., 2015). At each step of the local search process, every configuration of the current archive is explored individually and sequentially. When exploring a given configuration `config`, its neighbours are evaluated in random order (excluding any configurations already visited earlier in the same local search phase, using ParamILS tabu mechanism), until one is found that strictly dominates `config` or all neighbours have been visited. All non-dominated neighbours encountered during this process are added to the current archive, making sure that dominated solutions are removed; this can be seen as a generalised version of the acceptance criterion used in the single-objective ParamILS framework. The local search then stops when no more unvisited neighbour can be added to the archive. Of course, the local search also stops when the termination criteria of MO-ParamILS is met (no more budget, or global wall clock time exceeded), even if not explicitly handled here.

Finally, as the `quality` function now returns a vector of mean qualities rather than a scalar mean quality, in the `compare` function (Procedure 5.3) the condition is changed so that it returns the `challenger` configuration if it is either of better or equal quality, or if its quality is incomparable.

Following the existing versions of ParamILS, we also propose MO-BasicILS and MO-FocusedILS, two versions of MO-ParamILS. Both versions are direct equivalents of BasicILS and FocusedILS, with only very slight difference: regarding FocusedILS, the stopping criterion of the `intensify` mechanism (Procedure 5.6) is changed to stop the procedure if the new quality dominates the previous quality.

5.3.3 Configuration Protocol

The configuration protocol of MO-ParamILS is a direct multi-objective extension of the ParamILS configuration protocol, presented in the last section.

Training Run MO-ParamILS, multiple times independently with different ordering and subsets of the training instance set. Each run results in an archive of configurations.

Validation Reassess the quality of each final training configuration on the same subset of training instances, *regarding all performance indicators*. Filter dominated configurations out, to focus on the best configurations of the training set.

Test Reassess the quality of final validation configurations on the previously unseen test instances.

Again, MO-ParamILS runs of the training step are independent and can be conducted in parallel, which is also the case of both reassessments of validation and test steps.

5.4 Hybrid Multi-Objective Approaches

The configuration protocol of ParamILS and MO-ParamILS enable them to tackle single-objective and multi-objective configuration scenario, respectively. In this section, we discuss the direct use of ParamILS on multi-objective configuration scenarios. Specifically, we will consider a bi-objective scenario in which the performance of a target algorithm is optimised regarding two performance indicators o_1 and o_2.

5.4.1 Single Performance Indicator

The most straightforward approach that can be used is to first simply use ParamILS on a single performance indicator, i.e., either o_1 or o_2 independently, while completely ignoring the second performance indicator. As multiple runs of ParamILS are recommended in any case, the training budget time can easily be divided to train over the different performance indicators.

In the experiments of the following section, we show examples in which the training of ParamILS focus on solution quality (o_1) while being divided for multiple values of running time (o_2); and in which the training is evenly divided between the two performance indicators (solution quality and memory usage).

This approach only modifies the training step of the MO-ParamILS configuration protocol. The original multi-objective validation and test steps then follow, in

which all performance indicators are simultaneously assessed. While the training does not ultimately use the same objectives than the two following steps of the protocol, this problem is alleviated by first the validation step, which is performed in a multi-objective way on the training instances, and second by the multiple runs of the configurator recommended, that allow the training step to return multiple configurations.

5.4.2 Aggregation of Multiple Performance Indicators

The obvious downside of the previous approach is that it can only search configuration along the directions of the independent performance indicators in the multi-objective space.

The second approach that we propose is to use a weighted linear scalarisation of the two performance indicators o_1 and o_2 in order to obtain a single metric o_{agg}. With the addition of an aggregation coefficient $\alpha \in [0, 1]$ such that $o_{agg} = \alpha \cdot o_1 + (1 - \alpha) \cdot o_2$, this approach is able to optimise both performance indicators in a specific direction of the objective space, while enabling the use of existing single-objective configuration tools.

However, two questions arise. First, how to choose the value of the aggregation coefficient α? Then, how should the performance indicators o_1 and o_2 be normalised? Indeed, the coefficient α directly defines the direction in which the search will be conducted, and even very small variations of α may change the theoretical optimal configuration. If there is no clear relation or preference between the different cost function this will result in a very hard choice for the end user. Furthermore, such an aggregation will presuppose that both performance indicators are correctly normalised, otherwise the precise direction determined by the coefficient α will have no real meaning. One way to answer both question would be to sufficiently know in advance the quality of optimal configuration, which can only be approximated with costly preliminary experiments.

Finally, again, this approach only modifies the training step of the multi-objective configuration protocol. The original validation and test steps then follow, with the same upsides and downsides.

5.5 Framework Evaluation

In this section, we investigate the worth of MO-AAC and the efficiency of MO-ParamILS over several multi-objective configuration scenarios.

First, we investigate the trade-off between running time and solution quality for an anytime optimisation algorithm on multiple configuration scenarios.

Table 5.1 – Configuration scenarios

Dataset	Target	Training	Performance objectives	Abbrv.
Regions200	CPLEX	1 day	[quality, cutoff]	RCut
Regions200	CPLEX	1 day	[quality, running time]	RRun
CORLAT	CPLEX	1 day	[quality, cutoff]	CCut
CORLAT	CPLEX	1 day	[quality, running time]	CRun
QUEENS	CLASP	1 day	[memory usage, running time]	QUEENS

Our second example involves the simultaneous optimisation running time and memory usage. While both cases involves only two performance indicators, MO-ParamILS is not restricted to such bi-objective algorithm configuration problems.

5.5.1 Experimental Protocol

The three AAC approaches compared in the experimentation are as follows. First, we consider two MO-AAC approaches, that use MO-FocusedILS and MO-BasicILS, respectively. Regarding the MO-BasicILS approach, its parameter n is set to 100, meaning that estimations of configuration performance will use 100 training instances. Then, we consider a SO-AAC approaches that use the original single-objective ParamILS configurator, that we will refer to as SO-ParamILS. This second approach will use FocusedILS with every recommended improvement enabled (e.g., aggressive capping). Finally, these three AAC approaches are also compared to a baseline obtained by simply using the default configuration of the target algorithm.

The five configuration scenarios we consider are described in Table 5.1. These scenarios use three datasets and two target algorithms, which belong to ACLib[1], a comprehensive algorithm configuration library. They are already known and have been studied as single-objective algorithm configuration problems. Details of the two target algorithms are precised in Table 5.2. Note that as the neighbourhood relation of ParamILS requires every parameters to be categorical, ACLib provides discretised sets of values for the integer and continuous parameters of the target algorithms.

We first investigate the trade-off between solution quality and running time for the commercial solver CPLEX, a very well known and highly parameterised mixed integer programming (MIP) optimiser, on two different existing MIP data-

[1] http://aclib.net

Table 5.2 – Target algorithm parameters (with number of possible values)

Algorithm	Categorical	Integer	Continuous	Total configurations
CPLEX	5 (2)	65 (2–7)	2 (5–6)	$2.26 \cdot 10^{46}$
CLASP	15 (2–5)	43 (2–16)	8 (6–14)	$9.96 \cdot 10^{48}$

sets, Regions200 and CORLAT. To achieve this, we consider the cutoff time (i.e., the maximum running time) as an additional parameter with five possible values: 1, 2, 3, 5 and 10 CPU seconds. In these scenarios we compare using MO-ParamILS directly with using SO-FocusedILS independently on each running time values with proportional configuration budget; that is, as the configuration budget given to a single MO-ParamILS run is one day, the configuration budget given to a single run of ParamILS for a k CPU second cutoff is $^{k}/_{(1+2+3+5+10)} \times 24$ hours. Solution quality for these scenarios is the MIP gap. In the event of CPLEX not returning a MIP gap value within allocated time, the solution quality we set to 10^{10}.Finally, we considered two scenarios for each problem, one using directly the value of the maximum running time parameter value as objective, and the other using the real running time.

For the last scenario, we study the trade-off between memory usage and running time of the SAT solver CLASP on the QUEENS dataset. In this scenario we compare using MO-ParamILS directly with using SO-FocusedILS separately on each of the two objectives for 12 hours. In the event of CLASP not returning any solution within the allocated 300 CPU seconds, we use the PAR10 performance metric (Hutter et al., 2009) to penalise failed runs, i.e., apparent running time of failed runs is set to 10 times the cutting time (3000 CPU seconds).

In all scenarios, penalising configurations that lead to failure of the target algorithm with extremely bad performance values ensures that they are quickly discarded in favour of better performing configurations.

As for the configuration protocol, in the training step each approach is run 25 times with a configuration budget of one day each, using 25 different permutations of the training set and resulting in 25 archives (possibly reduced to a single solution) of configurations. For the CPLEX scenarios we filtered out configurations that resulted in some timeouts. In the validation step, we use a single subset of 100 training instances. In the test step, we use a single subset of 1000 test instances.

Table 5.3 – Hypervolume (top) and ε indicator values (bottom) for final test fronts.

Approach	RCut	RRun	CCut	CRun	Queens
MO-FocusedILS	9.02e-03	**2.07e-03**	**2.37e-02**	**7.63e-04**	**1.57e-02**
MO-BasicILS	**2.46e-03**	5.41e-02	5.53e-02	1.02e-01	5.49e-02
SO Approach	3.82e-02	5.82e-02	3.35e-01	1.72e-01	3.04e-02
Default	2.43e-01	3.57e-01	2.70e-01	5.30e-01	1.08e+00
MO-FocusedILS	**1.44e-02**	**9.05e-03**	**9.00e-02**	**8.06e-04**	**2.64e-02**
MO-BasicILS	1.80e-02	1.71e-01	1.11e-01	1.48e-01	8.35e-02
SO Approach	5.77e-02	1.38e-02	3.33e-01	1.42e-01	6.52e-02
Default	2.22e-01	2.69e-01	2.33e-01	3.90e-01	1.00e+00

5.5.2 Results

After the test step, final fronts have been compared using the hypervolume and ε indicators, after normalisation of every objective in the interval $[1, 2]$. For each scenario, the reference front have been computed by merging every front and filtering dominated points using Pareto dominance. Normalisation and indicator computation have been carried out using the PISA framework (Knowles et al., 2006). Table 5.3 shows the results of this performance assessment for both indicators, the best value for each scenario being highlighted.

Clearly, MO-FocusedILS finds considerably better Pareto fronts for the test sets of all our multi-objective configuration scenarios than the baseline single-objective approach in terms of hypervolume and ε indicator. In all but one case, MO-FocusedILS also produces better results than MO-BasicILS, which, in most cases, still produces better results than the single-objective approach, but with less of a margin.

Empirical results for each scenario, after both validation steps and test steps, are shown in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4, and Figure 5.5, respectively. These results are shown considering only instances solved before the given timeout, the corresponding number of unsuccessful runs being given in Table 5.4.

Figure 5.1 and Figure 5.2 show the fronts of configurations after the validation and test steps on the Regions200 dataset. Time is showed using a logarithmic scale. First, we note that because of the anytime nature of CPLEX, even the baseline, that uses the default configuration without training, resulted in a trade-off curve; however, it achieved much worse results. On the cutoff scenario even if the MO-FocusedILS seems slightly better after the validation step, all three configuration approaches achieve similar performance on unseen instances. In the other hand,

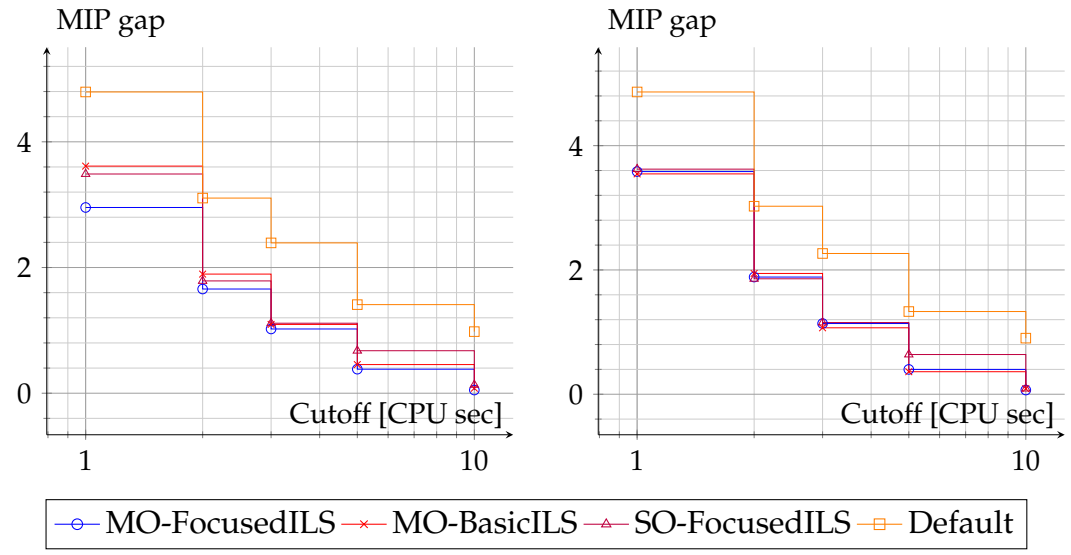


Figure 5.1 – Final fronts on the Regions200 – CPLEX (cutoff) scenario (left: validation; right: test)

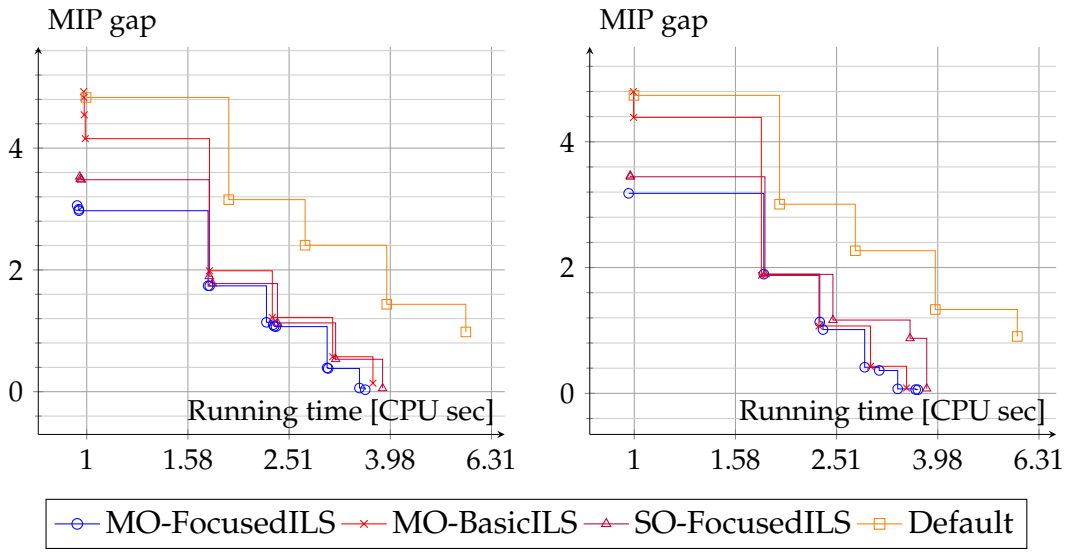


Figure 5.2 – Final fronts on the Regions200 – CPLEX (running time) scenario (left: validation; right: test)

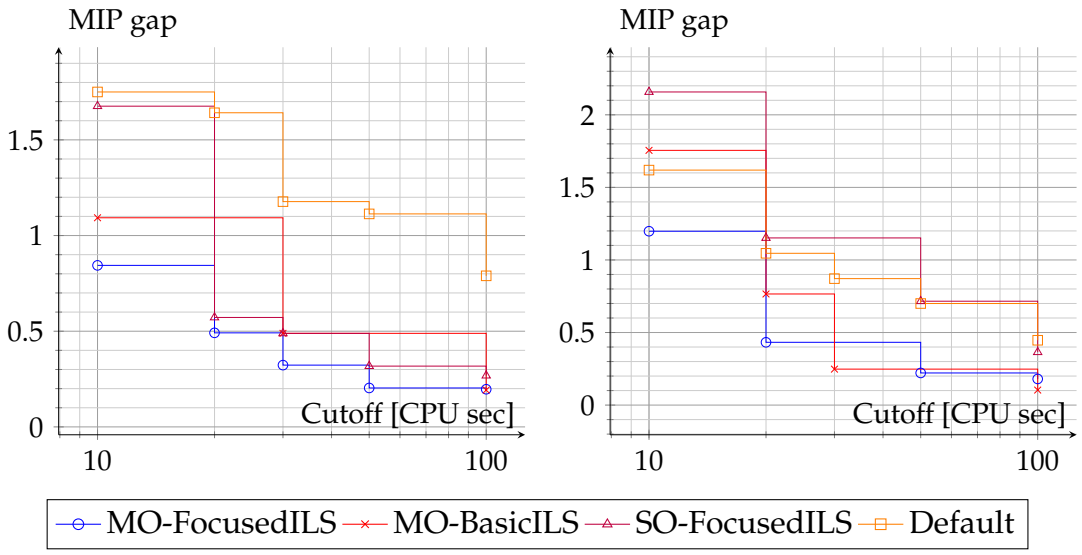


Figure 5.3 – Final fronts on the CORLAT – CPLEX (cutoff) scenario (left: validation; right: test)

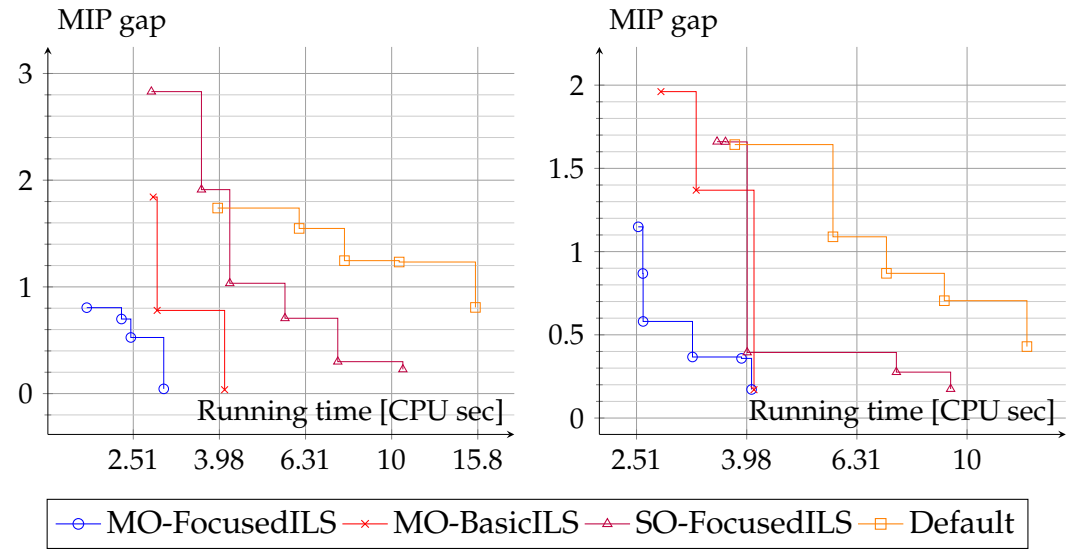


Figure 5.4 – Final fronts on the CORLAT – CPLEX (running time) scenario (left: validation; right: test)

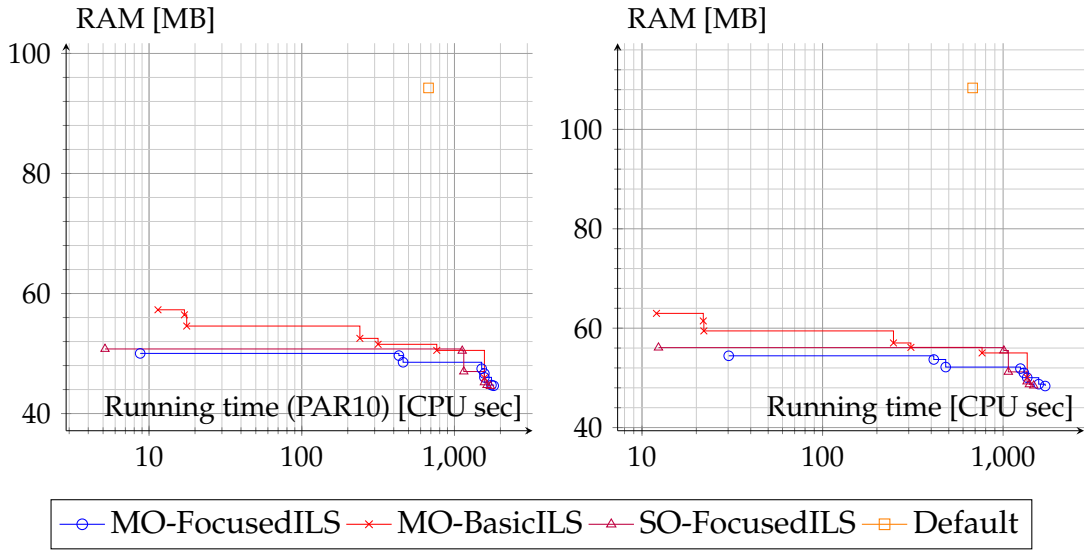


Figure 5.5 – Final fronts on the QUEENS – CLASP scenario (left: validation; right: test)

Table 5.4 – Average percentages of timeouts for final CPLEX configurations

Approach	Validation				Test			
	RCut	RRun	CCut	CRun	RCut	RRun	CCut	CRun
MO-FocusedILS	1.3	0.7	4.2	3.6	0	0	1.06	2.89
MO-BasicILS	0.1	0.6	3.6	2.9	0.04	0	0.47	3.78
SO Approach	0.3	0.4	4.8	5.1	0.12	0	1.87	1.87
Default	0	0	2.2	2.2	0	0	0.14	0.14

on the running time scenario, MO-FocusedILS achieve slightly better results after both steps, while MO-BasicILS is worse for the shortest running times and SO-FocusedILS is worse on the longer ones. For both scenarios, performance after the validation step is very similar to the performance after the test step.

On the CORLAT scenarios (Figure 5.3 and Figure 5.4) this is no more the case, as some approaches lead to results worse than the default configuration after generalisation on unseen instances. For the cutoff variant MO-FocusedILS leads to the best performance, while MO-BasicILS fails for the shortest cutoff, and SO-FocusedILS does not improve the default configuration. Note that MO-BasicILS finds the best configuration for 5 CPU seconds, while MO-FocusedILS has none; however, the configuration found by MO-Focused ILS for 2 CPU seconds could be (and probably should have been) used with the 5 CPU seconds cutoff to complete the front. For the running time variant, MO-FocusedILS clearly outperforms the other approaches.

On the QUEENS scenario (Figure 5.5), the default configuration of CLASP resulted as expected in a single point. All three other approaches successfully founded configuration with much better memory consumption and diverse running time. Overall, MO-FocusedILS achieved the best results.

When analysing these results, we also noticed that MO-FocusedILS evaluates many more unique configurations than MO-BasicILS (4752 *vs* 166 on average, over all five scenarios). This clearly indicates the efficacy of the way in which MO-FocusedILS controls the number of runs per configuration performed and mirrors analogous findings for BasicILS *vs* FocusedILS in the single-objective case (Hutter et al., 2009).

On all five scenarios, the default configurations of CPLEX and CLASP produced few unsuccessful runs on training or test instances. The three other approaches lead to configurations generating about as many timeouts as the default configuration. However, by also taking in account the configurations returned that have both more timeouts and better performance on successful instances, we were able to achieve even better results at the cost of a small loss of generality, as shown in Table 5.3. While our CLASP scenario uses PAR10 scores to take into account instances that could not be solved within the given cutoff time, as previously mentioned, the final Pareto fronts we produce for the CPLEX scenarios do not reflect a small number of instances for which no MIP gap was obtained within the allocated running time. The fraction of the validation and test sets on which this happened is shown in Table 5.4; as seen there, timeouts generally occur for a small fraction of instances, and while that fraction tends to increase as we configure CPLEX, it remains low enough in all cases to not raise serious concerns.

5.6 Perspectives

In this chapter, we presented ParamILS, a prominent algorithm configurator, then we introduced MO-ParamILS, a multi-objective extension enabling to consider multiple performance criteria simultaneously, while also proposing a configuration protocol for using standard single-objective configurators on multi-objective scenarios. We also validated our the performance of our framework by comparing the multiple variants of MO-ParamILS to the best variant of ParamILS on various configuration scenarios.

We detail in the following two perspectives related to MO-ParamILS and multi-objective automatic algorithm configurators.

Other multi-objective configurators. First of all, to propose our multi-objective algorithm configurator, we choose to extend the existing ParamILS configurator, based on its use of local search techniques. It is clear that other well-known configurators from the literature, such as for example irace (López-Ibáñez et al., 2016) or SMAC (Hutter et al., 2011), could also be similarly extended to bring both competition and other insights on automatic multi-objective algorithm configuration. Furthermore, while there have already been preliminary works on multi-objective racing (Zhang et al., 2015, 2016, 2018), users could really benefit by having more available ready- and easy-to-use configurators.

Configuration protocol. We proposed multiple configuration protocol and approaches to tackle multi-objective configuration scenarios, for both single-objective and multi-objective configurators. However, these are still primarily focused on ParamILS, and should be further analysed and discussed in general for all configurators. For example, we advocate parallelising the training step, running the simultaneously configurator multiple times on multiple subsets of the training set of instances, although other configurators (e.g., irace) advocate to run the configurator only once. While there may definitely are strong opinions, to our present knowledge there are no consequent results on, within other open questions, how to fairly compare the multiple configuration protocols associated to each configurator, how many times should be run the configurator, how the training instances should be selected and how exactly their distribution impact the configurator performance.

Chapter 6

MOLS Configuration

> Knowing yourself is the beginning of all wisdom.
>
> Aristotle

In this chapter, we conduct three successive studies on the static multi-objective local search (MOLS) algorithm presented in Chapter 4.

First, we investigate the configuration space of our static MOLS algorithm, to see to which extent using different configurations can lead to different results, and to better understand the impact of the different parameters and their relations between each others. Therefore, we consider a reduced number of parameters and we conduct an exhaustive analyse of all the resulting possible configurations of two permutation problems presented in Chapter 1: the flowshop scheduling problems (PFSP) and the travelling salesman problem (TSP).

Then, we investigate the automatic configuration of our static MOLS algorithm, by comparing the performance of three configurations approaches: two single-objective (SO-AAC) approaches based on a single performance indicator and on an aggregation of two performance indicators, and a multi-objective (MO-AAC) approach using a Pareto trade-off between the two performance indicators. This second study builds on the first study, by first considering the reduced configuration space used for the exhaustive analysis, before considering a much larger configuration space. Indeed, while the reduced configurations space enables exhaustive evaluation, global visualisation and general discussions, the larger configurations space is more representative of algorithms for which exhaustive analysis would be prohibitive.

Finally, we study the impact of objectives correlation of the multi-objective problem itself on the performance of the three automatic configuration approaches.

Therefore, we conduct this study using instances for which the correlation between objectives is controlled. In this final study, in addition to the PFSP and the TSP, we also use another classical permutation problem, the quadratic assignment problem (QAP), also presented in Chapter 1.

This chapter contributions are closely linked to the following publications:

- **Blot, A.**, Jourdan, L., and Kessaci-Marmion, M. (2017a). **Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem.** In Bosman, P. A. N., editor (2017). *Genetic and Evolutionary Computation Conference, GECCO 2017. Proceedings*, pages 227–234. ACM.
- **Blot, A.**, Pernet, A., Jourdan, L., Kessaci-Marmion, M., and Hoos, H. H. (2017c). **Automatically configuring multi-objective local search using multi-objective optimisation.** In Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M. M., Jin, Y., and Grimme, C., editors (2017). *Evolutionary Multi-Criterion Optimization – 9th International Conference, EMO 2017. Proceedings*, volume 10173 of *Lecture Notes in Computer Science*, pages 61–76. Springer.
- **Blot, A.**, Hoos, H. H., Kessaci, M., and Jourdan, L. (2018a). **Automatic configuration of multi-objective optimization algorithms. impact of correlation between objectives.** In *30th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2018.* IEEE Computer Society. (To appear).

Additionally, the following paper has been submitted in the special issue on algorithm selection and configuration of the *Evolutionary Computation* journal:

- **Blot, A.**, Kessaci-Marmion, M., Jourdan, L., and Hoos H. H.. **Automatic Configuration of Multi-Objective Local Search Algorithm for Permutation Problems.**

6.1 Exhaustive Analysis

In the first study, we focus on analysing the configuration space of our static MOLS algorithm. By considering a reduced configuration space of the MOLS algorithm, we are able to conduct an exhaustive analysis on various PFSP and TSP scenarios, and to draw general conclusions over MOLS parameters.

6.1.1 Experimental Protocol

In this study, we ensure that the size of the configuration scenario is small enough so that an exhaustive assessment of all possible configurations of the target al-

Table 6.1 – Small version of the MOLS configuration space (300 configurations)

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `oldest`}
Selection	`select-size`	{1, 10}
Exploration	`explor-strat`	{`imp`, `imp-ndom`, `ndom`}
Exploration	`explor-ref`	{`sol`, `arch`}
Exploration	`explor-size`	{1, 10}
Archive	`bound-strat`	{`rand`}
Archive	`bound-size`	{1000}
Perturbation	`perturb-strat`	{`kick`, `kick-all`, `restart`}
Perturbation	`perturb-size`	{10}
Perturbation	`perturb-strength`	{3, 10}

Selection: $(1 + 2 \times 2)$ combinations ; Exploration: $(3 \times 2 \times 2)$; Perturbation: $(2 \times 2 + 1)$; Total: $5 \times 12 \times 5 = 300$ configurations

gorithm is feasible. This has one direct consequence: we will be able to analyse the entire induced search space and the optimal configurations, which would be otherwise unfeasible. More specifically, we restrict the configuration space of our static MOLS algorithm according to Table 6.1. The full configuration space of the MOLS algorithm have been presented in Chapter 4 (see Table 4.1). In terms of parameter values, we removed the `newest` selection strategy, the `all` and `all-imp` exploration strategies, and the `replace` archive strategy; we also restricted numerical parameters to only one or two parameter values. These restrictions have been based on preliminary experiments with the two goals of first producing known efficient combinations of parameters and second keeping a very small total number of configurations (here, only 300). Note that the chosen `bound-size` parameter value, 1000, effectively disables the bounding mechanism as archives never achieve such a large size.

We consider six distinct scenarios: three PFSP scenarios with instances with 50, 100, and 200 jobs, all with 20 machines; and three TSP scenarios with 100, 300, and 500 cities. For each of the PFSP scenarios, we considered the corresponding existing 10 Taillard instances for the test set, while we generated 30 new instances for the training set using Taillard's generation procedure. For each of the TSP scenarios, we considered the 15 pairwise independent combinations of the existing 6 Paquete instances for the test set and generated 30 new pairs of instances for the

training set, obtained using the original DIMACS generator. Further details regarding these instances are provided in Chapter 1.

The running time of our static MOLS algorithm depends of the scenario size. On PFSP instances, it was set to $n^2 \cdot m / 1000$ CPU seconds, with n the number of jobs and $m = 20$ the number of machines (i.e., 50 CPU seconds, 3 minutes and 20 CPU seconds, and 13 minutes and 20 CPU seconds, respectively to instances with 50, 100, and 200 jobs, all with 20 machines). On TSP instances, it was set to $n \cdot 0.9$ CPU seconds, with n the number of cities (i.e., 1 minute and 30 CPU seconds, 4 minutes and 30 CPU seconds, and 7 minutes and 30 CPU seconds, respectively to instances with 100, 300, and 500 cities).

Additionally, on PFSP instances, the search was initialised using the 2-phase local search algorithm (Dubois-Lacoste et al., 2011b), which is based on the iterated greedy (IG) procedure (Ruiz and Stützle, 2007). This method is known to produce relatively good and well-distributed solutions sets in the objective space. We use 25% of the overall time budget for this initialisation, and 75% for the remainder of each MOLS run. On TSP instances, as no such initialisation procedure is known to produce quick and well-distributed solutions sets on the Paquete instances, the search is initialised using two independent solutions, obtained using a greedy procedure on each of the two distance matrices taken individually, to avoid starting only from solutions taken uniformly at random from the search space.

To compare each of the 300 configurations, we assessed each of them using a single run on each of the instance of the training set and 10 runs on each of the instance of the test set, averaging independently the hypervolume and Δ' spread values. These levels of detail of the approximations have been taken so they are compatible with the experimental protocol of the second study.

As introduced in Chapter 1, the hypervolume and the Δ' spread are two multi-objective performance indicators, capturing information about the accuracy and the distribution of Pareto sets of solutions. We recall that we transform the hypervolume (HV) into a minimisation measure (1−HV) to simplify the analysis of our results, and thus, when speaking of good hypervolume values, we refer to high HV (i.e., low values of 1−HV).

The experiments, for all three studies of this chapter, have been conducted on the *grace* cluster of the ADA research group at the Leiden Institute of Advance Computer Science (LIACS), in the Netherlands. Each of the 32 nodes of *grace* is equipped with two 16-core 2.10GHz Intel Xeon E5-2683 v4 CPUs with 40MB L3 cache and 94GB RAM, running CentOS 7.4.1708. Computations were conducted in parallel as much as possible.

6.1.2 Parameter Distribution Analysis

Figure 6.1 shows the parameter distribution of the 300 configurations on test instances of the PFSP and the TSP scenarios, highlighting the parameter values of the two parameters: `select-strat`, with crosses ($+ \times \star$), polygons ($\square\Delta\diamond$), and circles ($\circ\oplus\otimes$); and `explor-strat`, with red ($+\square\circ$), green ($\times\Delta\oplus$), and blue ($\star\diamond\otimes$) colours.

PFSP. (Figure 6.1, left) None among the 300 possible configurations simultaneously achieves good hypervolume and spread values. The Pareto front is distinctly non-convex. While for the smallest scenario, with 50 jobs, most of all configurations achieve good hypervolume values (i.e., low 1−HV), such configurations get rarer as the number of jobs increases. This result was expected, since it is known that larger PFSP instance are harder for MOLS algorithms. Examining these results in more detail, we observe that the `imp` exploration strategy always obtains rather bad hypervolume values. For 50 jobs, this strategy leads to better spread values; however, it tends to be no longer true for larger instances. For the three instance sizes, the `imp-ndom` and `ndom` strategies appear to give better performance in terms of hypervolume.

TSP. (Figure 6.1, right) The results on the TSP markedly differ from those on the PFSP. Firstly, we observe that the shape of the Pareto-optimal front of configurations varies with instance size: while it is convex for 100 cities with some degree of correlation between hypervolume and spread, for larger instances, the correlation between the two performance indicators decreases, and the front becomes non-convex. In contrast to the PFSP, where the two objectives are correlated, for our TSP benchmark sets the objectives are completely independent. Therefore, the final archives are much bigger, as there exist a larger space of trade-off solutions. The impact on spread is evident: values above 1 correspond to two tightly clustered sets of solutions separated by a large gap that the respective configuration of MOLS failed to cover, and spread values of 0 correspond to final sets containing only two solutions, which are produced when the `imp` exploration strategy fails to sufficiently diversify.

6.1.3 Optimal Configurations

Table 6.2 and Table 6.3 list the Pareto-optimal configurations within the exhaustively enumerated configuration space for both PFSP and TSP scenarios. A "$*$" symbol indicates that the value of the respective parameter does not impact the

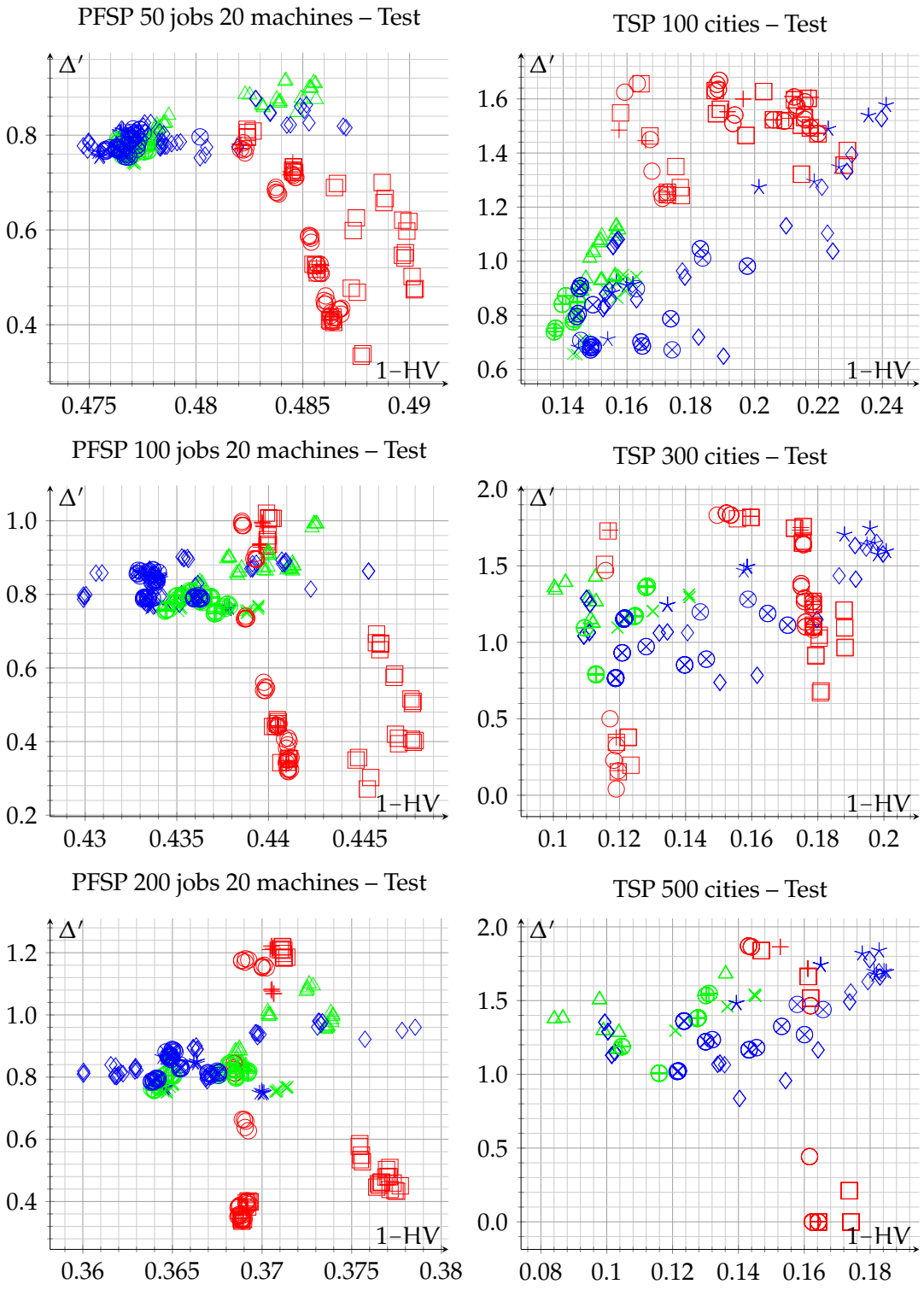


Figure 6.1 – Exhaustive analysis parameter distribution on test instances (left: PFSP; right: TSP); Selection strategy: +×⋆: `all` (crosses), □Δ◇: `oldest` (polygons), ○⊕⊗: `rand` (circles); Exploration strategy: +□○: `imp` (red), ×Δ⊕: `imp-ndom` (green), ⋆◇⊗: `ndom` (blue)

performance of the configured MOLS when the other parameter values are held fixed at the values shown. Conversely, when a specific parameter is shown, any deviation from it will reduce performance.

PFSP. (Table 6.2) Regarding the nature of the configurations, we observe a trend across the three instance sizes. The best hypervolume is always reached with the `oldest` selection strategy, the `ndom` exploration strategy and the `arch` exploration reference set choice. Slightly worse hypervolume, but better spread is achieved using the `imp-ndom` exploration strategy. Finally, the best spread values are obtained from configurations using the `imp` exploration strategy, although this comes at the cost of rather bad hypervolume. In almost every case, the perturbation strategy did not significantly impact the performance of the non-dominated configurations.

TSP. (Table 6.3) MOLS configurations achieving the best hypervolume values always use the `imp-ndom` exploration strategy with the `sol` reference set. While for 300- and 500-city instances, the `oldest` selection strategy is preferred, for 100 cities, the more common `rand` selection strategy performs better. Similarly to the PFSP, the choice of perturbation mechanism does not significantly impact the performance of optimal configurations.

6.1.4 Discussions

The exhaustive analysis of a small subset of the configuration space validates the worth of automatically configuring our static MOLS algorithm.

First, as shown in Figure 6.1, the possible configurations are both very distinct and well clustered in the objective space, confirming first that it is useful to search for the best possible configuration, and second that the parameter values have specific, non-random, impact on the performance of our MOLS algorithm. Interestingly, if for PFSP instances the three configuration spaces are very similar, hinting that observations on small instances could be generalised on larger instances, this is not true for TSP instances for which the three configuration spaces are very unalike.

General observations on MOLS strategies can nevertheless be made: the choice of the perturbation strategy is clearly the less important, as it mostly does not impact much the performance of the optimal configurations. Furthermore, save for the smallest TSP dataset, the `imp` exploration strategy generally lead to poorer convergence but better distribution of the solutions.

In conclusion, the analysis of the optimal configuration given by Table 6.2 and

Table 6.2 – PFSP *(optimal configurations)*

1−HV	Δ'	Selection		Exploration			Perturbation		
				(PFSP 50 jobs 20 machines)					
0.4747	0.7775	`oldest`	10	`ndom`	`arch`	1	*	10	*
0.4754	0.7640	`all`		`ndom`	`arch`	1	*	10	*
0.4770	0.7420	`all`		`imp-ndom`	`sol`	10	*	10	*
0.4837	0.6798	`rand`	1	`imp`	`arch`	10	*	10	*
0.4853	0.5856	`rand`	1	`imp`	`sol`	10	*	10	*
0.4855	0.5277	*	10	`imp`	`arch`	1	*	10	*
0.4860	0.4433	`rand`	1	`imp`	`arch`	1	*	10	*
0.4862	0.4093	*		`imp`	`sol`	1	*	10	*
0.4877	0.3336	`oldest`	1	`imp`	`sol`	1	`kick`	*	10
				(PFSP 100 jobs 20 machines)					
0.4299	0.7865	`oldest`	10	`ndom`	`arch`	1	`kick`	10	3
0.4299	0.7979	`oldest`	10	`ndom`	`arch`	1	`kick-all`		*
0.4332	0.7802	`oldest`	1	`ndom`	`arch`	1	`kick`	10	*
0.4336	0.7640	`all`		`ndom`	`arch`	1	*	10	*
0.4344	0.7541	`rand`	10	`imp-ndom`	`arch`	1	*	10	*
0.4351	0.7540	`all`		`imp-ndom`	`sol`	1	*	10	*
0.4370	0.7470	`rand`	10	`imp-ndom`	`arch`	10	*	10	*
0.4387	0.7338	`rand`	1	`imp`	`arch`	10	*	10	*
0.4397	0.5396	`rand`	1	`imp`	`sol`	10	*	10	*
0.4402	0.4409	*	10	`imp`	`arch`	1	*	10	*
0.4407	0.3428	`oldest`	10	`imp`	`sol`	1	*	10	*
0.4410	0.3201	`rand`	1	`imp`	`sol`	1	*	10	*
0.4410	0.3371	`all`		`imp`	`sol`	1	*	10	*
0.4454	0.2711	`oldest`	1	`imp`	`sol`	1	`kick`	10	*
				(PFSP 200 jobs 20 machines)					
0.3600	0.8093	`oldest`	1	`ndom`	`arch`	1	`restart`	10	*
0.3600	0.8093	`oldest`	1	`ndom`	`arch`	1	`kick`	10	*
0.3618	0.8027	`oldest`	10	`ndom`	`arch`	1	*	10	*
0.3638	0.7628	`rand`	1	`imp-ndom`	`arch`	1	*	10	*
0.3645	0.7534	`all`		`imp-ndom`	`arch`	1	*	10	*
0.3686	0.3511	`rand`	1	`imp`	`sol`	1	*	10	*
0.3687	0.3456	*	10	`imp`	`sol`	1	*	10	*

Table 6.3 – TSP *(optimal configurations)*

1−HV	Δ'	Selection		Exploration			Perturbation	
				(TSP 100 cities)				
0.1372	0.7389	`rand`	10	`imp-ndom`	`sol`	10	*	10 *
0.1431	0.6572	`all`		`imp-ndom`	`sol`	10	`restart`	
0.1443	0.6544	`all`		`imp-ndom`	`arch`	10	`restart`	
0.1902	0.6488	`oldest`	1	`ndom`	`sol`	1	`kick`	10 3
				(TSP 300 cities)				
0.1003	1.3582	`oldest`	10	`imp-ndom`	`sol`	1	*	10 *
0.1006	1.3417	`oldest`	10	`imp-ndom`	`sol`	10	*	10 *
0.1092	1.0409	`oldest`	10	`ndom`	`sol`	10	*	10 *
0.1128	0.7933	`rand`	10	`imp-ndom`	`arch`	1	*	10 *
0.1129	0.7880	`rand`	1	`imp-ndom`	`arch`	1	*	10 *
0.1171	0.5003	`rand`	1	`imp`	`sol`	10	`restart`	
0.1183	0.2288	`rand`	1	`imp`	`sol`	1	`restart`	
0.1190	0.0409	`rand`	1	`imp`	`arch`	1	`restart`	
				(TSP 500 cities)				
0.0841	1.3767	`oldest`	10	`imp-ndom`	`sol`	1	*	10 *
0.0989	1.2983	`oldest`	1	`imp-ndom`	`arch`	10	*	10 *
0.1003	1.2897	`oldest`	10	`ndom`	`arch`	10	*	10 *
0.1015	1.1290	`oldest`	10	`ndom`	`sol`	10	*	10 *
0.1159	1.0080	`rand`	*	`imp-ndom`	`arch`	1	*	10 *
0.1403	0.8468	`oldest`	10	`ndom`	`arch`	1	`kick`	10 *
0.1616	0.4420	`rand`	1	`imp`	`sol`	10	*	10 *
0.1624	0.0000	`rand`	1	`imp`	*	1	*	10 *

Table 6.3 confirm that there is no *default* configuration of the MOLS algorithm that would be optimal on all PFSP and TSP datasets. Indeed, optimal configurations are highly dependant of the both the problem tackled and the size of the instances. It also confirms that the MOLS configuration space is well structured and adapted to AAC.

6.2 AAC Approaches Analysis

In the second study, we focus on analysing the performance of automatic algorithm configuration (AAC) approaches on out static MOLS algorithm. We investigate three different AAC approaches, using diverse PFSP and TSP scenarios of multiple size. This study enables to validate efficient approaches to design MOLS algorithms.

6.2.1 Experimental Protocol

In this second study, as well as in the third study, we consider three AAC approaches: two single-objective AAC (SO-AAC) approaches and one multi-objective AAC (MO-AAC) approach, using ParamILS and MO-ParamILS, respectively. We recall the distinction made in Chapter 2: SO-AAC deals with the optimisation of a single scalar performance indicator, while MO-AAC simultaneously deals with the optimisation of a vector of performance indicators.

More specifically, we will compare:

HV, a SO-AAC approach that optimises the hypervolume indicator only;
HV+Δ′, a SO-AAC approach that optimises a weighted sum of hypervolume (with a 0.75 coefficient) and Δ' spread (with a 0.25 coefficient); and
HV||Δ′, a MO-AAC approach that simultaneously considers hypervolume and Δ' spread.

The latter two approaches are motivated by the previously mentioned belief that the performance assessment of multi-objective algorithms benefits from the use of multiple performance indicators (Zitzler et al., 2003). By comparing HV to the two other configuration approaches, we aim to assess this belief in the context of automatic configuration of MOLS algorithms. Furthermore, by comparing HV+Δ' and HV||Δ', we intend to assess the benefits of MO-AAC compared to SO-AAC with aggregated performance metrics. The aggregation coefficient, 0.75, results from the Δ' indicator being seen as a complementary measure to the hypervolume, in order to focus on convergence first and diversity second.

Table 6.4 – Large version of the MOLS configuration space (10 920 configurations)

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`select-size`	{1, 3, 10}
Exploration	`explor-strat`	{`all`, `all-imp`, `imp`, `imp-ndom`, `ndom`}
Exploration	`explor-ref`	{`sol`, `arch`}
Exploration	`explor-size`	{1, 3, 10}
Archive	`bound-strat`	{`rand`}
Archive	`bound-size`	{20, 50, 100, 1000}
Perturbation	`perturb-strat`	{`kick`, `kick-all`, `restart`}
Perturbation	`perturb-size`	{1, 5, 10}
Perturbation	`perturb-strength`	{3, 5, 10}

Selection: $(1 + 3 \times 3)$ combinations ; Exploration: $(1 + 2 + 3 \times 2 \times 3)$;
Perturbation: $(3 \times 3 + 3 + 1)$; Total: $10 \times 21 \times 13 \times 4 = 10\,920$ configurations

Note that all three AAC approaches are nevertheless used in a MO-AAC protocol, detailed hereafter, in which the *training* step is either performed in a single- or multi-objective way, while both *validation* and *test* steps are performed in a multi-objective, Pareto, way.

The three approaches use the FocusedILS variants of both ParamILS and MO-ParamILS configurators, since these usually give the best performance; regarding the HV and HV+Δ' approaches, they both follow the recommendation of using adaptive and aggressive capping (Hutter et al., 2009). Details on both configurators and their respective variants are given in Chapter 5.

We analyse the performance of all three approaches on two different configuration spaces. First, we use the small space of 300 configurations described in the first study of this chapter (see Table 6.1), to analyse the performance of the three approaches in an exhaustively enumerated context. Then, we consider a much richer space of 10 920 configurations, detailed in Table 6.4, on which the previous study would require a computational budget several orders of magnitude higher due to the relatively high running times for each configuration and the stochastic nature of the target algorithm.

We use the AAC protocol presented in Chapter 5, whose specific details are summarised in Table 6.5. The main protocol differences for the two configuration

Table 6.5 – AAC Experimental Protocol

Step	Small configuration space	Large configuration space
Training	No default configuration	No default configuration
	1 random configuration	10 random configurations
	10 ParamILS runs	20 ParamILS runs
	100 MOLS runs budget	1000 MOLS runs budget
	max 10 MOLS run per config.	max 100 MOLS run per config.
Validation	1 run per instance	1 run per instance
Test	10 runs per instance	10 runs per instance

spaces concern the training step. For the small (large) configuration space, ParamILS starts by evaluating a single (10) random configuration, and can execute 100 (1000) MOLS runs before stopping, where each selected configuration cannot be run more than 10 (100) times. Due to the reduced size of the small configuration space, only 10 independent runs of ParamILS are performed, compared to 20 runs for the large space. In the validation step, the configurations resulting from the training step are evaluated on all training instances, running every configuration once on each instance. In the test step, each of the configurations in the Pareto set obtained from the validation step is run 10 times on every test instance. For both validation and test steps, the performance of each configuration is assessed based on the average hypervolume and Δ' spread values over the runs. Obviously, for the small configuration space, our exhaustive analysis ensures that the performance of all configurations are known for all training and test instances, and we directly use these results in the validation and test steps to avoid recomputing the performance of configurations selected in the training step.

Lastly, Table 6.6 reports the bounds used for each scenario to compute the aggregation in the case of the HV+Δ' approach. These bounds have been determined using preliminary data from the exhaustive analysis on the training sets of instances.

6.2.2 Small Configuration Space Results

Figure 6.2 and Figure 6.3 show the results of the configuration process using the small configuration space on PFSP and TSP scenarios, respectively. The configurations in consideration by the three approaches are shown after the validation step on training instances, and after the test step on test instances. Every configuration

Table 6.6 – Indicator bounds used in the HV+Δ′ approach

Scenario	(1−HV) lower	(1−HV) upper	Δ lower	Δ upper
PFSP 50	0.48	0.5	0.2	1
PFSP 100	0.44	0.46	0.1	1.1
PFSP 100	0.355	0.375	0.3	1.3
TSP 100	0.13	0.24	0.6	1.7
TSP 300	0.09	0.2	0	2
TSP 500	0.08	0.18	0	2

not in consideration by any of the three approaches is also indicated, as already exhaustively enumerated during the first study.

PFSP. (Figure 6.2) All three approaches find very good, even near-optimal configurations – in particular, HV||Δ′, which results in configurations spreading over the entire Pareto-front. The 10 configurator runs of HV and HV+Δ′ produce close to 10 unique configurations each and all of these show good hypervolume values. However, after validation and testing, for both AAC approaches few configurations remain and those tend to have good hypervolume but average spread. On the other hand, the MO-AAC approach HV||Δ′ produces many more configurations after the training, validation and test steps. Compared to the two other approaches, HV||Δ′ clearly achieves better coverage of the optimal Pareto set of configurations. Note that all three approaches use the same time budget for configuration, the number of final solutions being strongly dependant of the kind (single-objective or multi-objective) of AAC used for training.

TSP. (Figure 6.3) The HV configuration approach produces few configurations, achieving near-optimal hypervolume. HV+Δ′ produces weak training results on the 100-city instances, but works well on the 300-city instances, because of the shape of the Pareto-optimal front. As for the PFSP, HV||Δ′ finds many more configurations and achieves far better coverage of the Pareto front. In the test instances from the literature, all three AAC approaches produces optimal configurations for 100-city instances, HV+Δ′ and HV||Δ′ also do on 300-city instances, but only HV||Δ′ manages to find most of the optimal configurations on the 500-city instance.

For both problems, within the small configuration space, all three AAC approaches are able to find configurations very close to the true Pareto-front. As ex-

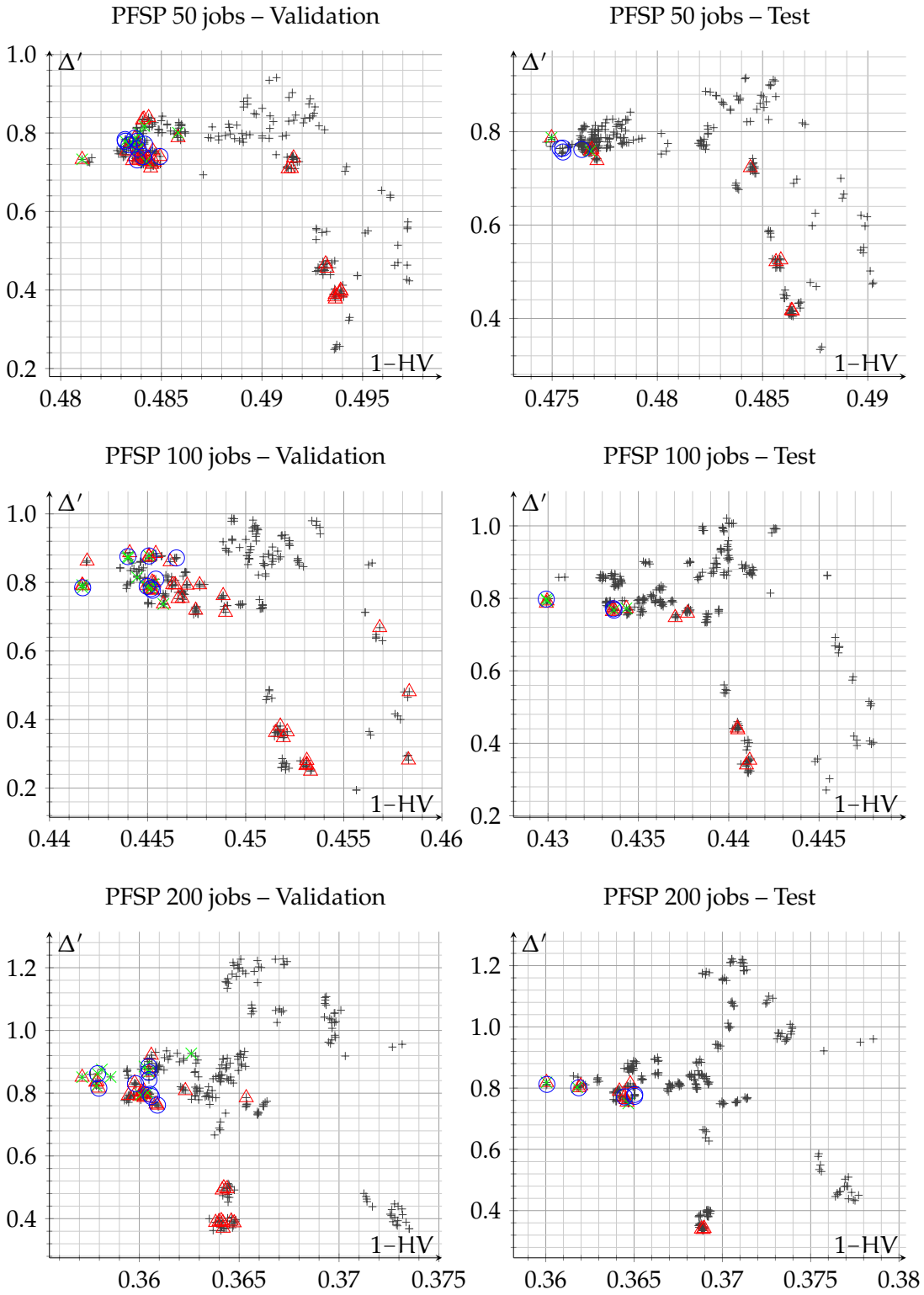


Figure 6.2 – Experiments on the small configuration space – PFSP scenarios
x: HV approach, o: HV+Δ′ approach, Δ: HV||Δ′ approach, +: exhaustive analysis

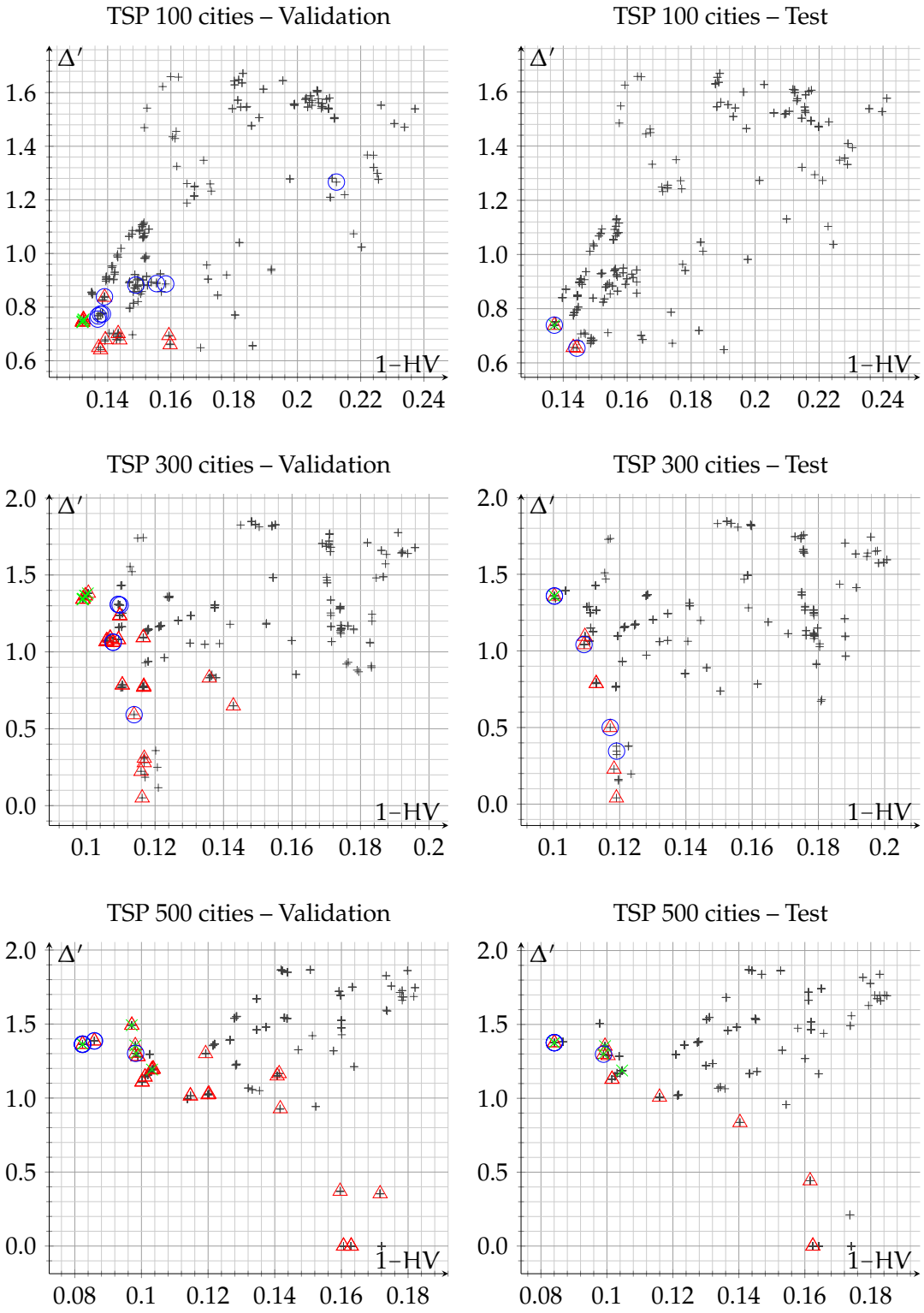


Figure 6.3 – Experiments on the small configuration space – TSP scenarios
x: HV approach, o: HV+Δ′ approach, Δ: HV||Δ′ approach, +: exhaustive analysis

pected, the two SO-AAC approaches strongly favours the hypervolume indicator. However, for similar training time, the MO-AAC approach is able to accurately cover the full range of Pareto-optimal configurations.

6.2.3 Large Configuration Space Results

Figure 6.4 shows the final configurations produced by all three AAC approaches for both PFSP and TSP scenarios, considering the larger configuration space of 10920 configurations. In contrary to the results on the small configuration space of 300 configurations only, the configurations in consideration by the three approaches are only shown on the test instances. We also show the 300 configurations of the smaller set of configurations that we previously exhaustively evaluated, in order to highlight that these final configurations map very closely those found within the small space, which suggests that the small space indeed captures the high-performance configurations from the much larger space and, more importantly, demonstrates that our AAC approaches effectively finds such configurations. In the following, we focus on the performance of the three AAC approaches.

SO-AAC. Both SO-AAC approaches, HV and HV+Δ', produce very few non-dominated configurations in their final testing step – typically between 2 and 4 on each instance set. As one might expect, HV always finds a final configuration with near-optimal hypervolume. The results for HV+Δ' are similar to those for HV for the PFSP, but markedly different on the TSP scenarios. For 100-city bTSP instances, HV+Δ' covers the Pareto front, while for 300 cities, it finds the two extreme configurations, due to accidentally well-adapted weights used for aggregating hypervolume and spread. However, due to the non-convex shape of the front, no trade-off configurations are found between these extremes. For 500-city instances, HV+Δ' only finds configurations with near-optimal hypervolume, similar to what we observed for the PFSP.

MO-AAC. In the other hand, the MO-AAC approach, HV||Δ', consistently provides many more non-dominated configurations, except for the small 100-city bTSP instance set, where the Pareto front is completely covered by all three approaches. In all cases, the sets of configurations found by HV||Δ' are very well distributed over the entire front of optimal configurations. Although HV+Δ' sometimes finds better configurations (e.g., on the 100- and 200-jobs PFSP scenarios), HV||Δ' always produces configurations with similar performance.

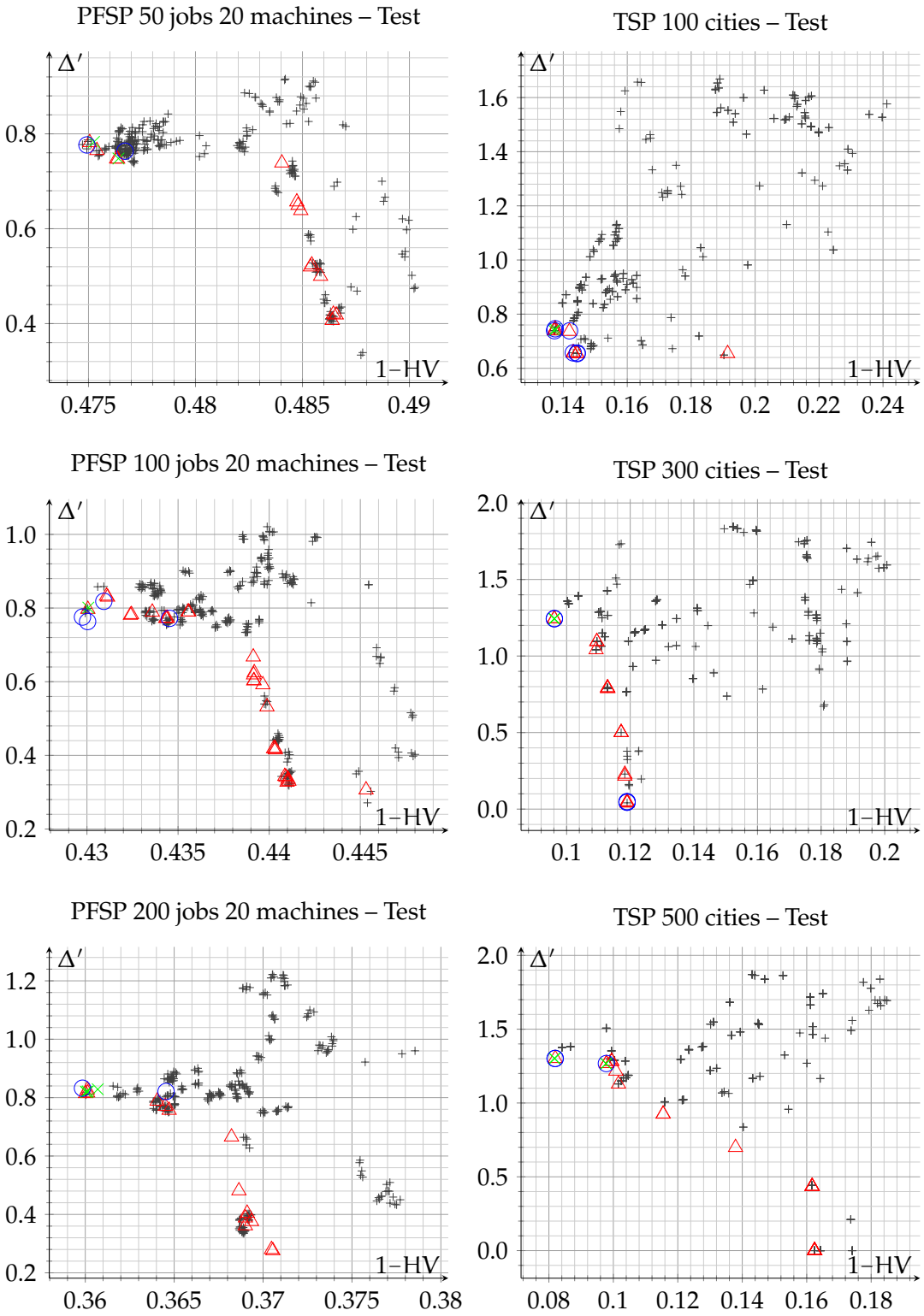


Figure 6.4 – Experiments on the large configuration space
x: HV approach, o: HV+Δ′ approach, Δ: HV||Δ′ approach, +: exhaustive analysis

Table 6.7 – Number of configurations after training, validation and testing

Scenario	Approach	Small space			Large space		
		Configs	Pareto	Final	Configs	Pareto	Final
PFSP 50	HV	10	2	2	20	2	2
	HV+Δ'	10	4	2	10	2	2
	HV$\|\|\Delta'$	32 (38)	9	7	145	14	11
PFSP 100	HV	10	4	3	19 (20)	1	1
	HV+Δ'	8 (10)	3	3	20	4	2
	HV$\|\|\Delta'$	36 (42)	12	6	171 (172)	27	19
PFSP 200	HV	10	3	3	20	4	3
	HV+Δ'	9 (10)	5	3	16 (20)	2	2
	HV$\|\|\Delta'$	29 (39)	11	8	111 (117)	14	9
TSP 100	HV	6 (10)	1	1	15 (20)	2	1
	HV+Δ'	6 (10)	2	2	15 (20)	6	4
	HV$\|\|\Delta'$	16 (26)	3	3	62 (73)	11	5
TSP 300	HV	9 (10)	2	2	13 (20)	4	2
	HV+Δ'	9 (10)	5	5	12 (20)	5	2
	HV$\|\|\Delta'$	33 (41)	8	6	107 (130)	18	12
TSP 500	HV	6 (10)	5	4	16 (20)	4	4
	HV+Δ'	8 (10)	5	4	14 (20)	3	2
	HV$\|\|\Delta'$	36 (40)	12	11	135 (145)	25	22

6.2.4 Discussions

In addition to the quality of the final configurations, the analysis of the sizes of the sets of configurations returned by each AAC approach at each step of the configuration protocol provides more insight on each approach performance.

Table 6.7 details the number of configurations resulting after each of the three steps of the configuration protocol, for each of the three configuration approaches, the six scenarios, and the two configuration spaces. The first column, "*Configs*", gives the number of unique configurations after the training step, the total number being given with parenthesis when a unique configuration results from multiple ParamILS runs. The second column, "*Pareto*", gives the number of configurations that are not dominated on the training instances, after the validation step. Finally, the third column, "*Final*", gives the number of configurations that, non-dominated on the training instances, also are non-dominated on the test instances.

First, the numbers of final training configurations of the SO-AAC approaches

are obviously limited by the number of ParamILS runs, as a maximum of one configuration is returned each run. Fewer configurations results when multiple runs returns the same configuration. On the other hand, each run of the MO-AAC approach returns a set of configuration, resulting on far more training configurations. After the validation many of the training configurations are discarded, while most of the non-dominated configurations on the training instance sets are also non-dominated on the associated testing instance sets, indicating that they generalise well on the unseen instances.

Overall, Figure 6.2, Figure 6.3, and Figure 6.4 show that the MO-AAC approach, HV||Δ', produces substantially better results than the two SO-AAC approaches, HV and HV+Δ'. HV finds excellent sets of configurations with respect to hypervolume, but only provides very few of those and consequently fails to achieve good spread. HV+Δ' sometimes provides better results and, under favourable circumstances, can cover the entire set of Pareto-optimal configurations; however, especially for more challenging scenarios, its performance is similar to that of HV. The main drawback of this approach is the requirement of a costly preliminary step for calibrating the weights used for aggregating the two optimisation objectives. Finally, HV||Δ', the MO-AAC approach, always efficiently covers the entire Pareto-front of configurations, while still finding sets of configurations with excellent hypervolume, as produced by the two SO-AAC approaches.

In conclusion, HV||Δ', the MO-AAC approach should clearly be preferred on all of our datasets. Despite using the same configuration budget (1000 target algorithm runs), it is able to efficiently cover the entire Pareto front of configurations while simultaneously matching the performance of both SO-AAC approaches HV and HV+Δ'.

6.3 Analysis of Objective Correlation

Finally, in the third study we focus on comparing the performance our AAC approaches on diverse scenarios on which the correlation between objectives is manually controlled. This study enables to control for the impact of objective correlation on our multi-objective scenarios.

6.3.1 Experimental Protocol

The classical instances of PFSP and TSP used in the previous sections greatly differ in terms of objective correlation. Indeed, in the PFSP Taillard instances, makespan

and total flow time, the two objective under optimisation, are positively very correlated. This is no surprise as they coincide to the maximum and the average, respectively, of the due dates of the flowshop schedule. In contrary, in our TSP instances we optimise the distance of a tour over two independent sets of cities, making the two objectives completely uncorrelated.

To study the specific impact of correlation between objectives on our AAC approaches, we tackle a new set of scenarios for which the correlation is manually controlled. We consider three problems, with the quadratic assignment problem (QAP), two sizes of instances per problem, and three degrees of correlation, following previous a work of Kessaci-Marmion et al. (2017). Specifically, for the sizes of instances, for each of the three problems we consider instances with $n = 50$ and $n = 100$, with n being the number of PFSP jobs, TSP cities, and QAP facilities, respectively.

On PFSP instances, we consider the simultaneous optimisation of two makespan measures computed from two independent matrices of processing times. These processing times are generated following the uniform distribution $\mathcal{U}([1;99])$. To manually correlate the two objectives, a percentage ρ of the processing times of the first matrix is carried over the second matrix, using the coverage method. No correlated instances were obtained using the two independent matrices directly, while medium correlation instances used the value $\rho = 0.6$ and high correlation instances the value $\rho = 0.9$.

On TSP instances, we consider the simultaneous optimisation of the distances of two tours computed from two sets of cities in the Euclidean plane. The position of each city on the plane is determined following the distribution $\mathcal{U}([1;3163]) \times \mathcal{U}([1;3163])$. To manually correlate the two objectives, the second set of cities is obtained by moving each city according to a normal distribution $\mathcal{N}(0, \rho)$. Medium correlation and high correlation instances were generated using the values $\rho = 600$ and $\rho = 150$, respectively.

On QAP instances, we consider the simultaneous optimisation of two costs associated with two flow matrices, the position of the facilities being shared. For both matrices the flow between any two facilities is generated following the uniform distribution $\mathcal{U}([0;99])$. As for the PFSP instances, we use the coverage method to carry over a percentage ρ of the first flow matrix to the second one. Again, no correlated instances were obtained using the two independent matrices directly, while medium correlation and high correlation instances were obtained using the values $\rho = 0.6$ and $\rho = 0.9$, respectively.

As for the configuration space of the MOLS algorithm, we use the same large configuration space as in the second study (Table 6.4), at the slight difference of

the `bound-strat` parameter, which now takes the value `replace` instead of the value `rand`, meaning that after reaching the maximum archive capacity non-dominated solutions are accepted if and only if they replace a solution from the archive, rather than discarding non-dominated solutions uniformly at random.

Lastly, we use the same three approaches as in the second study (i.e., HV, HV+Δ', and HV||Δ'), following the experimental protocol of the large configuration space (see Table 6.5 and Table 6.6).

6.3.2 Optimised Configurations

Table 6.8 to Table 6.13 show the final non-dominated configurations found by the three AAC approaches HV, HV+Δ', and HV||Δ', on all eighteen scenarios (three problems, two size of instances, three degrees of correlation).

PFSP. (Table 6.8 and Table 6.9) The configurations found across all six scenarios are very similar. The combination of the `ndom` exploration with either the `oldest` or the `rand` selection strategy seems to lead to the best performance in terms of hypervolume, while the combination of the `imp` exploration strategy with the `rand` selection strategy leads to solution sets with worse hypervolume but better Δ' spread. While both the `sol` and `arch` exploration reference choices are found within the final configurations for all scenarios, `arch` is slightly more favoured on larger instances, indicating that referencing more stringently against the current archive during exploration is beneficial for larger instance sizes.

TSP. (Table 6.10 and Table 6.11) Compared to the PFSP scenarios, the number of distinct non-dominated configurations is much smaller. The configurations we found vary strongly with both correlation level and problem size. Overall, the `ndom` exploration strategy is preferred, together with either the `rand` or the `oldest` strategy. However, for instance with medium or no correlation, the `arch` exploration reference leads to better HV performance, and may be used together with the `imp-ndom` exploration strategy when the number of cities increases. Furthermore, on the smallest high correlated instances the MOLS algorithm may benefit from using a bounded archive and restart between iterations, while a large archive of size 1000 is chosen (i.e., basically unbounded), along with a kick-based perturbation strategy, for all other instances. This is consistent with the idea that larger TSP instances benefit from a less aggressive perturbation mechanism in combination with a more diverse archive of candidate tours. Finally, correlation between objective has an impact similar to the problem size, making low (and

Table 6.8 – PFSP 50 jobs 20 machines *(optimised configurations)*

1−HV	Δ′	Selection	Exploration	Archive	Perturbation
			(high correlation)		
0.4942	1.3851	`oldest 1`	`ndom arch 1`	1000	`kick 1 10`
0.4945	1.3550	`oldest 1`	`ndom arch 1`	50	`kick 1 5`
0.4951	1.2305	`oldest 1`	`ndom sol 1`	20	`kick 1 3`
0.5052	1.1126	`rand 10`	`ndom arch 10`	1000	`restart`
0.5083	1.0840	`rand 10`	`all`	50	`kick-all 3`
0.5118	1.0449	`rand 10`	`ndom arch 1`	1000	`kick-all 3`
0.5133	1.0367	`all`	`ndom arch 3`	20	`restart`
0.5140	0.7539	`rand 1`	`all-imp arch`	100	`restart`
0.5150	0.6622	`rand 1`	`imp arch 3`	100	`restart`
0.5150	0.6622	`rand 1`	`imp arch 3`	50	`restart`
0.5176	0.3286	`rand 1`	`imp sol 1`	100	`restart`
0.5195	0.1154	`rand 1`	`imp arch 1`	20	`kick 10 10`
0.5198	0.0900	`oldest 1`	`imp arch 1`	20	`kick 1 3`
			(medium correlation)		
0.5032	1.2096	`oldest 1`	`ndom sol 3`	1000	`kick 5 3`
0.5035	1.1864	`oldest 1`	`ndom sol 3`	100	`kick 5 10`
0.5055	1.1257	`oldest 1`	`ndom arch 1`	50	`kick 10 10`
0.5088	1.0564	`rand 10`	`all`	50	`restart`
0.5168	0.8827	`all`	`ndom arch 3`	20	`kick 1 5`
0.5202	0.8554	`rand 1`	`imp arch 3`	1000	`restart`
0.5217	0.7230	`rand 1`	`imp arch 3`	100	`kick-all 5`
0.5217	0.7230	`rand 1`	`imp arch 3`	1000	`kick-all 5`
0.5217	0.7230	`rand 1`	`imp arch 3`	20	`kick-all 10`
0.5222	0.4934	`rand 1`	`imp sol 1`	100	`restart`
0.5222	0.4934	`rand 1`	`imp sol 1`	50	`restart`
0.5223	0.4787	`rand 1`	`imp arch 1`	1000	`restart`
0.5223	0.4787	`rand 1`	`imp arch 1`	20	`restart`
0.5258	0.1951	`rand 1`	`imp arch 1`	20	`kick-all 10`
0.5272	0.1160	`oldest 1`	`imp arch 1`	20	`kick 10 10`
0.5277	0.0283	`rand 1`	`imp arch 1`	100	`kick 1 5`
0.5277	0.0283	`rand 1`	`imp sol 1`	20	`kick 1 10`
0.5277	0.0283	`rand 1`	`imp sol 1`	50	`kick 1 10`
0.5278	0.0254	`oldest 1`	`imp arch 1`	20	`kick 1 3`

Table 6.8 – PFSP 50 jobs 20 machines *(optimised configurations, continued)*

1−HV	Δ′	Selection		Exploration			Archive	Perturbation		
					(no correlation)					
0.5214	0.9715	`rand`	10	`ndom`	`arch`	3	1000	`kick`	5	10
0.5214	0.9709	`rand`	10	`ndom`	`arch`	3	1000	`kick`	10	3
0.5246	0.8414	`rand`	3	`imp-ndom`	`sol`	10	50	`kick-all`		10
0.5317	0.8065	`newest`	1	`ndom`	`arch`	3	50	`kick`	5	10
0.5337	0.8064	`rand`	10	`all`			20	`kick`	5	10
0.5405	0.7626	`rand`	1	`imp`	`sol`	3	100	`kick`	10	3
0.5415	0.5862	`all`		`imp`	`arch`	1	100	`restart`		
0.5415	0.5862	`all`		`imp`	`arch`	1	1000	`restart`		
0.5415	0.5862	`all`		`imp`	`arch`	1	50	`restart`		
0.5415	0.5862	`newest`	10	`imp`	`arch`	1	100	`restart`		
0.5416	0.5792	`oldest`	3	`imp`	`sol`	1	20	`restart`		
0.5416	0.5792	`rand`	10	`imp`	`sol`	1	1000	`restart`		
0.5416	0.5792	`rand`	10	`imp`	`sol`	1	50	`restart`		
0.5416	0.5792	`rand`	3	`imp`	`sol`	1	1000	`restart`		
0.5455	0.3180	`rand`	1	`imp`	`arch`	1	20	`kick`	10	10
0.5455	0.3173	`rand`	1	`imp`	`sol`	1	1000	`kick`	10	10
0.5485	0.2275	`oldest`	1	`imp`	`sol`	1	1000	`kick`	10	5
0.5503	0.1294	`rand`	1	`imp`	`sol`	1	50	`kick`	1	10

Table 6.9 – PFSP 100 jobs 20 machines *(optimised configurations)*

1−HV	Δ′	Selection	Exploration	Archive	Perturbation
			(high correlation)		
0.3543	1.8848	`oldest 1`	`ndom arch 1`	1000	`kick 1 10`
0.3559	1.7607	`oldest 1`	`ndom arch 1`	100	`kick 1 3`
0.3650	1.5112	`oldest 1`	`ndom sol 1`	20	`kick 10 5`
0.3748	1.4685	`oldest 1`	`ndom arch 1`	20	`kick 1 3`
0.3753	1.4292	`oldest 1`	`ndom sol 1`	20	`restart`
0.3769	1.3065	`rand 10`	`all`	20	`restart`
0.3798	1.2688	`newest 10`	`ndom arch 3`	20	`restart`
0.3800	1.2406	`newest 10`	`ndom arch 10`	20	`restart`
0.3802	0.9826	`rand 1`	`all-imp arch`	1000	`restart`
0.3825	0.7963	`rand 1`	`imp arch 3`	1000	`restart`
0.3825	0.7963	`rand 1`	`imp arch 3`	20	`restart`
0.3838	0.7796	`rand 3`	`all-imp sol`	50	`kick 5 3`
0.3841	0.1203	`rand 1`	`imp arch 1`	20	`restart`
0.3845	0.0713	`rand 1`	`imp arch 1`	50	`kick-all 5`
			(medium correlation)		
0.3698	1.7468	`oldest 1`	`ndom arch 1`	1000	`kick 1 10`
0.3699	1.5941	`oldest 1`	`ndom arch 1`	100	`kick 5 3`
0.3707	1.5485	`oldest 1`	`ndom arch 1`	100	`kick 1 5`
0.3722	1.4407	`oldest 1`	`ndom arch 1`	50	`kick 10 3`
0.3787	1.4315	`rand 1`	`ndom arch 10`	1000	`restart`
0.3790	1.3252	`oldest 1`	`ndom arch 1`	20	`kick 10 10`
0.3792	1.2412	`rand 3`	`ndom arch 10`	100	`kick-all 3`
0.3813	1.0833	`rand 3`	`imp-ndom arch 10`	100	`kick 5 10`
0.3817	1.0583	`rand 1`	`imp-ndom sol 10`	100	`kick 5 5`
0.3818	1.0502	`rand 10`	`imp-ndom arch 10`	100	`kick-all 10`
0.3820	1.0159	`rand 3`	`imp-ndom arch 10`	50	`restart`
0.3824	1.0077	`rand 10`	`imp-ndom sol 10`	50	`kick-all 10`
0.3846	0.9621	`rand 10`	`all`	50	`kick 1 10`
0.3855	0.9461	`all`	`ndom arch 3`	20	`kick 1 3`
0.3875	0.8628	`rand 10`	`all`	20	`kick-all 10`
0.3908	0.1667	`rand 1`	`imp arch 1`	1000	`restart`
0.3929	0.0578	`rand 1`	`imp sol 1`	50	`kick 10 10`
0.3931	0.0062	`rand 1`	`imp sol 1`	20	`kick 1 10`
0.3931	0.0062	`rand 1`	`imp sol 1`	50	`kick 1 10`

Table 6.9 – PFSP 100 jobs 20 machines *(optimised configurations, continued)*

1−HV	Δ′	Selection		Exploration			Archive	Perturbation		
					(no correlation)					
0.3869	1.2148	rand	1	ndom	arch	1	1000	restart		
0.3873	1.1696	rand	3	ndom	arch	3	1000	kick-all		3
0.3873	1.1677	rand	3	ndom	arch	3	1000	kick	5	5
0.3873	1.1657	rand	3	ndom	arch	3	1000	kick-all		10
0.3883	0.9964	rand	10	ndom	arch	10	1000	restart		
0.3883	0.9892	rand	10	ndom	arch	10	1000	kick	1	10
0.3893	0.9579	rand	1	ndom	arch	10	100	kick	5	3
0.3894	0.9481	rand	10	ndom	arch	10	100	kick-all		3
0.3894	0.9424	rand	10	ndom	arch	10	100	kick	1	10
0.3906	0.9232	rand	1	imp-ndom	arch	3	100	kick-all		3
0.3906	0.9222	rand	1	imp-ndom	arch	3	100	restart		
0.3935	0.8042	all		ndom	arch	3	50	kick	10	5
0.4035	0.1481	rand	1	imp	sol	1	50	restart		
0.4035	0.1508	rand	1	imp	arch	1	1000	restart		
0.4035	0.1506	rand	1	imp	arch	1	50	restart		
0.4085	0.1480	rand	1	imp	sol	1	50	kick	10	10
0.4087	0.1304	rand	1	imp	arch	1	1000	kick-all		3
0.4117	0.0744	all		imp	arch	1	100	kick	1	5
0.4117	0.0744	all		imp	arch	1	50	kick	1	5
0.4117	0.0744	all		imp	sol	1	1000	kick	1	3
0.4117	0.0744	all		imp	sol	1	20	kick	1	10
0.4117	0.0744	newest	10	imp	arch	1	100	kick	1	3

no) correlated small instances significantly harder and requiring more aggressive mechanisms than equally sized high correlated instances.

QAP. (Table 6.12 and Table 6.13) Again, much fewer configurations were obtained than for the PFSP scenarios. These configurations are much more varied than for the PFSP and TSP, and vary with instance size as well as correlation between the objectives. The `restart` perturbation strategy is favoured for small, 50-facility instances, while kick-based perturbation strategies appear to work better for the larger 100-facility instances. Interestingly, the larger instances seem to be amenable to a wider range of exploration strategies. However, the degree of objective correlation affects the choice of exploration strategy; e.g., for the larger instances, the `imp-ndom` exploration strategy is only chosen when the objectives are uncorrelated. Similarly, we found that bounding the archive size appears to work only well for sufficiently correlated objectives, while the same observation holds for the `oldest` selection strategy.

6.3.3 Discussions

Table 6.14 summarises the performance of our three AAC approaches HV, HV+Δ', and HV||Δ' on all eighteen scenarios, and details the number of final configurations and the range of hypervolume and Δ' indicator values.

Clearly, HV||Δ' produces much larger sets of configurations than HV and HV+Δ', in particular for the PFSP. While HV+Δ' and HV||Δ' achieve overall similar hypervolume values to the dedicated HV approach, on some scenarios (highly correlated PFSP and uncorrelated 100-city TSP), HV achieves the best hypervolume, as could be expected. Surprisingly, on the uncorrelated 100-job PFSP scenario, the HV||Δ' approach performs best in terms of hypervolume. Regarding the complementary Δ' spread indicator, HV||Δ' generally achieves much better results, which are only occasionally matched by the HV+Δ' approach, when the direction of aggregation is compatible with the shape of the optimised front of solutions; but to ensure this is the case, a costly preliminary analysis is required to permit appropriate normalisation of hypervolume and Δ' spread. These observations are consistent with the previous study. As for correlation between objectives, there is no clear overall impact on the three AAC approaches. We note, however, that the single-objective HV approach clearly achieves the best hypervolume for the highly correlated PFSP scenarios.

In conclusion, we showed that the observations of the second study of this chapter generalised on other datasets, in which the correlation between objectives

Table 6.10 – TSP 50 cities *(optimised configurations)*

1−HV	Δ′	Selection	Exploration	Archive	Perturbation
			(high correlation)		
0.1575	0.8045	`oldest 1`	`ndom sol 3`	1000	`restart`
0.1575	0.8040	`oldest 1`	`ndom sol 3`	100	`restart`
0.1575	0.7320	`rand 1`	`ndom sol 1`	50	`restart`
0.1582	0.6705	`rand 1`	`ndom sol 3`	1000	`restart`
0.1582	0.6697	`rand 1`	`ndom sol 3`	100	`restart`
0.1582	0.6462	`rand 1`	`ndom sol 3`	50	`restart`
0.1591	0.6126	`rand 1`	`ndom sol 10`	50	`restart`
0.1599	0.6066	`all`	`ndom arch 10`	50	`restart`
0.1600	0.5911	`oldest 1`	`ndom sol 3`	1000	`kick 10 3`
0.1632	0.5621	`oldest 1`	`imp sol 10`	50	`restart`
0.1642	0.4527	`oldest 1`	`imp arch 1`	100	`restart`
			(medium correlation)		
0.1673	0.6627	`rand 1`	`ndom arch 1`	1000	`kick-all 3`
0.1675	0.6581	`rand 1`	`ndom arch 10`	1000	`kick 10 3`
0.1675	0.6581	`rand 1`	`ndom arch 10`	1000	`kick 10 10`
0.1675	0.6578	`rand 1`	`ndom arch 10`	1000	`restart`
			(no correlation)		
0.1850	0.6769	`rand 1`	`ndom arch 1`	1000	`kick-all 5`
0.1850	0.6767	`rand 1`	`ndom arch 1`	1000	`kick-all 10`
0.1850	0.6763	`rand 1`	`ndom arch 1`	1000	`kick-all 3`
0.1904	0.6555	`rand 1`	`ndom arch 1`	1000	`restart`
0.1957	0.6553	`rand 1`	`ndom sol 1`	1000	`kick-all 10`
0.1968	0.6250	`rand 3`	`ndom sol 3`	1000	`restart`
0.2070	0.6170	`rand 1`	`ndom sol 1`	1000	`restart`

Table 6.11 – TSP 100 cities *(optimised configurations)*

1−HV	Δ′	Selection		Exploration			Archive	Perturbation	
				(high correlation)					
0.1153	0.6292	rand	1	ndom	sol	3	1000	kick	10 3
0.1157	0.6232	rand	3	ndom	sol	10	1000	kick-all	3
				(medium correlation)					
0.1237	0.8163	rand	10	imp-ndom	arch	1	1000	kick	10 10
0.1237	0.6623	rand	3	ndom	sol	10	1000	kick-all	10
0.1237	0.6620	rand	3	ndom	sol	10	1000	restart	
0.1237	0.6615	rand	3	ndom	sol	10	1000	kick	1 3
0.1237	0.6614	rand	3	ndom	sol	10	1000	kick-all	3
0.1237	0.6608	rand	3	ndom	sol	10	1000	kick-all	5
0.1251	0.6546	rand	1	ndom	sol	3	1000	restart	
0.1258	0.6536	rand	1	ndom	sol	1	1000	kick-all	5
0.1268	0.6447	rand	1	ndom	sol	1	1000	restart	
				(no correlation)					
0.1395	0.9566	oldest	1	imp-ndom	arch	1	1000	kick	5 3
0.1395	0.9560	oldest	1	imp-ndom	arch	1	1000	kick	10 3
0.1395	0.9558	oldest	1	imp-ndom	arch	1	1000	kick	5 5
0.1398	0.9020	oldest	3	ndom	sol	10	1000	kick	10 10
0.1398	0.9012	oldest	3	ndom	sol	10	1000	kick	10 5
0.1403	0.8855	oldest	1	ndom	sol	10	1000	kick-all	5
0.1405	0.8574	oldest	10	ndom	sol	10	1000	kick	10 10
0.1416	0.6544	rand	10	ndom	sol	10	1000	kick	10 3
0.1416	0.6530	rand	10	ndom	sol	10	1000	kick	10 5
0.1450	0.6457	rand	1	ndom	sol	3	1000	kick	10 3
0.1451	0.6432	rand	3	ndom	sol	3	1000	kick	10 5
0.1456	0.6361	rand	10	ndom	sol	3	1000	kick-all	3

Table 6.12 – QAP 50 facilities (*optimised configurations*)

1−HV	Δ′	Selection	Exploration	Archive	Perturbation
			(high correlation)		
0.3194	0.8931	`oldest` 1	`ndom sol` 1	1000	`restart`
0.3194	0.8824	`oldest` 1	`ndom sol` 1	100	`restart`
0.3195	0.8724	`oldest` 1	`ndom sol` 1	20	`restart`
0.3208	0.4438	`oldest` 1	`imp arch` 3	50	`restart`
0.3209	0.3434	`rand` 1	`imp arch` 1	50	`restart`
0.3210	0.3090	`oldest` 1	`imp arch` 1	100	`restart`
0.3210	0.3011	`rand` 1	`imp sol` 1	50	`restart`
0.3210	0.2663	`oldest` 1	`imp sol` 1	20	`restart`
0.3213	0.1694	`oldest` 1	`imp arch` 1	20	`kick-all` 10
			(medium correlation)		
0.3211	0.8846	`oldest` 1	`ndom arch` 1	1000	`restart`
0.3211	0.8767	`oldest` 1	`ndom arch` 1	100	`restart`
0.3212	0.8614	`oldest` 1	`ndom sol` 1	100	`kick` 1 3
0.3212	0.8498	`oldest` 1	`ndom sol` 1	1000	`kick` 1 3
0.3212	0.8480	`oldest` 1	`ndom sol` 1	100	`kick` 1 5
0.3224	0.4980	`oldest` 1	`imp arch` 3	100	`kick` 5 3
0.3224	0.1728	`oldest` 1	`imp arch` 1	100	`kick` 5 3
			(no correlation)		
0.3221	0.7973	`rand` 10	`ndom arch` 3	1000	`restart`
0.3221	0.7870	`rand` 1	`ndom arch` 3	1000	`restart`
0.3221	0.7849	`rand` 1	`ndom arch` 3	1000	`kick` 10 3
0.3221	0.7829	`rand` 1	`ndom arch` 3	1000	`kick-all` 10
0.3223	0.7702	`rand` 1	`ndom sol` 10	1000	`kick-all` 10
0.3225	0.7168	`rand` 1	`imp sol` 1	100	`restart`
0.3226	0.7106	`oldest` 1	`imp sol` 1	50	`restart`

Table 6.13 – QAP 100 facilities (*optimised configurations*)

1−HV	Δ′	Selection		Exploration			Archive	Perturbation		
					(high correlation)					
0.3199	0.8395	`oldest`	1	`ndom`	`sol`	1	50	`restart`		
0.3200	0.8158	`oldest`	1	`ndom`	`sol`	1	20	`restart`		
0.3208	0.5251	`rand`	1	`imp`	`arch`	3	1000	`kick`	10	3
0.3209	0.2974	`rand`	1	`imp`	`sol`	3	50	`kick`	5	3
0.3209	0.1000	`rand`	1	`imp`	`sol`	1	100	`kick-all`		10
0.3210	0.0900	`newest`	10	`imp`	`arch`	1	1000	`kick-all`		10
0.3210	0.0800	`all`		`imp`	`sol`	1	50	`kick`	10	3
					(medium correlation)					
0.3209	0.9070	`oldest`	1	`ndom`	`arch`	1	1000	`restart`		
0.3209	0.8865	`oldest`	1	`ndom`	`arch`	1	1000	`kick`	10	3
0.3209	0.8784	`oldest`	1	`ndom`	`sol`	1	1000	`kick`	10	5
0.3209	0.8688	`oldest`	1	`ndom`	`sol`	1	1000	`kick`	1	5
0.3209	0.8389	`oldest`	1	`ndom`	`arch`	1	1000	`kick`	1	3
0.3211	0.8088	`rand`	1	`imp-ndom`	`arch`	1	1000	`kick-all`		3
0.3211	0.7974	`rand`	3	`imp-ndom`	`arch`	1	1000	`kick-all`		3
0.3213	0.4084	`rand`	1	`all-imp`	`sol`		1000	`kick-all`		5
0.3213	0.3026	`rand`	1	`all-imp`	`sol`		100	`kick`	5	5
0.3213	0.2727	`rand`	1	`all-imp`	`sol`		100	`kick`	10	10
0.3213	0.1575	`rand`	1	`imp`	`sol`	3	50	`kick-all`		5
0.3213	0.0000	`rand`	1	`imp`	`arch`	1	50	`kick`	1	10
					(no correlation)					
0.3210	0.7131	`rand`	3	`imp-ndom`	`arch`	1	1000	`restart`		
0.3210	0.7107	`rand`	3	`imp-ndom`	`arch`	1	1000	`kick`	1	3
0.3210	0.7081	`rand`	1	`imp-ndom`	`arch`	1	1000	`kick-all`		3
0.3210	0.7024	`rand`	3	`imp-ndom`	`arch`	1	1000	`restart`		
0.3210	0.7023	`rand`	10	`imp-ndom`	`arch`	1	1000	`kick-all`		10
0.3212	0.2618	`rand`	1	`imp`	`sol`	3	50	`kick`	5	3
0.3212	0.0000	`rand`	1	`imp`	`arch`	1	1000	`kick-all`		5

Table 6.14 – AAC performance: number of final configurations and objective ranges

	#	1−HV values	Δ′ values	#	1−HV values	Δ′ values
		PFSP 50 jobs 20 machines			**PFSP 100 jobs 20 machines**	
			(high correlation)			
HV	3	**0.4942**–0.4951	1.2305–1.3851	3	**0.3543**–0.3753	1.4995–1.8848
HV+Δ′	5	0.4954–0.5191	0.4021–1.5234	3	0.3551–0.3755	1.4856–1.9303
HV||Δ′	13	0.4951–0.5198	0.0900–1.4546	15	0.3551–0.3845	0.0713–1.9302
			(medium correlation)			
HV	1	0.5035	1.1864	3	**0.3698**–0.3707	1.5517–1.7486
HV+Δ′	4	**0.5032**–0.5088	1.0564–1.2096	8	**0.3698**–0.3820	1.0159–1.7468
HV||Δ′	19	0.5035–0.5278	0.0254–1.1879	14	0.3707–0.3931	0.0062–1.5485
			(no correlation)			
HV	1	**0.5214**	0.9709	1	0.3873	1.1677
HV+Δ′	1	**0.5214**	0.9715	5	0.3873–0.3906	0.8222–1.1702
HV||Δ′	17	0.5215–0.5503	0.1294–0.9754	20	**0.3869**–0.4117	0.0744–1.2148
		TSP 50 cities			**TSP 100 cities**	
			(high correlation)			
HV	3	**0.1575**–0.1582	0.6692–0.8045	2	**0.1153**–0.1157	0.6232–0.6292
HV+Δ′	4	0.1582–0.1600	0.5911–0.6705	2	0.1154	0.6275–0.6297
HV||Δ′	6	**0.1575**–0.1642	0.4527–0.7320	2	0.1154–0.1157	0.6233–0.6303
			(medium correlation)			
HV	1	0.1675	0.6578	2	**0.1237**	0.6620–0.8163
HV+Δ′	2	**0.1673**–0.1675	0.6581–0.6627	3	**0.1237**	0.6608–0.6623
HV||Δ′	2	**0.1673**–0.1675	0.6581–0.6632	5	**0.1237**–0.1268	0.6447–0.8156
			(no correlation)			
HV	1	**0.1850**	0.6767	4	**0.1395**–0.1403	0.8841–0.9560
HV+Δ′	1	**0.1850**	0.6763	5	**0.1395**–0.1416	0.6544–0.9566
HV||Δ′	5	**0.1850**–0.2070	0.6170–0.6769	8	**0.1395**–0.1456	0.6361–0.9559

Table 6.14 – AAC performance: number of final configurations and objective ranges (continued)

	#	1−HV values	Δ' values	#	1−HV values	Δ' values
		QAP 50 facilities			**QAP 100 facilities**	
			(high correlation)			
HV	2	**0.3194**–0.3195	0.8810–0.8824	1	**0.3199**	0.8395
HV+Δ'	2	**0.3194**–0.3195	0.8724–0.8931	1	**0.3199**	0.8217
HV||Δ'	8	**0.3194**–0.3213	0.1694–0.8914	6	0.3200–0.3210	0.0800–0.8158
			(medium correlation)			
HV	3	**0.3211**–0.3212	0.8498–0.8846	1	**0.3209**	0.8688
HV+Δ'	3	**0.3211**–0.3212	0.8480–0.8788	5	**0.3209**–0.3213	0.0000–0.8865
HV||Δ'	3	0.3212–0.3224	0.1728–0.8614	8	**0.3209**–0.3214	0.0000–0.9070
			(no correlation)			
HV	2	**0.3221**	0.7870–0.7949	4	**0.3210**	0.7023–1.2196
HV+Δ'	2	**0.3221**	0.7814–0.7960	3	**0.3210**	0.7107–1.2491
HV||Δ'	5	**0.3221**–0.3226	0.7106–0.7973	4	**0.3210**–0.3212	0.0000–0.7081

was controlled.

HV||Δ', the MO-AAC approach, should generally be preferred on all of the datasets we considered. Using similar configuration budgets, it is able to match the performance of both SO-AAC approaches HV and HV+Δ' on their dedicated search direction, while simultaneously efficiently cover the entire Pareto front.

6.4 Perspectives

In this chapter, we conducted three successive studies to analyse the automatic configuration of a static MOLS algorithm on different permutation problems. We first focused on a restricted set of parameters values to perform an exhaustive evaluation of the resulting configuration space, then compared the performance of three AAC approaches, with both the restricted and a larger set of parameter values, before finally specifically investigating the impact of objective correlation on the configuration process.

We detail in the following three perspectives related to multi-objective automatic algorithm configuration in general.

Artificial configuration spaces. The main difficulty while designing a configurator and in general while tackling configuration problems is the sheer amount of time required in order to complete a optimisation pass of the target algorithm parameters. Indeed, it is no surprise that the configuration process is orders of magnitude longer than the time spent by the target algorithm over a single instance. It follows that in-depth analysis of the performance of the configurator itself is again orders of magnitude longer, therefore generally prohibitively expensive. In particular, automatically configuring the configurator itself is mostly inefficient (e.g., see Hutter et al., 2009), even if surrogate models can be used to speed-up the process to a reasonable length (e.g., see Dang et al., 2017).

In order to considerably reduce the time spent by the configuration process, it would be really useful to have access to exhaustively pre-computed target algorithm runs, which would enable to avoid the overhead of effectively running the target algorithm. The obvious problem is that this pre-computation is of course far more expensive than the configuration process itself, even if it can be used multiple times. Another problem would be that the configuration spaces of all target algorithms are not similar, so many of such highly expensive pre-computations would be necessary.

One possible solution would be to have instead a dummy target algorithm, able to instantly produce outputs similar to existing target algorithms. This dummy algorithm should be quick, highly parameterisable so it can generate various configuration spaces, and able to emulates complex parameter interactions. To obtain such an algorithm, it would be necessary to analyse samples of existing configuration spaces (e.g., such as Figure 6.1), to propose a cohesive model (possibly several models) that would enable exploration of configuration space at reasonable cost, thus transforming otherwise prohibitively expensive configuration problem into much faster to solve optimisation problems.

Search space analysis tool. Configuration spaces are hard to visualise. Most importantly, they are very expensive to sample, and the relation between parameters values may be complex and not always clearly discernible. In Figure 6.1, we showed how few parameters could explain how configuration are clustered, while some other produce essentially localised noise, as show Table 6.2 and Table 6.3 in which the setting of perturbation parameters had essentially no effect on the performance of optimal configurations. These analyses were performed by hand, and only possible because first we already knew what would be the most impactful parameters, and then because they were very few parameters overall. A tool able to automatically produce insights on the full parameter space, as opposed to configurators, that only focuses on optimal configurations, could be very useful and

crucial in the obtainment of artificial configuration spaces. In some sense, it would be related to parameter ablation (e.g., see Hutter et al., 2013; Biedenkapp et al., 2017), which aims to identify and focus on impactful parameters.

Other multi-objective applications. MO-ParamILS, our multi-objective extension of ParamILS, is a fully fledged multi-objective configurator, able to accommodate other multi-objective scenario than the solution quality versus running time trade-offs of Chapter 5 and the optimisation of multiple multi-objective performance indicators of Chapter 6. Within the possible applications not tackled in this thesis, three are detailed hereafter.

First, regarding multi-objective algorithms, we only considered in this thesis optimising both hypervolume and Δ' spread, resulting in a trade-off between accuracy and distribution. It would be interesting to investigate other kind of trade-offs, using for example the objectives of the target algorithm directly. For a bi-objective algorithm optimising objectives o_1 and o_2, this could mean for example simultaneously optimising the $n+1$ aggregations ${}^{(k\cdot o_1+(n-k)\cdot o_2)}/_{n}$ (for $k \in 0,\ldots,n$ with $n > 0$) to find complementing variants of the target algorithms efficient regarding distinct directions of the objective space.

It would also be interesting to use multi-objective configuration to optimise the parameters of binary classification algorithms. While the configuration of such algorithms have already been tackled before (AutoWeka, Thornton et al., 2013), the configuration process only considered the loss (misclassification rate) of the underlying learning algorithm. With MO-ParamILS, or any future multi-objective configurator, it would be possible to consider independently multiple performance measures, such as the sensitivity, the specificity, the precision, or the recall.

Then, a frequent performance measure in single-objective configuration is the *penalised average runtime* PAR measure (e.g., Hutter et al., 2009), where failures by the target algorithm are penalised by a given factor; usually a factor of 10 ("PAR10"), but sometimes also as low as 2 or as high as 100. The use of a multi-objective configurator could first enable to analyse more precisely the performance of penalised algorithms (e.g., considering the failure ratio as an independent measure to minimise), while also analysing the PAR measure itself postmortem.

Finally, regarding the trade-off between solution quality and running time, we proposed approaches in which the running time is taken as algorithm parameter, while optimising either the true running time or directly the running time parameter value. Other approaches could be investigated, in particular to focus on optimising the *anytime* properties of the target algorithm, using for example an *area under the curve* -based indicator, or simply using multiple objectives being the performance at given key time points.

Part IV

Automatic Online Design

To succeed, planning alone is insufficient.
One must improvise as well.

Foundation
Isaac Asimov

Chapter 7

MOLS Control

> Intelligence is the ability to adapt to change.
>
> Stephen Hawking

Previously, in Chapter 6, we used automatic configuration approaches in order to optimise the performance of static MOLS algorithms. Conversely, in this chapter, we focus on using simple control mechanisms that are able to modify the current configuration of a MOLS algorithm during its execution, rather than predicting a single best configuration that is used for the entire run.

The motivation of this chapter is to provide complementary tools to the automatic algorithm configuration (AAC) procedure studied in Chapter 6. One of the major drawback of AAC is that it ultimately provides a single configuration of the target algorithm, that will be used on future instances: in addition to being very computationally expensive, only the final prediction matters while slightly less optimal alternatives are ultimately discarded. Furthermore, AAC restricts to the use of the same configuration of the target algorithm for the entire search, leaving no space for combining the strengths and potentials of multiple high-performing configurations. This chapter aims to investigate parameter control as a possible solution to overcome these problems.

In this chapter, We use the adaptive MOLS algorithm presented in Chapter 4, that can incorporate mechanisms to control the exploration of the MOLS algorithm. We focus on only three different exploration strategies (`ndom`, `imp-ndom`, and `imp`) that were analysed in Chapter 6, and two very simple control mechanisms: a uniform control mechanism and an ε-greedy mechanism. Other, more complex, alternatives to the generic control mechanisms are also discussed.

We conduct three successive experiments, using first the three possible strategies, then using only the better two strategies, before finally studying how a *long-term*

learning mechanism could allow to further improve performance by delaying the prediction of the best strategies. The experiments are conducted on the permutation flowshop scheduling problem (PFSP) instances presented in Chapter 1.

The results presented in this chapter are linked to the following publication:

- **Blot, A.**, Kessaci, M., Jourdan, L., and Causmaecker, P. D. (2018c). **Adaptive multi-objective local search algorithms for the permutation flowshop scheduling problem.** In Pardalos, P. and Kotsireas, I., editors, *Learning and Intelligent Optimization – 12th International Conference, LION 12. Revised Selected Papers, Lecture Notes in Computer Science*. Springer. (To appear).

7.1 Adaptive MOLS Algorithm

We first introduce the adaptive MOLS algorithm used in the experiments, and review the possible control mechanisms that can be integrated into it.

7.1.1 Adaptive Algorithm

We use Algorithm 4.2, an adaptive variant of Algorithm 4.1 that use a control mechanism to select an exploration strategy every time the local search is restarted.

We focus on three exploration strategies, using the `imp`, `imp-ndom`, and `ndom` parameter values introduced in Chapter 4. All three explorations use the current archive as reference, and stops after the first accepted neighbour (`explor-size` = 1). These three strategies have been considered to investigate a specific scenario in which there a very good strategy is known (`ndom`), an effective alternative is considered (`imp-ndom`), together with a less effective one (`imp`). As for the other parameters, for the selection phase a single neighbour is selected uniformly at random (`select-strat` = `rand` and `select-size` = 1), the archive is unbounded (`bound-strat` = `unbounded`), and for the perturbation step a single configuration is selected uniformly at random and then replaced three times by one neighbour selected uniformly at random (`perturb-strat` = `kick`, `perturb-size` = 1, and `perturb-strength` = 3). These choices are motivated by preliminary expert knowledge that includes observations resulting from Chapter 6

As for the initialisation of the MOLS algorithm, we use a simple single-objective greedy algorithm on the two objectives independently. Indeed, using smarter initialisation procedures, the starting solutions would be too close to the optimal Pareto front, which is undoubtedly detrimental to the current study since we aim

to emphasise the impact of the control mechanisms over the algorithm itself. To obtain two solutions of reasonable quality, we choose the NEH procedure (Nawaz et al., 1983) for the two objectives independently. NEH is often used to seed state-of-the-art PFSP initialisation procedures as for example the 2-phase local search algorithm (Dubois-Lacoste et al., 2011b), which is the initialisation procedure used on the PFSP instances in Chapter 6.

The termination criterion of both the static algorithms and the adaptive algorithms is a total running time fixed to $n^2m/500$ CPU seconds. While being twice as long as the running time used in Chapter 6, we are here less constrained as we run the *target* MOLS algorithm orders of magnitude lower than in the previous configuration scenarios. The termination criterion of the inner MOLS algorithm (Algorithm 4.3) is a combination of either n^2 solution evaluations or n iterations without improvement. These criteria are well adapted to the PFSP since they enable a sufficient number of iterations of both the inner algorithm and the control mechanism.

In the following experiments, these termination criteria resulted in about 1600 executions of the inner MOLS algorithm for instances with 20 jobs, then about 750, 400, 250 and 100 iterations for instances with 50, 100, 200 and 500 jobs, respectively. This decrease of the number of executions when the number of jobs increases is explained by the exploration step that becomes more and more long and challenging as the size of the neighbourhood quickly grows (typical PFSP neighbourhoods are quadratic in the number of jobs of the instance).

7.1.2 Generic Online Mechanisms

Parameter control mechanisms are generally classified between deterministic, adaptive and self-adaptive approaches, following the taxonomy of Eiben et al. (1999) (see Chapter 2). In the following, we focus on adaptive approaches that are parameter and algorithm-independent. More in-depth surveys on parameter control, focused on evolutionary algorithms, can be found in (Eiben et al., 2007; Karafotias et al., 2015; di Tollo et al., 2015; Aleti and Moser, 2016).

To facilitate comparisons between the approaches that we present in the following, we use the following notations. *Arms* designate the different variants of the algorithm being controlled. More precisely, an arm can be related to either a single or a combination of specific parameter values, operators, or specific strategies of the algorithm. For approaches using probabilities, the arm i is chosen at time t with probability $p_i(t)$, returning a reward $r_i(t)$. In addition, two scalars $q_i(t)$ and $n_i(t)$ can also be defined at time t to estimate the reward of the arm i and indicate how many times the arm i was chosen, respectively. Finally, we will suppose that

there is a finite, positive, number of arms N.

Uniform Random Control

The most simple random control mechanism is based on uniform distribution of arms (see Equation 7.1).

$$p_i(t+1) = \frac{1}{N}, \text{ for all arms } i \tag{7.1}$$

Probability Matching

Other random approaches exist, that use a specific distribution to select the next arm: arms with better results on average in the past are assigned a higher probability to be selected. The second most simple random algorithm is then perhaps the probability matching algorithm (Thierens, 2005), that simply assigns probabilities linearly to their expected reward (see Equation 7.2).

$$p_i(t+1) = \frac{q_i(t)}{\sum_{\text{arm } j} q_j(t)}, \text{ for all arms } i \tag{7.2}$$

Additionally, to avoid situations when an arm is never selected again because its expected reward is too low, a minimum probability p_{min} (with $0 < p_{min} < {}^1/_n$) and maximum probability p_{max} (with $p_{max} = 1 - n \cdot p_{min}$) can be introduced to ensure minimal exploration (e.g., see Equation 7.3; Thierens, 2005).

$$p_i(t+1) = p_{min} + p_{max} \cdot \frac{q_i(t)}{\sum_{\text{arm } j} q_j(t)}, \text{ for all arms } i \tag{7.3}$$

For all control mechanisms in general, considering the average reward at time t (see Equation 7.4, with $n_i(t)$ the number of times the arm i was selected at time t) only works well for stationary systems.

$$q_i(t+1) = \begin{cases} \frac{n_i(t) \cdot q_i(t) + r_i(t)}{n_i(t)+1}, & \text{if the arm } i \text{ was selected} \\ q_i(t), & \text{otherwise} \end{cases} \tag{7.4}$$

For non-stationary systems, when the estimate of the reward of an arm is only reliable when the rewards received are not too old, a solution is to use a recency-weighted average that updates the current estimate with a fraction of the difference of the target value and the current estimate (see Equation 7.5; Thierens, 2005). This solution use an adaptation rate α (with $0 < \alpha \leqslant 1$), where the value $\alpha = 1$ means

that only the last reward is used, and values of α close to 0 mean that the estimate is only slightly steered with recent reward values.

$$q_i(t+1) = \begin{cases} r_i(t) + (1-\alpha) \cdot q_i(t), & \text{if the arm } i \text{ was selected} \\ q_i(t), & \text{otherwise} \end{cases} \tag{7.5}$$

Softmax algorithms

Another possibility is to use a Boltzman distribution with the probability to select arm i at time $t+1$ is given in terms of the average rewards at time t (Sutton and Barto (1998); see Equation 7.6).

$$p_i(t+1) = \frac{e^{q_i(t)/\tau}}{\sum\limits_{\text{arm } j} e^{q_j(t)/\tau}}, \text{ for all arms } i \tag{7.6}$$

In Equation 7.6, τ is a temperature parameter that can be taken constant or decreasing (Cesa-Bianchi and Fischer, 1998; Vermorel and Mohri, 2005).

Adaptive Pursuit

Pursuit algorithms relate to classical techniques from machine learning in that the probability to select any arm is adjusted after each selection of a specific arm. Given a learning rate β, the probabilities at $t+1$ are adapted after t (Thathachar and Sastry (1985); Rajaraman and Sastry (1996); Sutton and Barto (1998); see Equation 7.7).

$$p_i(t+1) = \begin{cases} p_i(t) + \beta \cdot (1 - p_i(t)), & \text{if } i = arg\ max_j\ q_j(t) \\ p_i(t) + \beta \cdot (0 - p_i(t)), & \text{otherwise} \end{cases} \tag{7.7}$$

In order to deal with non-stationary contexts, *adaptive* pursuit was conceived (Thierens, 2005). In adaptive pursuit, again, a minimum and a maximum (p_{min} and p_{max}, with $0 < p_{min} < {}^1/_n$ and $p_{max} = 1 - n \cdot p_{min}$) for the probabilities p_i are introduced to leave room for exploration (see Equation 7.8).

$$p_i(t+1) = \begin{cases} p_i(t) + \beta \cdot (p_{max} - p_i(t)), & \text{if } i = arg\ max_j\ q_j(t) \\ p_i(t) + \beta \cdot (p_{min} - p_i(t)), & \text{otherwise} \end{cases} \tag{7.8}$$

Multi Armed Bandits

Multi armed bandits (MAB) algorithms, metaphorically referring to the infamous gambling machines, model a decision problem where the only (or main in some

versions) source of information is history of previous selections (Lai and Robbins, 1985; Sutton and Barto, 1998; Auer et al., 2002). The only decision to be taken is which arm to pull next. The aim is to maximise the expected outcome of a finite series of decisions. In an adaptive local search setting, MAB algorithms can be used to model the decisions to be taken on which sub-algorithm or neighbourhood to select in the next step given the history of the current search. MAB algorithms have been applied in evolutionary algorithms (Costa et al., 2008; Maturana et al., 2009; Fialho et al., 2010; Belluz et al., 2015) as well as in evolution based hyperheuristic settings (Sabar et al., 2015). An early example of an application to combinatorial optimisation in networks is Gai et al. (2012). The design of algorithms using MAB was studied in Drugan and Nowé (2013); Yahyaa et al. (2014).

To cope with dynamic situations, MAB-based control approaches can for example be augmented with restart mechanisms, e.g., the dynamic MAB of (Costa et al., 2008; Maturana et al., 2009) coupled with a Page-Hinkley test, as well with sliding mechanisms to only focus on the newest rewards (Fialho et al., 2010).

Upper Confidence Bound

The upper confidence bound 1 (UCB1) algorithm is a MAB algorithm based on the principle of *optimism in the face of uncertainty* (Auer et al., 2002). Every decision, it selects the arm that optimises the expected reward while simultaneously minimises the associated *regret* (see Equation 7.9).

$$p_i(t+1) = \begin{cases} 1, & \text{if } i = arg\ max_j \left(q_j(t) + \sqrt{\frac{2 \cdot log(t)}{n_j(t)}} \right) \\ 0, & \text{otherwise} \end{cases} \tag{7.9}$$

While the left part of the equation, $q_j(t)$ is simply the expected reward associated to the arm j (Equation 7.4), the right part ensures that every arm will eventually being selected an infinite number of times. In the context of algorithm control, a scaling of the right part of the equation by a given constant is necessary to accommodate non-Boolean rewards (Costa et al., 2008).

ε-Greedy

MAB algorithms, having only history to learn from, need to make decisions that both optimise the immediate result and optimise the lessons learned for better future results. Handling this exploitation versus exploration dilemma is what makes a strategy for a MAB. A simple strategy is to pick the decision that has delivered the best result on average in the past. The MAB can then eventually start with an exploration phase where every arm is tried once or a predetermined number of

times, after which the greedy strategy is used, adjusting the averages after every decision. In dynamical situations, the average may be weighted to introduce a bias towards recent history. These approaches are termed "greedy". In a slightly more explorative approach, a probability is introduced to allow for random selection of an arm, independently of its average success rate. This kind of approaches are called ε-greedy (e.g., see Aleti and Moser, 2016). One possibility is to select at time $t+1$ the best performing arm with a probability of $(1-\varepsilon)$, leaving a probability of ε to uniformly select an arm at random (see Equation 7.10).

$$p_i(t+1) = \begin{cases} (1-\varepsilon) + \varepsilon/N, & \text{if } i = arg\ max_j\ q_j(t) \\ \varepsilon/N, & \text{otherwise} \end{cases} \tag{7.10}$$

Reinforcement Learning

In addition to, as in MAB algorithms, *simply* learning the optimal arm to select, reinforcement learning algorithms (Sutton and Barto, 1998) also focus on describing the possible states the system can be in, to then learn the optimal arm to select in each possible state. Examples of RL algorithms used for parameter control include Eiben et al. (2006); Whiteson and Stone (2006); Sakurai et al. (2010); Karafotias et al. (2014).

Other Approaches

Other simple approaches can also be devised. An hybrid mechanism between uniform random control and probability matching would be one in which the different probabilities for each arm are statistically defined before the execution. These probabilities could be set using expert knowledge, with regard to the expected reward for each arm, or also optimised using an offline automatic configurator.

It would also be possible to use a deterministic approach, in which each arm is used following a sequence defined before the execution. Again, this sequence could be hand crafted using expert knowledge but also automatically worked out using an offline automatic configurator.

Finally, other more complex or less general control approaches include the probabilistic rule-driven adaptive model (PRAM, Wong et al., 2003), predictive parameter control (Aleti and Moser, 2011; Aleti et al., 2014), or adaptive range parameter control (Aleti et al., 2012; Aleti and Moser, 2013).

7.2 Experimental Protocol

In the experiments, we first compare the three deterministic instantiations of Algorithm 4.1, each using a single exploration strategy (denoted simply by `imp`, `imp-ndom`, and `ndom`, respectively), to adaptive algorithms (Algorithm 4.2), using a basic random control mechanism or a ε-greedy control mechanism. While many of the other more sophisticated mechanisms could have also been compared, as well as other existing adaptive MOLS algorithms (Veerapen and Saubion, 2011; Gretsista and Burke, 2017), they would likely be very similar as only three arms were considered, which considerably limits the number of dissimilar decision strategies.

In the random control mechanism decisions are uniformly taken at random, without any feedback from the search, in contrary to the ε-greedy control mechanism that uses feedback to take decisions. This feedback is computed every iteration using the hypervolume difference between the hypervolume of the new archive and the one of the previous iteration. It is then used to update the reward associated to the current strategy using a learning rate $\alpha = 0.8$ (Equation 7.5). In this study, we set $\varepsilon = 0.1$ (Equation 7.10), meaning that the best performing strategy (i.e., the arm $\arg\max_i q_i(t)$) is chosen with 93.3% probability, either strategy being selected uniformly at random otherwise (3.33% probability each).

For both control strategies, we consider four different variants, that differ by the subset of exploration strategies that are available. First, the three exploration strategies are available for both adaptive algorithms (`rand_3`, `greedy_3`). Note that it is already known that the `imp` strategy leads to poorer results on the PFSP. But, we still decide to make available this bad strategy in order to evaluate the control mechanism without any a priori knowledge. Secondly, we use this expertise and only make available the two strategies `imp-ndom` and `ndom` for both adaptive algorithms (`rand_2`, `greedy_2`). Finally, the last two variants introduce a *long-term learning* scheme, beginning with the three strategies but switching to only use the two best strategies during the search. Two possibilities are evaluated: either after half the total running time of both adaptive algorithms (`rand_ltl_50`, `greedy_ltl_50`), or after twenty percent of the total running time (`rand_ltl_20`, `greedy_ltl_20`).

The experimental protocol is reduced to the simple exhaustive comparison of all approaches on all benchmark instances. Experiments are conducted across all classical PFSP Taillard instances, separated in twelve benchmarks of 10 instances sharing the same number of jobs and machines. Because of the stochasticity of both the algorithm and the control mechanisms, all approaches are run 20 times

Table 7.1 – Experiments summary

Type	Approach	3 arms	2 arms	LTL
Deterministic	`imp`	✓		
Deterministic	`imp-ndom`	✓	✓	
Deterministic	`ndom`	✓	✓	
Random	`rand_3`	✓		✓
Random	`rand_2`		✓	✓
Random	`rand_ltl_50`			✓
Random	`rand_ltl_20`			✓
ε-greedy	`greedy_3`	✓		✓
ε-greedy	`greedy_2`		✓	✓
ε-greedy	`greedy_ltl_50`			✓
ε-greedy	`greedy_ltl_20`			✓

on each instance, using a given set of 20 random seeds.

In total, the eleven approaches (three deterministic, eight adaptive) are compared in 4 successive steps as detailed in Table 7.1. First, we compare the three deterministic approaches with the two adaptive approaches that use all three explorations strategies. Then, we focus on the two best strategies, and compare the respective two deterministic approaches with the two adaptive approaches that use them only. Finally, we investigate the potential of a *long-term learning* scheme for the two control mechanisms independently, first by switching from three arms to two arms after half of the runtime has passed (`rand_ltl_50`, `greedy_ltl_50`), then after only twenty percent of the runtime (`rand_ltl_20`, `greedy_ltl_20`).

7.3 Experimental Results

The experiments have been conducted on part of the cluster of the ORKAD research group, CRIStAL laboratory, at the University of Lille, France. The two nodes used are equipped with two 12-core 3.00GHz Intel Xeon E5-2687W v4 CPUs with 8MB L3 cache and 64GB RAM, running Archlinux. Computations were conducted in parallel as much as possible.

Table 7.2, Table 7.3, and Table 7.4 present the rankings resulting of the experiments for the twelve instance sizes, together with the resulting average ranks. For each instance size, the ranking is computed using pairwise Wilcoxon signed rank

Table 7.2 – 3-arm ranking

Approach	Instance (N, M)												Avg.
	20			50			100			200		500	
	5	10	20	5	10	20	5	10	20	10	20	20	
`imp`	5	5	5	5	5	5	5	5	5	5	5	5	5
`imp-ndom`	4	4	3	4	4	4	4	1	2	1	2	1	2.8
`ndom`	1	1	3	1	1	1	1	1	1	1	1	1	1.2
`rand_3`	1	1	1	1	1	1	1	1	2	3	3	3	1.6
`greedy_3`	1	1	1	1	1	1	1	1	2	3	3	3	1.6

tests; a Friedman test post hoc analysis is used to check the statistical equivalence between algorithms ranked 1 and their difference with the others.

7.3.1 3-arm Results

First, we focus on the 3-arm adaptive approaches, `rand_3` and `greedy_3`, comparing them to the three respective deterministic approaches `imp`, `imp-ndom` and `ndom`. As detailed on Table 7.2, the `imp` and `ndom` approaches always perform very poorly and very well, respectively. Meanwhile, the `imp-ndom` approach performs rather poorly in small instances, but achieves very good results on the largest ones. Surprisingly, the two adaptive approaches (`rand_3` and `greedy_3`) equivalently perform. More precisely, they perform very well on the first eight instances (rank 1), but their performance are more debatable on the four largest ones. Indeed, for instances with 100 jobs and 20 machines, they are outperformed by the deterministic approach `ndom` and equivalently perform with the `imp-ndom` approach. But, for instances with 200 and 500 jobs, they are also outperformed by this latter approach. In these cases, they are still better than the `imp` approach. This results show that the `imp` approach affects more the adaptive algorithms when the problem gets harder.

7.3.2 2-arm Results

Then, in the second step, as shown on Table 7.3, the `imp` approach is discarded and we focus on the 2-arm adaptive approaches (`rand_2` and `greedy_2`) that have only the choice between the `imp-ndom` and `ndom` exploration strategies. Once again, the two adaptive approaches equivalently perform. However, they are rank 1 for all the instances except for the 100-jobs 20-machines instances (rank 2). Con-

Table 7.3 – 2-arm ranking

Approach	Instance (N, M)												Avg.
	20			50			100			200		500	
	5	10	20	5	10	20	5	10	20	10	20	20	
`imp-ndom`	4	4	3	4	4	4	4	4	4	4	4	1	3.7
`ndom`	1	1	3	1	1	1	1	1	1	1	1	1	1.2
`rand_2`	1	1	1	1	1	1	1	1	2	1	1	1	1.1
`greedy_2`	1	1	1	1	1	1	1	1	2	1	1	1	1.1

sidering only the well performing exploration strategies largely improves the adaptive approaches for the largest instances.

7.3.3 Long Term Learning Results

Having validated that the `imp` arm should not be used on larger machines instances, we finally investigate a long-term learning scheme where arms can be discarded during the execution of the algorithm if they are worse than the others. Two approaches have been tested: the discard of the worst strategy is done after either fifty percent (`rand_ltl_50`, `greedy_ltl_50`) or twenty percent (`rand_ltl_20`, `greedy_ltl_20`) of the total running time. In order to effectively analyse this long-term learning scheme, the two adaptive approaches are investigated and ranked separately. Results are shown on Table 7.4. Unsurprisingly, the 2-arm versions of both adaptive approaches always statistically outperform their respective 3-arm versions and so for the versions using the long-term learning. Introducing long-term learning to only keep well-performing arms is efficient. The ranking between the two control mechanisms `rand` and `greedy` are not the same size by size, but the average ranks show that it is more efficient to discard an arm sooner since `rand_ltl_20` and `greedy_ltl_20` are better ranked than `rand_ltl_50` and `greedy_ltl_50` respectively. These results demonstrate how control mechanisms can effectively identify and evaluate the performance of strategies during the search.

7.4 Discussions

Table 7.5 summarises all experiments and shows the overall ranking of the eleven approaches on each instance size and shows the final average ranks.

Table 7.4 – Long-time learning ranking

Approach	Instance (N, M)												Avg.
	20			50			100			200		500	
	5	10	20	5	10	20	5	10	20	10	20	20	
rand_3	4	4	2	4	4	4	4	4	4	4	4	3	3.8
rand_ltl_50	3	1	2	1	1	1	3	3	3	2	3	3	2.2
rand_ltl_20	1	1	2	1	1	1	1	1	1	2	2	2	1.3
rand_2	1	1	1	1	1	1	1	1	1	1	1	1	1
greedy_3	1	1	1	1	4	4	4	4	4	4	4	3	2.9
greedy_ltl_50	1	1	1	1	1	1	3	3	3	3	2	3	1.9
greedy_ltl_20	1	1	1	1	3	1	1	1	1	1	2	2	1.3
greedy_2	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 7.5 – Complete ranking

Approach	Instance (N, M)												Avg.
	20			50			100			200		500	
	5	10	20	5	10	20	5	10	20	10	20	20	
imp	11	11	11	11	11	11	11	11	11	11	11	11	11
imp-ndom	10	10	9	10	10	10	9	7	6	4	4	1	7.5
ndom	1	1	9	1	1	1	1	1	1	1	1	1	1.7
rand_3	9	9	6	9	9	7	10	10	9	9	9	9	8.8
rand_ltl_50	8	1	6	1	5	1	7	7	6	6	7	7	5.2
rand_ltl_20	1	1	6	1	5	1	5	1	2	6	5	5	3.2
rand_2	1	1	1	1	1	1	1	1	2	1	1	1	1.1
greedy_3	1	1	1	1	8	9	8	9	9	9	10	9	6.2
greedy_ltl_50	1	1	1	1	1	7	5	6	8	6	7	7	4.2
greedy_ltl_20	1	1	1	1	5	1	1	1	2	4	6	5	2.4
greedy_2	1	1	1	1	1	1	1	1	2	1	1	1	1.1

Regarding the three deterministic approaches, while the `imp` approach is always ranked last, the `ndom` approach is almost always ranked first, only beaten on the 20-job 20-machine instance where it is outperformed by all adaptive approaches. Regarding the adaptive approaches, both the 2-arm approaches `rand_2` and `greedy_2` are the best performing, then are ranked the `ltl_20` ones and the `ltl_50` ones. The approaches using the random control mechanism generally perform worse than the ones using the ε-greedy mechanism, especially for the 3-arm variants and the long-term learning variants. Interestingly, even the random adaptive approach performs really well when considering the two `imp-ndom` and `ndom` arms, potentially meaning that the adaptive algorithms will achieve very good results as long as there is no critically bad performing arm available. However, the long-term learning variants show that it is possible to identify, remove and recover in such event.

In conclusion, our experiments showed that is was possible, and beneficial, to combine multiple MOLS configurations during the search. While the adaptive algorithm did not in the end outperformed the best static MOLS variant, it was able to statistically match its performance. Note that the configuration provided by a AAC procedure is relative to a given distribution of training instances, and aims to optimise the performance on all instances; being able to use multiple configurations means a better probability of using the best configurations of the particular instance being solved.

Finally, we also showed that if a *bad* configuration was initially provided to the adaptive MOLS algorithm, it was possible to identify and remove it during the search while still achieving good results.

7.5 Perspectives

In this chapter we focused on one of the feature and drawback of offline algorithm design (Part III): it predicts a single configuration whose parameters values stay fixed during the execution.

More specifically, we focused on integrating generic control mechanisms in multi-objective local search (MOLS) algorithms, to enable the use of multiple strategies during the execution of the MOLS algorithm. We showed that even very simple control mechanisms were able to efficiently combine multiple exploration strategies without noticeable performance loss, given that the least effective strategies are correctly and rapidly identified.

In the following, we detail four perspective that are related to this chapter.

Controlling a larger number of arms. We focused on using only three arms, which individual performance were already well known due to previous experiments, in order to be able to better understand and explain our results.

Having demonstrated that it was possible to identify and focus on the best-performing strategies during the run, the natural next step is to consider more alternative strategies, possibly on other problems (e.g., the travelling salesman problem or the quadratic assignment problem described in Chapter 1), to study scenarios in which the best arms are less clear.

Multi-parameters control mechanisms. Our adaptive MOLS algorithm focused on a single parameter of the static MOLS algorithm: the exploration strategy, while every other parameter was fixed. However, offline mechanisms (also, configuration scheduling) are able to optimise many more individual independent parameters. Another perspective would be to focus on simultaneously controlling multiple parameters of the static MOLS algorithm, for example either two parameters from the exploration step (e.g., the exploration strategy and the number of neighbours sought as an additional numerical parameter), or two different categorical parameters (e.g., the selection and explorations strategies).

Using more complex control mechanisms. Naturally, more alternative strategies means that it would be required to use the more advanced control mechanisms that were until now not considered. After the very simple mechanisms that we used, the most relevant mechanisms would be those directly based on multi-armed bandits algorithms and reinforcement learning.

Offline design of adaptive algorithms. Finally, control mechanisms themselves often expose new parameters that need to be set, in addition to the parameters of the underlying algorithm. There may also be alternatives in use of the feedback that is used to compute performance predictions. Another natural perspective would then be to use an offline configurator (e.g., (MO-)ParamILS, presented in Chapter 5) to automatically select the best performing control mechanisms, together with its set of parameter and the most suitable feedback source.

Chapter 8

MOLS Configuration Scheduling

Weak emperors mean strong viceroys.

Foundation
Isaac Asimov

In Chapter 6, we analysed the *offline* automatic design of a static multi-objective local search (MOLS) algorithm, using automatic algorithm configuration (AAC) and more specially the MO-ParamILS, the multi-objective automatic configurator presented in Chapter 5. Then, in Chapter 7, we investigated the *online* automatic design of an adaptive MOLS algorithm, through the use of generic control mechanisms, to complement and overcome some of the drawbacks of AAC.

In this chapter, we investigate the use of schedules of MOLS configurations, and their automatic offline configuration. That is, instead of either predicting the best single MOLS configuration (algorithm configuration), or dynamically trying to find the best parameters values during the search (parameter control), we propose to predict the best sequence of configurations, in order to enable more modularity and increase potential performance. This approach is described in detail in Chapter 4.

We more specifically aim to overcome one specific drawback of AAC: that different configurations may be optimal at different parts of the search. Indeed, it may for example be beneficial to use some strategies at the start of the search to start from good solutions, then switch to another strategy to efficiently converge close to the optimal solutions, then finally use yet another strategy once improving solutions becomes very hard.

After having presented the subset of the static MOLS configuration space that we use in the experiments, we investigate the automatic configuration of MOLS schedules in three successive steps. First, we perform an exhaustive analysis of

Table 8.1 – Investigated MOLS configuration space (60 configurations)

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`select-size`	1
Exploration	`explor-strat`	{`all`, `all-imp`, `imp`, `imp-ndom`, `ndom`}
Exploration	`explor-ref`	`arch`
Exploration	`explor-size`	5
Archive	`bound-strat`	`unbounded`
Perturbation	`perturb-strat`	{`kick`, `kick-all`, `restart`}
Perturbation	`perturb-size`	1
Perturbation	`perturb-strength`	3

the subset of the MOLS configuration space, thus constructing the baseline that can be obtained by using schedules containing a single configurations. Then, we successively analyse the automatic design of schedules accommodating two, and then three configurations, while the possible time budget splits are set in advance.

The experiments are conducted on the permutation flowshop scheduling problem (PFSP) instances presented in Chapter 1.

8.1 MOLS Configurations

We use the static MOLS algorithm presented in Chapter 6, using the configuration space described in Table 8.1. This configuration space Θ only comprises 60 possible configurations.

Three configurable dynamic algorithm frameworks are considered, in which multiple types of schedules are allowed. We denote by K the maximal length k allowed for the schedules. First, as a baseline, we consider frameworks with $K = 1$, meaning that the schedule is constituted by a single configuration for the entire run. Then, we investigate frameworks with $K = 2$ and $K = 3$, i.e., enabling the use of schedules of size 2 and 3, respectively. For the frameworks with $K = 2$, we consider possible time budgets of (T), $(T/2, T/2)$, $(T/4, 3T/4)$, and $(3T/4, T/4)$. For the frameworks with $K = 3$, we consider in addition possible time budgets of $(T/3, T/3, T/3)$, $(T/4, T/4, T/2)$, and $(T/2, T/4, T/4)$. These seven types of schedules are pictured in Figure 8.1. As every sub-configuration of a schedule has 60 possible values, for $K = 2$ (i.e., $k \in 1, 2$), this results in a final configuration space of $60 + 3 \cdot 60^2 = 10\,860$ possible schedules. For $K = 3$, (i.e., $k \in 1, 2, 3$), this results in a final configuration space of $60 + 3 \cdot 60^2 + 3 \cdot 60^3 = 658\,860$ possible schedules, a much

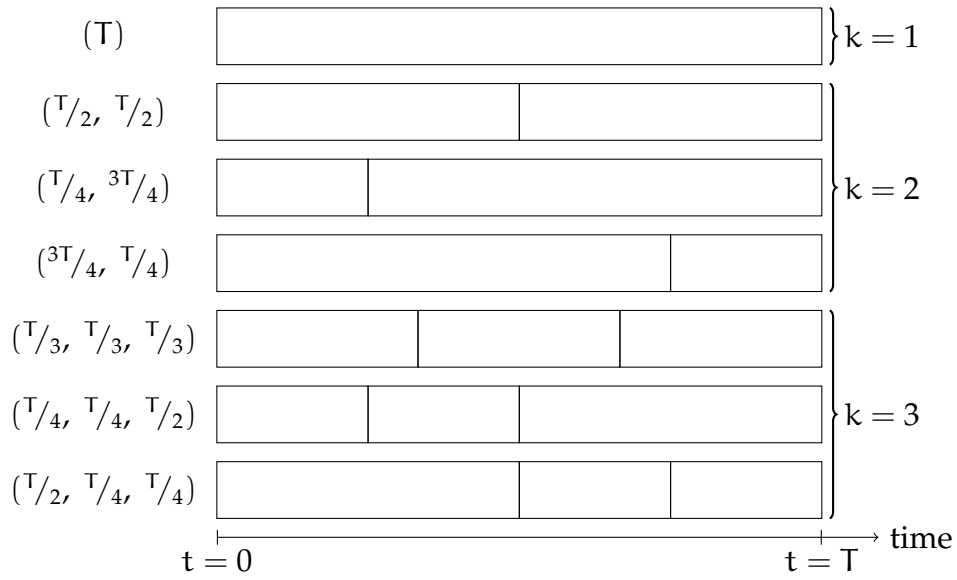


Figure 8.1 – The seven types of schedules used in the experiments

bigger space that the *large* space of Chapter 6. This exponential growth explain why only schedules of size $k \in 1, 2, 3$ have been considered, why we restricted the initial Θ configuration space to only 60 configurations, and why the possible time budget were statistically fixed.

As for the algorithm itself, its entire time budget is devoted to the configuration schedule and an instantaneous initialisation of 10 solutions uniformly taken at random from the search space was preferred to more efficient but longer initialisation procedures. The total running time is fixed to $T = {}^{n^2 m}/_{1\,000}$ CPU seconds as in Chapter 6.

8.2 Experimental Protocol

Three experiments are conducted successively, to consider the different sizes of frameworks independently. First, we consider the $K = 1$ frameworks. As there are only 60 possible schedules using a single configuration, we investigate them exhaustively by aggregating their performance over 15 runs on each of the 10 Taillard instances. Similarly as in previous chapters, we use both the hypervolume (HV) and the Δ' spread.

Then, for both $K = 2$ and $K = 3$ frameworks, as the number of possible schedules grows exponentially we use MO-ParamILS to automatically configure the sequences of configurations. We use the configuration protocol of Chapter 5. For training, we generated 100 new Taillard-like instances, using the original instance

Table 8.2 – Training computational time

Jobs	Machines	1 MOLS run	$K = 1$	$K = 2$	$K = 3$
20	20	8 seconds	8 minutes	2.22 hours	22.22 hours
50	20	50 seconds	50 minutes	13.89 hours	5.79 days
100	20	3.33 minutes	3.33 hours	2.31 days	23.15 days
200	20	13.33 minutes	13.33 hours	9.26 days	92.59 days

generator process. MO-ParamILS was run 20 times for both frameworks, with a configuration budget of 1 000 MOLS runs for $K = 2$ (for 10 860 possible schedules) and 10 000 runs for $K = 3$ (for 658 860 possible schedules).

Table 8.2 describes, for different PFSP instance sizes, the running time of a single MOLS runs, the time required to exhaustively evaluate $K = 1$ frameworks (i.e., 60 MOLS runs), and the training time of a single MO-ParamILS runs for $K = 2$ (i.e., 1 000 MOLS runs), and $K = 3$ (i.e., 10 000 MOLS runs). Note that to obtain the total computation time, theses training times (for $K = 2$ and $K = 3$) should be multiplied by the number of MO-ParamILS runs (here, 20), and that the time required for the validation and test steps are not included.

In the following, we choose to conduct the experiments on two smallest classes of small PFSP instances of Table 8.2. We focus on the classical Taillard instances with 50 jobs and 20 machines, as they are already much harder to solve than smaller instances, while keeping the running time quick enough (50 CPU seconds). They are also the smallest PFSP instances considered in Chapter 6. In addition to them, we also consider smaller instances, with only 20 jobs and 20 machines, as they are much faster to solve. Due to the considerable amount of time required by the experiments, larger size of instances (e.g., 100 or 200 jobs) have not been considered.

8.3 Experimental Results

The experiments have been conducted on part of the cluster of the ORKAD research group, CRIStAL laboratory, at the University of Lille, France. The two nodes used are equipped with two 12-core 3.00GHz Intel Xeon E5-2687W v4 CPUs with 8MB L3 cache and 64GB RAM, running Archlinux. Computations were conducted in parallel as much as possible.

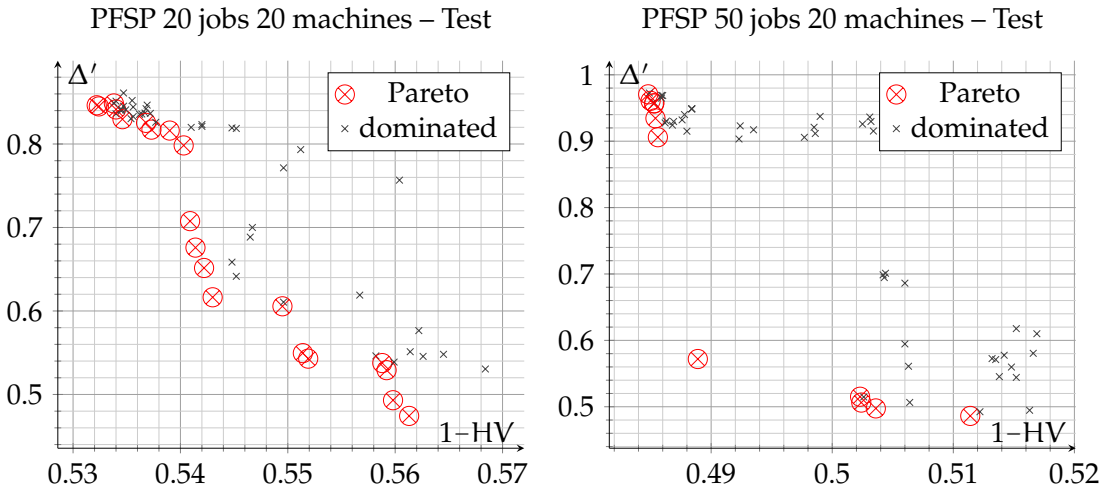


Figure 8.2 – Initial search space and optimal configurations (K = 1; left: 20 jobs; right: 50 jobs)

8.3.1 Exhaustive Enumeration

Figure 8.2 shows the average performance of all 60 possible schedules using exactly one configuration (K = 1) over the 10 Taillard instances (left: instances with 20 jobs and 20 machines; right: instances with 50 jobs and 20 machines). Pareto optimal configurations are circled in red, while also described in Table 8.3 and Table 8.4.

On the 50-job instances, 10 of the 60 configurations are non-dominated. Results are coherent with the exhaustive study of Chapter 6, with a cluster of configurations having a good hypervolume (low $1 - \text{HV}$) but a poor Δ' spread, and configurations with a much better distribution but a much poorer hypervolume. The Pareto front is well separated between the two types of configurations, the first using the `ndom` exploration strategy, while the second uses the `imp` exploration strategy.

Very interestingly, a single of the 60 configurations achieves both a very good hypervolume and a very good spread, using a combination of the `all` selection strategy, the `imp` exploration strategy, and the `restart` perturbation strategy. This is surprising first because first it has no close neighbour in the objective space, hinting that the performance of the configuration is more due to the particular combination of parameters, rather than only its strategy components, and second because this particular region of the objective space was not reached by any of the final configurations of Chapter 6.

On the 20-job instances, 20 of the 60 configurations are non-dominated. The

Table 8.3 – Optimal configurations ($K = 1$; PFSP 20 jobs)

$1 - HV$	Δ'	Selection	Exploration	Perturbation
0.5322	0.8464	`random`	`imp-ndom`	`restart`
0.5324	0.8450	`random`	`ndom`	`restart`
0.5338	0.8486	`random`	`ndom`	`kick`
0.5340	0.8414	`older`	`ndom`	`kick`
0.5346	0.8297	`random`	`ndom`	`kick-all`
0.5368	0.8253	`older`	`all`	`restart`
0.5373	0.8172	`older`	`all`	`kick`
0.5390	0.8161	`older`	`all`	`kick-all`
0.5403	0.7983	`newest`	`ndom`	`kick`
0.5409	0.7076	`all`	`all-imp`	`restart`
0.5414	0.6759	`all`	`imp`	`restart`
0.5422	0.6515	`random`	`all-imp`	`restart`
0.5430	0.6164	`random`	`imp`	`restart`
0.5495	0.6056	`all`	`all-imp`	`kick-all`
0.5514	0.5495	`all`	`imp`	`kick-all`
0.5519	0.5429	`all`	`imp`	`kick`
0.5588	0.5376	`older`	`imp`	`kick`
0.5592	0.5291	`older`	`imp`	`kick-all`
0.5598	0.4931	`random`	`all-imp`	`kick-all`
0.5613	0.4743	`random`	`all-imp`	`kick`

Table 8.4 – Optimal configurations (K = 1; PFSP 50 jobs)

$1 - HV$	Δ'	Selection	Exploration	Perturbation
0.4848	0.9706	`random`	`ndom`	`kick-all`
0.4850	0.9608	`random`	`ndom`	`kick`
0.4853	0.9568	`all`	`ndom`	`restart`
0.4854	0.9345	`older`	`ndom`	`kick`
0.4856	0.9060	`older`	`ndom`	`kick-all`
0.4889	0.5719	`all`	`imp`	`restart`
0.5023	0.5151	`all`	`imp`	`kick`
0.5024	0.5063	`all`	`imp`	`kick-all`
0.5036	0.4975	`random`	`imp`	`restart`
0.5114	0.4862	`random`	`imp`	`kick-all`

configurations seem less clustered, and the configurations better distributed along the Pareto front. The optimal configurations are much more diverse than on the 50-job instances, and include strategies such as the `newest` selection strategy, the `all`, `all-imp`, and `imp-ndom` exploration strategies. For both sizes of instances, there is no clear consensus on the perturbation strategy.

8.3.2 K = 2 Configuration Schedules

Figure 8.3 shows the average performance of the configuration schedules resulting from the test step of MO-ParamILS, on the 10 Taillard instances, when schedules using two configurations (K = 2) are available (left: instances with 20 jobs; right: instances with 50 jobs). These configurations are separated according to the length k of their schedule, and are described in Table 8.5 and Table 8.6. To facilitate the analysis, the 60 configurations of the exhaustive enumeration are also shown for both sizes of instances.

On 50-job instances, the AAC approaches only resulted in schedules using two successive configurations. Conversely, on 20-job instances, half of the Pareto front, corresponding to the schedules that achieves better hypervolume, use a single configuration, while the other half use two successive configurations. The schedules that use two successive configurations are slightly more efficient than the schedules using a single configuration on the 20-job instances. On the 50-job instances, save from the singular configuration that achieves both a very good hypervolume

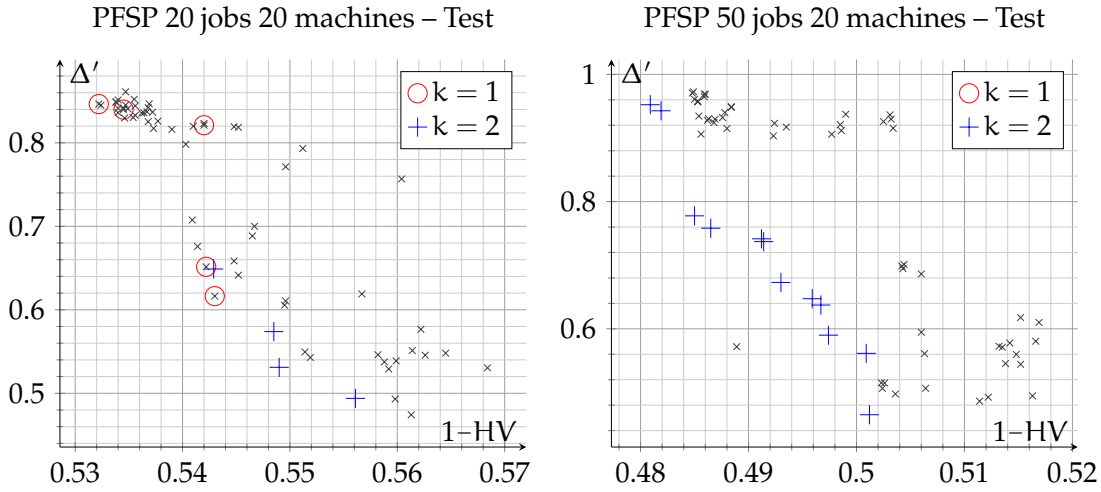


Figure 8.3 – Final optimised configuration schedules (K = 2; left: 20 jobs; right: 50 jobs)

Table 8.5 – Final optimised configuration schedules (K = 2; PFSP 20 jobs)

$1-HV$	Δ'	k	T_i	Selection	Exploration	Perturbation
0.5322	0.8424	1	–	`random`	`imp-ndom`	`restart`
0.5345	0.8398	1	–	`older`	`imp-ndom`	`restart`
0.542	0.8233	1	–	`newest`	`imp-ndom`	`restart`
0.542	0.8211	1	–	`newest`	`ndom`	`restart`
0.5422	0.6515	1	–	`random`	`all-imp`	`restart`
0.5429	0.6489	2	1/4	`newest`	`imp`	`kick`
			3/4	`random`	`imp`	`restart`
0.543	0.6164	1	–	`random`	`imp`	`restart`
0.5485	0.5739	2	1/4	`newest`	`all-imp`	`restart`
			3/4	`all`	`all-imp`	`kick`
0.549	0.5311	2	1/4	`older`	`all-imp`	`restart`
			3/4	`random`	`all-imp`	`kick`
0.5561	0.4939	2	3/4	`all`	`imp`	`kick-all`
			1/4	`older`	`imp`	`kick-all`

Table 8.6 – Final optimised configuration schedules ($K = 2$; PFSP 50 jobs)

$1 - HV$	Δ'	k	T_i	Selection	Exploration	Perturbation
0.4809	0.9523	2	1/2	`older`	`ndom`	`restart`
			1/2	`random`	`all`	`kick-all`
0.4819	0.9428	2	3/4	`older`	`imp-ndom`	`restart`
			1/4	`all`	`ndom`	`restart`
0.4850	0.7776	2	3/4	`older`	`ndom`	`restart`
			1/4	`all`	`all-imp`	`restart`
0.4865	0.7582	2	1/2	`older`	`ndom`	`restart`
			1/2	`all`	`imp`	`kick`
0.4912	0.7415	2	3/4	`all`	`ndom`	`kick-all`
			1/4	`random`	`all-imp`	`restart`
0.4914	0.7371	2	3/4	`all`	`ndom`	`kick-all`
			1/4	`random`	`all-imp`	`kick-all`
0.4930	0.6729	2	3/4	`newest`	`ndom`	`restart`
			1/4	`random`	`all-imp`	`kick`
0.4959	0.6473	2	3/4	`newest`	`ndom`	`kick`
			1/4	`random`	`all-imp`	`kick`
0.4967	0.6374	2	1/4	`newest`	`ndom`	`kick-all`
			3/4	`all`	`imp`	`kick`
0.4974	0.5900	2	1/2	`all`	`all`	`kick-all`
			1/2	`all`	`imp`	`kick-all`
0.5009	0.5614	2	1/2	`newest`	`ndom`	`kick-all`
			1/2	`random`	`all-imp`	`kick-all`
0.5012	0.4649	2	3/4	`all`	`imp`	`restart`
			1/4	`random`	`all-imp`	`kick-all`

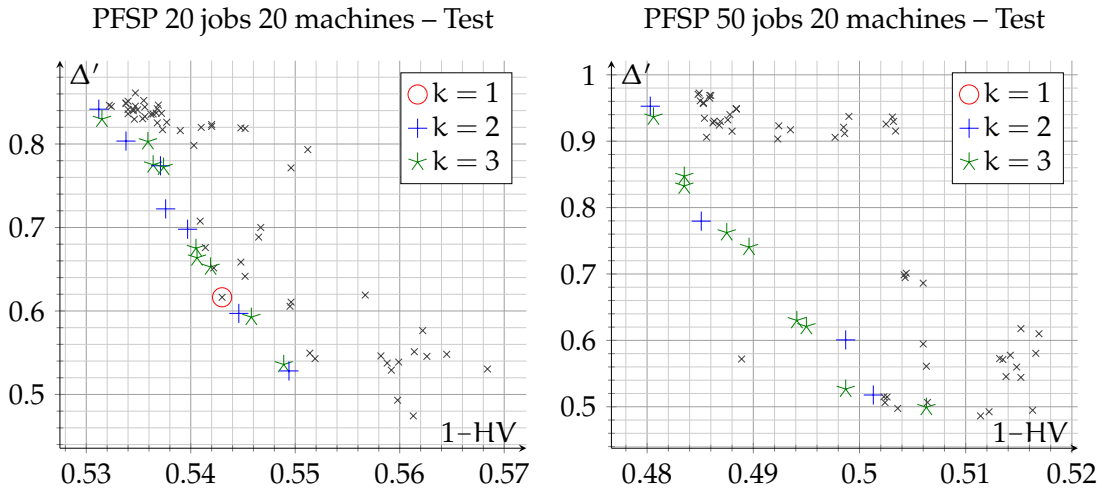


Figure 8.4 – Final optimised configuration schedules (K = 3; left: 20 jobs; right: 50 jobs)

and a very good spread, the 59 other schedules composed by a single configuration are all dominated by the final schedules. Unfortunately, no general trend arises from the time budget decomposition used.

On 50-job instances, a large number of the configuration schedules are able to achieve very good compromises between hypervolume and Δ' spread, that were not achieved by any of the configurations of Chapter 6.

Regarding the strategies selected in the final schedules, the only sensible observation is that the `restart` perturbation strategy seems slightly favoured on 20-job instances, while `kick-all` is slightly favoured on 50-job instances. An explanation would be that 20-job instances are easy enough so that few *lucky* individual MOLS runs can reach optimal solutions, implying that always restarting is a viable strategy. This does not hold for larger instances.

8.3.3 K = 3 Configuration Schedules

Similarly, Figure 8.4 shows the average performance of the configuration schedules resulting from the test step of MO-ParamILS, on the 10 Taillard instances, when schedules using three successive configurations (K = 3) are available (left: instances with 20 jobs; right: instances with 50 jobs). These configurations are separated according to the length k of their schedule, and are described in Table 8.7 and Table 8.8. Again, to facilitate the analysis, the 60 configurations of the exhaustive enumeration are also shown for both sizes of instances.

On the 20-job instances, a single of the seventeen final schedules uses a single

Table 8.7 – Final optimised configuration schedules ($K = 3$; PFSP 20 jobs)

$1 - HV$	Δ'	k	T_i	Selection	Exploration	Perturbation
0.5312	0.8417	2	3/4	random	all	restart
			1/4	random	ndom	restart
0.5315	0.8298	3	1/2	random	all	restart
			1/4	all	ndom	kick
			1/4	random	all	restart
0.5338	0.8035	2	3/4	older	ndom	restart
			1/4	random	all-imp	restart
0.5359	0.8029	3	1/2	older	imp-ndom	restart
			1/4	random	imp	restart
			1/4	all	all-imp	restart
0.5364	0.7751	3	1/3	newest	ndom	kick-all
			1/3	older	ndom	restart
			1/3	all	imp	restart
0.5371	0.7739	2	1/4	older	ndom	restart
			3/4	all	imp	restart
0.5374	0.7723	3	1/2	random	imp-ndom	kick-all
			1/4	older	all	restart
			1/4	all	imp	restart
0.5376	0.7223	2	3/4	newest	imp-ndom	restart
			1/4	random	all-imp	restart
0.5397	0.6979	2	3/4	all	imp	restart
			1/4	random	all-imp	restart
0.5405	0.6751	3	1/2	older	imp	restart
			1/4	newest	imp	restart
			1/4	random	all-imp	restart
0.5406	0.6637	3	1/2	newest	imp	restart
			1/4	random	all-imp	kick-all
			1/4	random	imp	restart
0.5419	0.6527	3	1/2	older	imp	kick-all
			1/4	newest	imp	restart
			1/4	random	all-imp	restart

Table 8.7 – Final optimised configuration schedules (K = 3; PFSP 20 jobs; continued)

1 − HV	Δ′	k	T_i	Selection	Exploration	Perturbation
0.543	0.6164	1	—	`random`	`imp`	`restart`
0.5446	0.5971	2	3/4	`newest`	`imp`	`restart`
			1/4	`random`	`imp`	`kick-all`
0.5458	0.5926	3	1/2	`older`	`all-imp`	`restart`
			1/4	`newest`	`imp`	`kick`
			1/4	`random`	`imp`	`kick`
0.5489	0.5359	3	1/2	`random`	`imp`	`kick`
			1/4	`newest`	`imp`	`restart`
			1/4	`all`	`imp`	`kick`
0.5494	0.5282	2	1/4	`newest`	`all-imp`	`restart`
			3/4	`random`	`imp`	`kick-all`

configuration. The other schedules, together with the schedules of the 50-job instances, use two or three configurations while very slightly favouring using three configurations. In comparison to Figure 8.4, this time every of the 60 schedules composed by a single configuration are dominated by at least one of the final schedules. Unfortunately, as previously, no general trend arises from the time budget decomposition used.

Again, on 50-job instances, a large number of the configuration schedules are able to achieve very good compromises between hypervolume and Δ′ spread, that were not achieved by the configurations of Chapter 6.

Finally, the perturbations strategies of the final schedules are coherent to the previous ones: the `restart` strategies is clearly favoured on 20-job instances, while the schedules on the 50-jobs instances often use a combination of both the `restart` and the `kick-all` perturbation strategies.

8.4 Discussions

Figure 8.5 compares the three sets of non-dominated configurations from the exhaustively enumerated search space (K = 1) and optimised (K = 2 and K = 3) configuration schedules.

Table 8.8 – Final optimised configuration schedules ($K = 3$; PFSP 50 jobs)

$1 - HV$	Δ'	k	T_i	Selection	Exploration	Perturbation
0.4803	0.9527	2	3/4	older	ndom	restart
			1/4	all	ndom	kick-all
0.4806	0.9366	3	1/2	older	ndom	restart
			1/4	random	imp-ndom	kick
			1/4	random	ndom	kick
0.4835	0.8326	3	1/2	older	ndom	restart
			1/4	all	ndom	kick-all
			1/4	all	all-imp	restart
0.4851	0.7796	2	3/4	older	ndom	restart
			1/4	all	imp	kick-all
0.4875	0.7623	3	1/2	older	ndom	restart
			1/4	random	all-imp	kick
			1/4	all	all-imp	restart
0.4896	0.7403	3	1/2	older	imp-ndom	restart
			1/4	random	imp	kick-all
			1/4	random	all-imp	restart
0.4941	0.63	3	1/2	newest	imp-ndom	restart
			1/4	older	all-imp	kick-all
			1/4	random	all-imp	restart
0.495	0.621	3	1/2	newest	imp-ndom	restart
			1/4	random	imp	kick
			1/4	random	all-imp	kick-all
0.4987	0.5266	3	1/2	all	imp	restart
			1/4	all	all-imp	kick
			1/4	random	imp	restart
0.5013	0.5177	2	1/4	newest	all-imp	restart
			3/4	random	all-imp	restart
0.5063	0.4992	3	1/2	newest	all-imp	restart
			1/4	random	imp	kick-all
			1/4	random	imp	kick

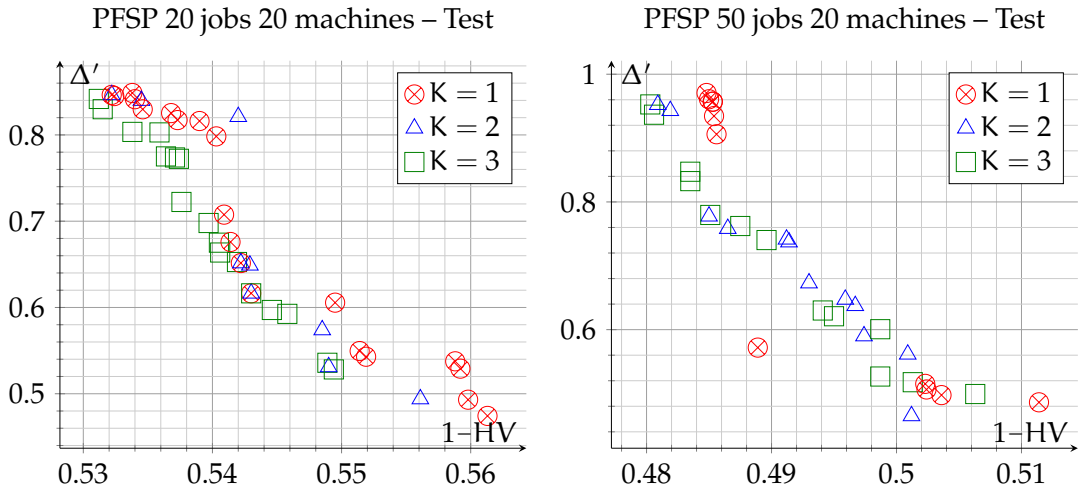


Figure 8.5 – Final comparison

On 20-job instances, while the enumeration of the 60 configurations resulted in a very well-distributed Pareto front, the search of $K = 2$ configuration schedules was only very slightly beneficial, giving very similar performing configurations schedules. On the contrary, with $K = 3$ many new configuration schedules outperform the previously found ones.

On 50-jobs instances, the enumeration of the 60 considered configurations resulted in the two clusters of configurations is coherent with the observations of Chapter 6, and a single, very surprising, configuration achieving a very good compromise between hypervolume and Δ' spread, that previously investigated configurations were unable to achieve. Both $K = 2$ and $K = 3$ configuration schedules achieve similar looking Pareto fronts, with numerous schedules with good compromises between hypervolume and Δ' spread.

In conclusion, on both datasets, the use of configuration schedules led to many new algorithms that outperformed most if not all of the initial configurations they are constituted from. However, the major drawback of this approach is the combinatorial explosion of the number of possible configuration schedules.

Nonetheless, this chapter only presented preliminary results, that did not took this drawback into account but yet sufficed to show the potential of this approach. Future works on this topic should, within other perspectives, focus on alleviate this combinatorial explosion by, for example, better consider the structure of the search space by iteratively constructing the schedules and pruning the search space of uninteresting strategies.

8.5 Perspectives

In this chapter we continued to focus on one of the drawback of offline algorithm design (Part III), which is that it predicts a single configuration whose parameters values stay fixed during the execution.

More specifically, we investigated using offline algorithm design techniques on schedules of configurations. We showed that even statically determined schedules of configurations were able to easily reach better performance and compromises between convergence and distribution that were not found in the experiments of Chapter 6.

In the following, we detail two perspectives that are related to this chapter.

Schedule-independent parameters. The size of the configuration space of the schedule grows exponentially with the number of sub-configurations considered. As such, only very few parameter were considered to be controlled by the schedule, while all the other were manually fixed. A continuation of our work would be to include these parameters, independently, in addition to the schedule parameters, so they can also be automatically optimised by the configuration process.

Dynamic time budgets. Furthermore, during the experiments of Chapter 8 we used a very small number of possible time schedules, that were manually fixed. These schedule, along with the number of configurations composing the schedule, can also be automatically determined. While a too precise description of the time budget would induce far too many possible schedules, it would nevertheless be very interesting to allow more diversity in how the budget is divided.

General Conclusion

> Not all those who wander are lost.
>
> *The Fellowship of the Ring*
> J.R.R. Tolkien

Automatic algorithm design (AAD) is a recent but thriving research field. It has reached a point where efficient tools became easily usable and actually used in practice. Multi-objective optimisation is one of the most recent direction toward which AAD is developing, with renewed interest from both communities. We strongly believe that this direction of work will highly benefit designers of single-objective algorithms, multi-objective algorithms, and automatic design tools altogether, by further enabling even better raw performance and understanding of algorithmic components.

Contribution Summary

In this thesis, we investigated the multi-objective automatic design of a particular class of multi-objective metaheuristics, the multi-objective local search (MOLS) algorithms. In the following, we summarise our main contributions.

Automatic algorithm design (AAD). Within many others, algorithm selection, algorithm configuration, parameter tuning, parameter control, hyper-heuristics, reactive and autonomous search, are research fields that focus on systematically optimising the search process, as opposed to solely optimising the search results.

We proposed a new taxonomy to unify these research fields under the larger field of automatic algorithm design. Our taxonomy is based on two general viewpoints. First, a temporal viewpoint, according to which approaches are already divided between *offline* approaches that are optimised prior to their execution and *online* approaches that are adapted during their execution Then, a structural viewpoints, according to which approaches are classified based on the

algorithmic structure of the elements being optimised. We also discussed a complementary complexity viewpoint, related to the available available knowledge sources.

Multi-objective local search (MOLS) algorithms. Local search algorithms are first and foremost known for being very effective on single objective problems. Often being introduced either as direct multi-objective extensions of known single-objective local search algorithms, as part of bigger evolutionary algorithms, or as independent original multi-objective algorithms, many multi-objective approaches use local search techniques. However, these approaches are less well-known and studied than their single-objective counterparts.

We first conducted a chronological survey of the use of local search techniques on multi-objective problems, and discussed their characteristics, detailing for each local search component the existing strategies found in the literature. From this analysis followed a new unification proposition, building on existing unifications but further generalising to other single-trajectory multi-objective local search algorithms.

MO-ParamILS. ParamILS is an efficient and very well known algorithm configuration framework. However, similarly to the other available configuration tools of the literature (e.g., irace, SMAC, GGA, GGA++), it is only able to consider a single performance metric.

In order to overcome this limitation, we formally defined multi-objective automatic algorithm configuration, discussed its use cases, and more importantly we presented MO-ParamILS, our extension of ParamILS for multiple performance indicators. We validated the efficiency of MO-ParamILS by comparing its two variants MO-BasicILS and MO-FocusedILS against a baseline using ParamILS, on various classical optimisation scenarios extended to two performance indicators. In particular, we used MO-ParamILS to configure both the final solution quality and the running time of CPLEX, a well known commercial mixed integer programming optimiser, and to configure both the memory usage and running time of SAT-solver CLASP. In all five scenarios we considered, similarly as for ParamILS for which FocusedILS generally outperforms BasicILS, MO-FocusedILS outperformed MO-BasicILS as well as the baseline.

MOLS automatic configuration. Performance assessment of multi-objective algorithms is traditionally conducted using multiple complementary multi-objective performance indicators. While multi-objective algorithms can be optimised by classical single-objective configurators according to a single metric, one of the main

goal of introducing multi-objective algorithm configuration and proposing MO-ParamILS was specifically to take advantage of multiple performance indicators during the optimisation process.

We investigated this use case in depth by considering the automatic configuration of a MOLS algorithm with regard to two complementary performance traits: convergence and distribution of the final solutions. We considered three classical bi-objective permutation-based problems: the permutation flowshop scheduling problem (PFSP), the travelling salesman problem (TSP), and the quadratic assignment problem (QAP). The MOLS algorithm considered is a classical, static, and highly parameterised MOLS algorithm that exposes many possible combinations of strategies found in the literature. We first exhaustively investigated a subset of the design space of the MOLS algorithm to analyse it directly, then automatically configured it first on literature instances, then on specially crafted instance on which correlation between objectives is hand-tuned.

Regarding MOLS algorithms in particular, we highlighted the impact of individual strategies on convergence and distribution and computed optimised combinations of strategies of the various considered scenarios. Regarding configuration of multi-objective algorithms in general, we showed that multi-objective approaches (such as MO-ParamILS) were more suitable than hybrid approaches using classical single-objective configurators (such as ParamILS). These observations have been validated across the various scenarios involving different sizes of instances as well as different degree of correlation between objectives.

One of key point of *classical* automatic algorithm configuration is that the configurator tool predict parameter values that stay fixed during the execution of the algorithm. We investigated two extensions of algorithm configuration, for which preliminaries are presented in the last two chapters of this thesis, that focus on algorithms that are able to combine multiple strategies along the execution: first parameter control, then configuration scheduling.

MOLS parameter control. The first extension was to consider control mechanisms that are able to repeatedly switch between several alternative strategies during the execution of the algorithm.

We discussed the integration of generic control mechanisms inside our static MOLS algorithm, proposed an *adaptive* MOLS algorithm in which the exploration strategy is adapted between each iteration of an iterated local search scheme according to its performance. After reviewing the simplest control mechanisms that we could easily integrate in our adaptive algorithm, we selected and investigated the impact of two of them: a uniform random control mechanism and a

ε-greedy mechanism. We conducted three successive studies: first using a set of three possible exploration strategies, then a set of the two most efficient exploration strategies, and finally an hybrid approach of *long-term learning* in which the third and least effective strategy is identified and removed during the search process.

As result, we showed that it was possible to take advantage of several strategies without loss of performance. Even using the very simplest control mechanisms, we were able to effectively match the performances of a static approach solely using the best exploration strategy, even if sub-optimal strategies were considered at the start of the search process.

MOLS configuration scheduling. Finally, the second extension was to consider multiple MOLS configurations, that are sequentially used with regard to a static schedule. This approach enables modifications of the strategies and parameter values during the execution of the algorithm, which is classically not possible. Configurations schedules can also be optimised using standard automatic algorithm configuration tools, thus conserving one of the main advantages of static algorithms.

We formally defined configuration scheduling, and investigated schedules of MOLS configurations on the PFSP. Specifically, we analysed the automatic configuration of schedules of two and three configurations, using MO-ParamILS as the automatic configurator. We showed very promising results, with many schedules easily outperforming static MOLS variants while simultaneously achieving much better and diverse compromises between convergence and distribution.

Future Research

Many perspectives arose from the different contributions of this thesis, were detailed at the end of their respective chapters. In the following, we focus on the two perspectives that we consider to be the most natural and believe to have the most potential.

Anytime behaviour of algorithms. The key feature of multi-objective automatic algorithm configuration is to be able to investigate trade-offs between multiple performance indicators. One of the most interesting use case is the anytime behaviour of algorithms: the trade-off between solution quality and running time, i.e., the questions of how long should the algorithm run with regard to the expected final quality, and whether the algorithm can or not provide good intermediary solutions before the end of the execution and the final output.

There are many ways in which this problematic can be encoded in terms of a multi-objective scenario, that can lead to many different ways to analyse the anytime behaviour of algorithms. Some have been proposed in this thesis (see Chapter 5), some were only outlined as perspectives (e.g., using the *area under the curve*), and others are yet to be devised. We strongly believe that MO-ParamILS and other multi-objective configurators will in the future bring valuable insight to many algorithms.

Artificial configuration spaces. The biggest disadvantage of automatic configurators is the very large amount of computation they can require in order to optimise their prediction of the best configuration. Indeed, if many configurations are available, many runs are necessary to have a reliable estimation of their performance, and the total configuration time is directly function of the running time of the target algorithm.

It follows that optimising the design of automatic configurators, such as for example ParamILS and MO-ParamILS, is a very slow and manual procedure, at least orders of magnitude longer than the base configuration process. A problem can also arise that the configurator may have been over tuned for a particular class of configuration scenario. In general, it is very difficult to analyse the performance of configurators.

To enable quick and fruitful comparisons of configuration processes and configuration protocols, it is essential that the time taken by the configuration process becomes reasonably small. We believe that the most promising solution would be to create a specific target algorithm that would be able to run instantaneously and accurately model distinct configuration scenarios. It would be obtained by thoroughly analysing existing target algorithms with the goal of obtaining stand-alone reusable models.

Publications

The following papers have been, in chronological order, submitted, accepted, and have or will be presented in international conferences during this thesis:

- **Blot, A.**, Hoos, H. H., Jourdan, L., Kessaci-Marmion, M., and Trautmann, H. (2016). **MO-ParamILS: A multi-objective automatic algorithm configuration framework.** In Festa, P., Sellmann, M., and Vanschoren, J., editors (2016). *Learning and Intelligent Optimization – 10th International Conference, LION 10. Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*, pages 32–47. Springer.
- **Blot, A.**, Pernet, A., Jourdan, L., Kessaci-Marmion, M., and Hoos, H. H. (2017c). **Automatically configuring multi-objective local search using multi-objective optimisation.** In Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M. M., Jin, Y., and Grimme, C., editors (2017). *Evolutionary Multi-Criterion Optimization – 9th International Conference, EMO 2017. Proceedings*, volume 10173 of *Lecture Notes in Computer Science*, pages 61–76. Springer.
- **Blot, A.**, Kessaci-Marmion, M., and Jourdan, L. (2017b). **AMH: a new framework to design adaptive metaheuristics.** In *12th Metaheuristics International Conference, MIC 2017. Proceedings*, pages 586–588.
- **Blot, A.**, Jourdan, L., and Kessaci-Marmion, M. (2017a). **Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem.** In Bosman, P. A. N., editor (2017). *Genetic and Evolutionary Computation Conference, GECCO 2017. Proceedings*, pages 227–234. ACM.
- **Blot, A.**, Kessaci, M., and Jourdan, L. (2018b). **Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation.** *Journal of Heuristics*.
- **Blot, A.**, Kessaci, M., Jourdan, L., and Causmaecker, P. D. (2018c). **Adaptive multi-objective local search algorithms for the permutation flowshop scheduling problem.** In Pardalos, P. and Kotsireas, I., editors, *Learning and Intelligent Optimization – 12th International Conference, LION 12. Revised Selected Papers, Lecture Notes in Computer Science*. Springer. (To appear).

- **Blot, A.**, López-Ibáñez, M. Kessaci, M., and Jourdan, L. (2018d). **New initialisation techniques for multi-objective local search application on the bi-objective permutation flowshop.** In Auger, A., Fonseca, C. M., Lourenço, N., Machado, P., Paquete, L., and Whitley, D., editors, *Parallel Problem Solving from Nature – 15th International Conference, PPSN XV. Proceedings, Part I*, volume 11101 of *Lecture Notes in Computer Science*. Springer. (To appear).
- **Blot, A.**, Hoos, H. H., Kessaci, M., and Jourdan, L. (2018a). **Automatic configuration of multi-objective optimization algorithms. impact of correlation between objectives.** In *30th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2018.* IEEE Computer Society. (To appear).

Additionally, the following papers also have been submitted to international journals and conferences, and are either currently under review or revision:

- Pageau, C., **Blot, A.**, Kessaci-Marmion, M., Jourdan, L., and Hoos H. H.. **Automatic design of a dynamic multi-objective local search algorithm.**
- **Blot, A.**, Kessaci-Marmion, M., Jourdan, L., and Hoos H. H.. **Automatic configuration of multi-objective local search algorithms for permutation problems.**

Bibliography

This book was written using 100% recycled words.

Wyrd Sisters
Terry Pratchet

Abbasi, M., Paquete, L., and Pereira, F. B. (2015). Local search for multiobjective multiple sequence alignment. In Guzman, F. M. O. and Rojas, I., editors, *Bioinformatics and Biomedical Engineering – Third International Conference, IWBBIO 2015. Proceedings, Part II*, volume 9044 of *Lecture Notes in Computer Science*, pages 175–182. Springer. (citation on page 43)

Adenso-Díaz, B. and Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114. (citation on page 29)

Aguirre, H. E. and Tanaka, K. (2005). Random bit climbers on multiobjective mnk-landscapes: Effects of memory and population climbing. *IEICE Transactions*, 88-A(1):334–345. (citations on pages 47, 50, 61, and 62)

Aleti, A. and Moser, I. (2011). Predictive parameter control. In Krasnogor and Lanzi (2011), pages 561–568. (citations on pages 30 and 155)

Aleti, A. and Moser, I. (2013). Entropy-based adaptive range parameter control for evolutionary algorithms. In Blum, C. and Alba, E., editors, *Genetic and Evolutionary Computation Conference, GECCO 2013. Proceedings*, pages 1501–1508. ACM. (citation on page 155)

Aleti, A. and Moser, I. (2016). A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49(3):56:1–56:35. (citations on pages 29, 151, and 155)

Aleti, A., Moser, I., Meedeniya, I., and Grunske, L. (2014). Choosing the appropriate forecasting model for predictive parameter control. *Evolutionary Computation*, 22(2):319–349. (citation on page 155)

Aleti, A., Moser, I., and Mostaghim, S. (2012). Adaptive range parameter control. In *IEEE Congress on Evolutionary Computation, CEC 2012. Proceedings*, pages 1–8. IEEE. (citation on page 155)

Amadini, R., Gabbrielli, M., and Mauro, J. (2014). SUNNY: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, 14(4-5):509–524. (citations on pages 27 and 76)

Angel, E., Bampis, E., and Gourvès, L. (2004). Approximating the Pareto curve with local search for the bicriteria TSP(1, 2) problem. *Theoretical Computer Science*, 310(1-3):135–146. (citations on pages 47, 50, 61, and 62)

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., and Tierney, K. (2015). Model-based genetic algorithms for algorithm configuration. In Yang, Q. and Wooldridge, M., editors, *Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015. Proceedings*, pages 733–739. AAAI Press. (citation on page 29)

Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. P., editor, *Principles and Practice of Constraint Programming – 15th International Conference, CP 2009. Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer. (citation on page 29)

Arroyo, J. E. C., dos Santos Ottoni, R., and de Paiva Oliveira, A. (2011). Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. *Electronic Notes in Theoretical Computer Science*, 281:5–19. (citation on page 45)

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256. (citation on page 154)

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Aguilera, M. J. B., Blum, C., Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics – 4th International Workshop, HM 2007. Proceedings*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer. (citation on page 28)

Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12(3):269–283. (citation on page 44)

Bartz-Beielstein, T., Branke, J., Filipic, B., and Smith, J., editors (2014). *Parallel Problem Solving from Nature – 13th International Conference, PPSN XIII. Proceedings*, volume 8672 of *Lecture Notes in Computer Science*. Springer. (citations on pages 190 and 196)

Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation, CEC 2005. Proceedings*, pages 773–780. IEEE. (citation on page 29)

Bartz-Beielstein, T. and Markon, S. (2004). Tuning search algorithms for real-world applications: a regression tree based approach. In *IEEE Congress on Evolutionary Computation, CEC 2004. Proceedings*, pages 1111–1118. IEEE. (citation on page 29)

Basseur, M. and Burke, E. K. (2007). Indicator-based multi-objective local search. In *IEEE Congress on Evolutionary Computation, CEC 2007. Proceedings*, pages 3100–3107. IEEE. (citations on pages 12, 43, 48, 50, 61, and 62)

Basseur, M., Zeng, R., and Hao, J. (2012). Hypervolume-based multi-objective local search. *Neural Computing and Applications*, 21(8):1917–1929. (citation on page 48)

Battiti, R., Brunato, M., and Mascia, F. (2008). *Reactive Search and Intelligent Optimization*. Springer Publishing Company, Incorporated, 1st edition. (citation on page 31)

Baykasoglu, A., Owen, S., and Gindy, N. (1999). A taboo search based approach to find the Pareto optimal set in multiple objective optimization. *Engineering Optimization*, 31(6):731–748. (citation on page 44)

Beausoleil, R. P. (2001). Multiple criteria scatter search. In *4th Metaheuristics International Conference*, pages 534–539. (citation on page 44)

Belluz, J., Gaudesi, M., Squillero, G., and Tonda, A. P. (2015). Operator selection using improved dynamic multi-armed bandit. In Silva and Esparcia-Alcázar (2015), pages 1311–1317. (citation on page 154)

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305. (citation on page 28)

Bezerra, L. C. T., López-Ibáñez, M., and Stützle, T. (2014). Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In Bartz-Beielstein et al. (2014), pages 508–517. (citation on page 19)

Biedenkapp, A., Lindauer, M. T., Eggensperger, K., Hutter, F., Fawcett, C., and Hoos, H. H. (2017). Efficient parameter importance analysis via ablation with surrogates. In Singh, S. P. and Markovitch, S., editors, *Thirty-First AAAI Conference on Artificial Intelligence. Proceedings*, pages 773–779. AAAI Press. (citation on page 146)

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B., Cantú-Paz, E., Mathias, K. E., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V. G., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. K., and Jonoska, N., editors, *Genetic and Evolutionary Computation Conference, GECCO 2002. Proceedings*, pages 11–18. Morgan Kaufmann. (citation on page 28)

Blot, A., Aguirre, H. E., Dhaenens, C., Jourdan, L., Marmion, M., and Tanaka, K. (2015). Neutral but a winner! How neutrality helps multiobjective local search algorithms. In Gaspar-Cunha, A., Antunes, C. H., and Coello, C. A. C., editors, *Evolutionary Multi-Criterion Optimization – 8th International Conference, EMO 2015. Proceedings, Part I*, volume 9018 of *Lecture Notes in Computer Science*, pages 34–47. Springer. (citations on pages 51, 54, and 101)

Blot, A., Hoos, H. H., Jourdan, L., Kessaci-Marmion, M., and Trautmann, H. (2016). MO-ParamILS: A multi-objective automatic algorithm configuration framework. In Festa et al. (2016), pages 32–47. (citations on pages 32, 87, and 185)

Blot, A., Hoos, H. H., Kessaci, M., and Jourdan, L. (2018a). Automatic configuration of multi-objective optimization algorithms. impact of correlation between objectives. In *30th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2018*. IEEE Computer Society. (To appear). (citations on pages 114 and 186)

Blot, A., Jourdan, L., and Kessaci-Marmion, M. (2017a). Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem. In Bosman (2017), pages 227–234. (citations on pages 56, 60, 114, and 185)

Blot, A., Kessaci, M., and Jourdan, L. (2018b). Survey and unification of local

search techniques in metaheuristics for multi-objective combinatorial optimisation. *Journal of Heuristics*. (To appear). (citations on pages 41 and 185)

Blot, A., Kessaci, M., Jourdan, L., and Causmaecker, P. D. (2018c). Adaptive multi-objective local search algorithms for the permutation flowshop scheduling problem. In Pardalos, P. and Kotsireas, I., editors, *Learning and Intelligent Optimization – 12th International Conference, LION 12. Revised Selected Papers*, volume to appear of *Lecture Notes in Computer Science*. Springer. (To appear). (citations on pages 150 and 185)

Blot, A., Kessaci-Marmion, M., and Jourdan, L. (2017b). AMH: a new framework to design adaptive metaheuristics. In *12th Metaheuristics International Conference, MIC 2017. Proceedings*. (citations on pages 64 and 185)

Blot, A., López-Ibáñez, M., Kessaci, M., and Jourdan, L. (2018d). Archive-aware scalarisation-based multi-objective local search for a bi-objective permutation flowshop problem. In Auger, A., Fonseca, C. M., Lourenço, N., Machado, P., Paquete, L., and Whitley, D., editors, *Parallel Problem Solving from Nature – 15th International Conference, PPSN XV. Proceedings, Part I*, volume 11101 of *Lecture Notes in Computer Science*. Springer. (citation on page 186)

Blot, A., Pernet, A., Jourdan, L., Kessaci-Marmion, M., and Hoos, H. H. (2017c). Automatically configuring multi-objective local search using multi-objective optimisation. In Trautmann et al. (2017), pages 61–76. (citations on pages 60, 114, and 185)

Bosman, P. A. N., editor (2017). *Genetic and Evolutionary Computation Conference, GECCO 2017. Proceedings*. ACM. (citations on pages 190 and 192)

Burke, E. K., Gendreau, M., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724. (citation on page 30)

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In Gendreau and Potvin (2010), pages 449–468. (citations on pages 30, 31, 37, and 38)

Cesa-Bianchi, N. and Fischer, P. (1998). Finite-time regret bounds for the multiarmed bandit problem. In Shavlik, J. W., editor, *Fifteenth International Conference on Machine Learning, ICML 1998. Proceedings*, pages 100–108. Morgan Kaufmann. (citation on page 153)

Coello, C. A. C., editor (2011). *Learning and Intelligent Optimization – 5th International Conference, LION 5. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*. Springer. (citations on pages 196 and 205)

Coello, C. A. C. and Cortés, N. C. (2005). Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190. (citation on page 15)

Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A., et al. (2007). *Evolutionary algorithms for solving multi-objective problems*. Springer, 2nd edition. (citation on page 13)

Costa, L. D., Fialho, Á., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In Ryan, C. and Keijzer, M., editors, *Genetic and Evolutionary Computation Conference, GECCO 2008. Proceedings*, pages 913–920. ACM. (citations on pages 30 and 154)

Cowling, P. I., Kendall, G., and Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In Burke, E. K. and Erben, W., editors, *Practice and Theory of Automated Timetabling III – Third International Conference, PATAT 2000. Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer. (citation on page 30)

Czyzak, P. and Jaszkiewicz, A. (1996). A multiobjective metaheuristic approach to the location of petrol stations by the capital budgeting model. *Control and Cybernetics*, 25:177–187. (citations on pages 14, 43, 44, 50, 54, 56, 61, and 62)

Czyzak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47. (citations on pages 44 and 53)

Dang, N., Cáceres, L. P., Causmaecker, P. D., and Stützle, T. (2017). Configuring irace using surrogate configuration benchmarks. In Bosman (2017), pages 243–250. (citation on page 145)

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons. (citations on pages 13 and 56)

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Guervós, J. J. M., and Schwefel, H., editors, *Parallel Problem Solving from Nature – 6th International*

Conference, PPSN VI. Proceedings, volume 1917 of *Lecture Notes in Computer Science*, pages 849–858. Springer. (citation on page 13)

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197. (citations on pages 13, 15, and 17)

di Tollo, G., Lardeux, F., Maturana, J., and Saubion, F. (2015). An experimental study of adaptive control for evolutionary algorithms. *Applied Soft Computing*, 35:359–372. (citations on pages 29, 68, and 151)

Doerr, B. and Doerr, C. (2015). Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In Silva and Esparcia-Alcázar (2015), pages 1335–1342. (citation on page 29)

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41. (citation on page 13)

Drugan, M. M. and Nowé, A. (2013). Designing multi-objective multi-armed bandits algorithms: A study. In *International Joint Conference on Neural Networks, IJCNN 2013*, pages 1–8. IEEE. (citation on page 154)

Drugan, M. M. and Thierens, D. (2010). Path-guided mutation for stochastic Pareto local search algorithms. In Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors, *Parallel Problem Solving from Nature – 11th International Conference, PPSN XI. Proceedings, Part I*, volume 6238 of *Lecture Notes in Computer Science*, pages 485–495. Springer. (citation on page 59)

Drugan, M. M. and Thierens, D. (2012). Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18(5):727–766. (citations on pages 48, 50, 56, 59, 61, and 62)

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011a). Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In Krasnogor and Lanzi (2011), pages 2019–2026. (citation on page 32)

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011b). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236. (citations on pages 18, 48, 116, and 151)

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011c). Improving the anytime behavior of two-phase local search. *Annals of Mathematics and Artificial Intelligence*, 61(2):125–154. (citation on page 19)

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2012). Pareto local search algorithms for anytime bi-objective optimization. In Hao, J. and Middendorf, M., editors, *Evolutionary Computation in Combinatorial Optimization – 12th European Conference, EvoCOP 2012. Proceedings*, volume 7245 of *Lecture Notes in Computer Science*, pages 206–217. Springer. (citations on pages 49 and 56)

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2015). Anytime Pareto local search. *European Journal of Operational Research*, 243(2):369–385. (citations on pages 19, 43, 49, and 51)

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141. (citations on pages 25, 29, 31, 32, 33, 37, and 151)

Eiben, A. E., Horváth, M., Kowalczyk, W., and Schut, M. C. (2006). Reinforcement learning for online control of evolutionary algorithms. In Brueckner, S., Hassas, S., Jelasity, M., and Yamins, D., editors, *Engineering Self-Organising Systems – 4th International Workshop, ESOA 2006. Revised and Invited Papers*, volume 4335 of *Lecture Notes in Computer Science*, pages 151–160. Springer. (citations on pages 30 and 155)

Eiben, A. E., Michalewicz, Z., Schoenauer, M., and Smith, J. E. (2007). Parameter control in evolutionary algorithms. In Lobo, F. G., Lima, C. F., and Michalewicz, Z., editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer. (citations on pages 29 and 151)

Eiben, A. E. and Smit, S. K. (2012). Evolutionary algorithm parameters and methods to tune them. In Hamadi et al. (2012), pages 15–36. (citation on page 25)

Engrand, P. (1998). A multi-objective optimization approach based on simulated annealing and its application to nuclear fuel management. Technical report, Électricité de France. (citation on page 44)

Feo, T. A., Resende, M. G. C., and Smith, S. H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878. (citation on page 44)

Festa, P., Sellmann, M., and Vanschoren, J., editors (2016). *Learning and Intelligent Optimization – 10th International Conference, LION 10. Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*. Springer. (citations on pages 190 and 200)

Fialho, Á., Costa, L. D., Schoenauer, M., and Sebag, M. (2010). Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1-2):25–64. (citation on page 154)

Fonseca, C. M., Fleming, P. J., Zitzler, E., Deb, K., and Thiele, L., editors (2003). *Evolutionary Multi-Criterion Optimization – Second International Conference, EMO 2003. Proceedings*, volume 2632 of *Lecture Notes in Computer Science*. Springer. (citations on pages 198 and 202)

Gai, Y., Krishnamachari, B., and Jain, R. (2012). Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478. (citation on page 154)

Geiger, M. J. (2008). Randomised variable neighbourhood search for multi objective optimisation. In *Design and Evaluation of Advanced Hybrid Meta-Heuristics – 4th EU/ME Workshop. Proceedings*, pages 34–42. (citations on pages 45, 50, 61, 62, and 98)

Gendreau, M. and Potvin, J.-Y., editors (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2nd edition. (citations on pages 191 and 200)

Glover, F. (1989). Tabu search – part I. *ORSA Journal on computing*, 1(3):190–206. (citations on pages 42 and 44)

Glover, F. W. and Laguna, M. (1997). *Tabu Search*. Springer US. (citations on pages 13, 42, and 44)

Gretsista, A. and Burke, E. K. (2017). An iterated local search framework with adaptive operator selection for nurse rostering. In Battiti, R., Kvasov, D. E., and Sergeyev, Y. D., editors, *Learning and Intelligent Optimization – 11th International Conference, LION 11. Revised Selected Papers*, volume 10556 of *Lecture Notes in Computer Science*, pages 93–108. Springer. (citation on page 156)

Hamadi, Y., Monfroy, E., and Saubion, F., editors (2012). *Autonomous Search*. Springer. (citations on pages 25, 31, 32, 34, and 194)

Hansen, M. P. (1997). Tabu search for multiobjective optimization: MOTS. In Stewart, T., editor, *Multiple Criteria Decision Making – 13th International Conference. Proceedings*, pages 574–586. Springer. (citations on pages 14, 43, 44, 50, 53, 54, 56, 61, and 62)

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press. (citation on page 13)

Hoos, H., Lindauer, M. T., and Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, 14(4-5):569–585. (citations on pages 27 and 35)

Hoos, H. H. (2012). Programming by optimization. *Communications of the ACM*, 55(2):70–80. (citation on page 24)

Hoos, H. H., Kaminski, R., Lindauer, M. T., and Schaub, T. (2015). aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming*, 15(1):117–142. (citations on pages 27 and 76)

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann. (citations on pages 14, 31, 43, and 89)

Horn, D., Schork, K., and Wagner, T. (2016). Multi-objective selection of algorithm portfolios: Experimental validation. In Handl, J., Hart, E., Lewis, P. R., López-Ibáñez, M., Ochoa, G., and Paechter, B., editors, *Parallel Problem Solving from Nature – 14th International Conference, PPSN XIV. Proceedings*, volume 9921 of *Lecture Notes in Computer Science*, pages 421–430. Springer. (citation on page 32)

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello (2011), pages 507–523. (citations on pages 29, 35, 98, and 111)

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Identifying key algorithm parameters and instance features using forward selection. In Nicosia and Pardalos (2013), pages 364–381. (citation on page 146)

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306. (citations on pages 29, 89, 94, 105, 110, 123, 145, and 146)

Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Twenty-Second AAAI Conference on Artificial Intelligence. Proceedings*, pages 1152–1157. AAAI Press. (citations on pages 29, 89, and 94)

Inja, M., Kooijman, C., de Waard, M., Roijers, D. M., and Whiteson, S. (2014). Queued Pareto local search for multi-objective optimization. In Bartz-Beielstein et al. (2014), pages 589–599. (citation on page 49)

Ishibuchi, H. and Murata, T. (1996). Multi-objective genetic local search algorithm. In *Evolutionary Computation, IEEE International Conference. Proceedings*, pages 119–124. IEEE. (citations on pages 14, 43, 46, 50, 61, and 62)

Ishibuchi, H. and Murata, T. (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. (citation on page 46)

Ishibuchi, H., Tsukamoto, N., and Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. In *IEEE Congress on Evolutionary Computation, CEC 2008. Proceedings*, pages 2419–2426. IEEE. (citation on page 43)

Jaeggi, D., Asselin-Miller, C., Parks, G., Kipouros, T., Bell, T., and Clarkson, J. (2004). Multi-objective parallel tabu search. In Yao et al. (2004), pages 732–741. (citation on page 44)

Jaeggi, D., Parks, G. T., Kipouros, T., and Clarkson, P. J. (2008). The development of a multi-objective tabu search algorithm for continuous optimisation problems. *European Journal of Operational Research*, 185(3):1192–1212. (citation on page 44)

Jaszkiewicz, A. (2002). Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, 137(1):50–71. (citations on pages 43 and 46)

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Algorithm selection and scheduling. In Lee, J. H., editor, *Principles and Practice of Constraint Programming – 17th International Conference, CP 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 454–469. Springer. (citation on page 27)

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. (2010). ISAC – instance-specific algorithm configuration. In Coelho, H., Studer, R., and Wooldridge, M., editors, *19th European Conference on Artificial Intelligence, ECAI 2010. Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press. (citation on page 27)

Karafotias, G., Eiben, Á. E., and Hoogendoorn, M. (2014). Generic parameter control with reinforcement learning. In Arnold, D. V., editor, *Genetic and Evolutionary Computation Conference, GECCO 2014. Proceedings*, pages 1319–1326. ACM. (citations on pages 70 and 155)

Karafotias, G., Hoogendoorn, M., and Eiben, Á. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187. (citations on pages 26, 29, and 151)

Karafotias, G., Smit, S. K., and Eiben, A. E. (2012). A generic approach to parameter control. In Chio, C. D., Agapitos, A., Cagnoni, S., Cotta, C., de Vega, F. F., Caro, G. A. D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Langdon, W. B., Guervós, J. J. M., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A., Togelius, J., Urquhart, N., Uyar, S., and Yannakakis, G. N., editors, *Applications of Evolutionary Computation - EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC. Proceedings*, volume 7248 of *Lecture Notes in Computer Science*, pages 366–375. Springer. (citation on page 68)

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *IEEE International Conference on Neural Networks. Proceedings*, volume 4, pages 1941–1948. (citation on page 13)

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307. (citation on page 21)

Kessaci-Marmion, M., Dhaenens, C., and Humeau, J. (2017). Neutral neighbors in bi-objective optimization: Distribution of the most promising for permutation problems. In Trautmann et al. (2017), pages 344–358. (citations on pages 19, 20, 22, and 132)

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680. (citations on pages 42 and 43)

Knowles, J., Thiele, L., and Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland. revised version. (citation on page 106)

Knowles, J. D. and Corne, D. (2003). Instance generators and test suites for the multiobjective quadratic assignment problem. In Fonseca et al. (2003), pages 295–310. (citation on page 22)

Knowles, J. D. and Corne, D. W. (1999). The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation. In *IEEE Congress on Evolutionary Computation, CEC 99. Proceedings*, pages 98–105. IEEE. (citations on pages 14, 43, 46, 47, 50, 53, 61, and 62)

Knowles, J. D. and Corne, D. W. (2000a). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172. (citations on pages 46, 50, 61, and 62)

Knowles, J. D. and Corne, D. W. (2000b). M-PAES: A memetic algorithm for multiobjective optimization. In *IEEE Congress on Evolutionary Computation, CEC 00. Proceedings*, pages 325–332. IEEE. (citation on page 46)

Knowles, J. D. and Corne, D. W. (2002). On metrics for comparing nondominated sets. In *IEEE Congress on Evolutionary Computation, CEC 2002. Proceedings*, volume 1, pages 711–716. IEEE. (citation on page 14)

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In Bessiere, C., Raedt, L. D., Kotthoff, L., Nijssen, S., O'Sullivan, B., and Pedreschi, D., editors, *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 149–190. Springer. (citation on page 27)

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5. (citation on page 35)

Krasnogor, N. and Lanzi, P. L., editors (2011). *Genetic and Evolutionary Computation Conference, GECCO 2011. Proceedings*. ACM. (citations on pages 187 and 193)

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22. (citation on page 154)

Langdon, W. B. (2015). Genetically improved software. In Gandomi, A. H., Alavi, A. H., and Ryan, C., editors, *Handbook of Genetic Programming Applications*, pages 181–220. Springer. (citation on page 32)

Lawler, E. L., Lenstra, J. K., Kan, A. H. R., and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522. (citation on page 18)

Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., and Talbi, E. (2012). On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18(2):317–352.
(citations on pages 43, 48, 50, 51, 52, 53, 60, 61, 62, and 64)

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516. (citation on page 21)

Lindauer, M., Bergdoll, R., and Hutter, F. (2016). An empirical study of per-instance algorithm scheduling. In Festa et al. (2016), pages 253–259. (citations on pages 27 and 76)

Lindauer, M. T., Hoos, H. H., Hutter, F., and Schaub, T. (2015). AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778. (citation on page 35)

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58. (citations on pages 28, 98, and 111)

López-Ibáñez, M. and Stützle, T. (2010a). Automatic configuration of multi-objective ACO algorithms. In Dorigo, M., Birattari, M., Caro, G. A. D., Doursat, R., Engelbrecht, A. P., Floreano, D., Gambardella, L. M., Groß, R., Sahin, E., Sayama, H., and Stützle, T., editors, *Swarm Intelligence – 7th International Conference, ANTS 2010. Proceedings*, volume 6234 of *Lecture Notes in Computer Science*, pages 95–106. Springer. (citation on page 13)

López-Ibáñez, M. and Stützle, T. (2010b). The impact of design choices of multiobjective antcolony optimization algorithms on performance: an experimental study on the biobjective TSP. In Pelikan, M. and Branke, J., editors, *Genetic and Evolutionary Computation Conference, GECCO 2010. Proceedings*, pages 71–78. ACM. (citation on page 13)

Lourenço, H., Martin, O., and Stützle, T. (2010). Iterated local search: Framework and applications. In Gendreau and Potvin (2010), chapter 9, pages 363–397. (citation on page 42)

Lourenço, H., Martin, O., and Stützle, T. (2003). Iterated local search. In Glover, F. W. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Springer. (citations on pages 59 and 89)

Lust, T. and Teghem, J. (2010). Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510. (citation on page 48)

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2013). Algorithm portfolios based on cost-sensitive hierarchical clustering. In Rossi, F., editor, *Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013. Proceedings*, pages 608–614. IJCAI/AAAI. (citation on page 27)

Mansour, I. B. and Alaya, I. (2015). Indicator based ant colony optimization for multi-objective knapsack problem. *Procedia Computer Science*, 60:448–457. (citation on page 12)

Martí, R., Campos, V., Resende, M. G. C., and Duarte, A. (2015). Multiobjective GRASP with path relinking. *European Journal of Operational Research*, 240(1):54–71. (citation on page 45)

Maturana, J., Fialho, Á., Saubion, F., Schoenauer, M., and Sebag, M. (2009). Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *IEEE Congress on Evolutionary Computation, CEC 2009. Proceedings*, pages 365–372. IEEE. (citations on pages 30, 70, and 154)

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100. (citations on pages 42 and 45)

Moalic, L., Caminada, A., and Lamrous, S. (2013). A fast local search approach for multiobjective problems. In Nicosia and Pardalos (2013), pages 294–298. (citations on pages 48, 50, 61, and 62)

Moffaert, K. V., Drugan, M. M., and Nowé, A. (2013). Hypervolume-based multi-objective reinforcement learning. In Purshouse, R. C., Fleming, P. J., Fonseca, C. M., Greco, S., and Shaw, J., editors, *Evolutionary Multi-Criterion Optimization - 7th International Conference, EMO 2013. Proceedings*, volume 7811 of *Lecture Notes in Computer Science*, pages 352–366. Springer. (citation on page 70)

Molina, J., Laguna, M., Martí, R., and Caballero, R. (2007). SSPMO: A scatter tabu search procedure for non-linear multiobjective optimization. *INFORMS Journal on Computing*, 19(1):91–100. (citation on page 44)

Moslehi, G. and Mahnam, M. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics*, 129(1):14–22. (citation on page 47)

Murata, T., Ishibuchi, H., and Gen, M. (2000). Cellular genetic local search for multi-objective optimization. In Whitley, L. D., Goldberg, D. E., Cantú-Paz, E.,

Spector, L., Parmee, I. C., and Beyer, H., editors, *Genetic and Evolutionary Computation Conference, GECCO '00. Proceedings*, pages 307–314. Morgan Kaufmann. (citation on page 46)

Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95. (citation on page 151)

Nicosia, G. and Pardalos, P. M., editors (2013). *Learning and Intelligent Optimization – 7th International Conference, LION 7. Revised Selected Papers*, volume 7997 of *Lecture Notes in Computer Science*. Springer. (citations on pages 196 and 201)

Okabe, T., Jin, Y., and Sendhoff, B. (2003). A critical survey of performance indices for multi-objective optimisation. In *IEEE Congress on Evolutionary Computation, CEC 2003. Proceedings*, pages 878–885. IEEE. (citation on page 14)

Paquete, L., Chiarandini, M., and Stützle, T. (2004). Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., editors, *Metaheuristics for multiobjective optimisation*, volume 535, pages 177–199. Springer Science & Business Media. (citations on pages 47, 50, 53, 61, and 62)

Paquete, L. and Stützle, T. (2003). A two-phase local search for the biobjective traveling salesman problem. In Fonseca et al. (2003), pages 479–493. (citation on page 47)

Pareto, V. (1896). *Cours d'économie politique*, volume 1. F. Rouge. (citation on page 11)

Petke, J., Haraldsson, S. O., Harman, M., Langdon, W. B., White, D. R., and Woodward, J. R. (2017). Genetic improvement of software: a comprehensive survey. *IEEE Transactions on Evolutionary Computation*. (citation on page 32)

Rajaraman, K. and Sastry, P. S. (1996). Finite time analysis of the pursuit algorithm for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(4):590–598. (citation on page 153)

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier. (citation on page 26)

Riquelme, N., von Lücken, C., and Barán, B. (2015). Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference, CLEI 2015*, pages 1–11. IEEE. (citation on page 14)

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049. (citation on page 116)

Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2015). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, 45(2):217–228. (citation on page 154)

Sakurai, Y., Takada, K., Kawabe, T., and Tsuruta, S. (2010). A method to control parameters of evolutionary algorithms by using reinforcement learning. In Yétongnon, K., Dipanda, A., and Chbeir, R., editors, *Sixth International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2010*, pages 74–79. IEEE Computer Society. (citation on page 155)

Schumer, M. and Steiglitz, K. (1968). Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276. (citation on page 30)

Serafini, P. (1994). Simulated annealing for multi objective optimization problems. In *Multiple Criteria Decision Making*, pages 283–292. Springer. (citations on pages 14, 43, 50, 54, 56, 61, and 62)

Silva, S. and Esparcia-Alcázar, A. I., editors (2015). *Genetic and Evolutionary Computation Conference, GECCO 2015. Proceedings*. ACM. (citations on pages 189, 193, and 206)

Suman, B. (2003). Simulated annealing-based multiobjective algorithms and their application for system reliability. *Engineering Optimization*, 35(4):391–416. (citation on page 44)

Suman, B. and Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160. (citation on page 44)

Suppapitnarm, A. and Parks, G. (1999). Simulated annealing: an alternative approach to true multiobjective optimization. In Banzhaf, W., Daida, J. M., Eiben, A. E., Garzon, M. H., Honavar, V. G., Jakiela, M. J., and Smith, R. E., editors, *Genetic and Evolutionary Computation Conference, GECCO 1999. Proceedings*, pages 406–407. Morgan Kaufmann. (citation on page 44)

Suresh, R. K. and Mohanasundaram, K. M. (2004). Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives. In *IEEE*

Conference on Cybernetics and Intelligent Systems, CIS 2004. Proceedings, volume 2, pages 712–717. IEEE. (citation on page 44)

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.
(citations on pages 30, 153, 154, and 155)

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285. (citation on page 19)

Talbi, E., Rahoual, M., Mabed, M. H., and Dhaenens, C. (2001). A hybrid evolutionary approach for multicriteria optimization problems: Application to the flow shop. In Zitzler, E., Deb, K., Thiele, L., Coello, C. A. C., and Corne, D., editors, *Evolutionary Multi-Criterion Optimization – 1st International Conference, EMO 2001. Proceedings*, pages 416–428. Springer.
(citations on pages 14, 43, 45, 46, 47, 50, 61, and 62)

Thathachar, M. and Sastry, P. S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):168–175. (citation on page 153)

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In Beyer, H. and O'Reilly, U., editors, *Genetic and Evolutionary Computation Conference, GECCO 2005. Proceedings*, pages 1539–1546. ACM.
(citations on pages 30, 152, and 153)

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In Dhillon, I. S., Koren, Y., Ghani, R., Senator, T. E., Bradley, P., Parekh, R., He, J., Grossman, R. L., and Uthurusamy, R., editors, *Knowledge Discovery and Data Mining – 19th ACM SIGKDD International Conference, KDD 2013*, pages 847–855. ACM. (citations on pages 35 and 146)

Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M. M., Jin, Y., and Grimme, C., editors (2017). *Evolutionary Multi-Criterion Optimization – 9th International Conference, EMO 2017. Proceedings*, volume 10173 of *Lecture Notes in Computer Science*. Springer. (citations on pages 191 and 198)

Tricoire, F. (2012). Multi-directional local search. *Computers & Operations Research*, 39(12):3089–3101. (citation on page 49)

Ulungu, B., Fortemps, P., and Teghem, J. (1995). Heuristic for multi-objective combinatorial optimization problems by simulated annealing. In Gu, J., Chen, G.,

Wei, Q., and Wang, S., editors, *MCDM: Theory and Applications 1995*, pages 229–238. Sci-Tech. (citations on pages 14, 43, 50, 54, 56, 61, and 62)

Ulungu, B., Teghem, J., Fortemps, P., and Tuyttens, D. (1999). MOSA method: a tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8(4):221. (citations on pages 43 and 53)

van Veldhuizen, D. A. and Lamont, G. B. (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147. (citation on page 14)

Veerapen, N. and Saubion, F. (2011). Pareto autonomous local search. In Coello (2011), pages 392–406. (citations on pages 72 and 156)

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In Gama, J., Camacho, R., Brazdil, P., Jorge, A., and Torgo, L., editors, *16th European Conference on Machine Learning, ECML 2005. Proceedings*, volume 3720 of *Lecture Notes in Computer Science*, pages 437–448. Springer. (citation on page 153)

Vianna, D. S. and Arroyo, J. E. C. (2004). A GRASP algorithm for the multiobjective knapsack problem. In *XXIV International Conference of the Chilean Computer Science Society, SCCC 2004. Proceedings*, pages 69–75. IEEE Computer Society. (citation on page 44)

Whiteson, S. and Stone, P. (2006). On-line evolutionary computation for reinforcement learning in stochastic domains. In Cattolico, M., editor, *Genetic and Evolutionary Computation Conference, GECCO 2006. Proceedings*, pages 1577–1584. ACM. (citation on page 155)

Wong, Y.-Y., Lee, K.-H., Leung, K.-S., and Ho, C.-W. (2003). A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing*, 7(8):506–515. (citations on pages 30 and 155)

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606. (citation on page 27)

Yahyaa, S. Q., Drugan, M. M., and Manderick, B. (2014). Annealing-Pareto multi-objective multi-armed bandit algorithm. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. ADPRL 2014*, pages 1–8. IEEE. (citation on page 154)

Yao, X., Burke, E. K., Lozano, J. A., Smith, J., Guervós, J. J. M., Bullinaria, J. A., Rowe, J. E., Tiño, P., Kabán, A., and Schwefel, H., editors (2004). *Parallel Problem Solving from Nature – 8th International Conference, PPSN VIII. Proceedings*, volume 3242 of *Lecture Notes in Computer Science*. Springer.
(citations on pages 197 and 206)

Zhang, Q. and Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731. (citation on page 13)

Zhang, T., Georgiopoulos, M., and Anagnostopoulos, G. C. (2015). SPRINT multi-objective model racing. In Silva and Esparcia-Alcázar (2015), pages 1383–1390.
(citations on pages 32, 98, and 111)

Zhang, T., Georgiopoulos, M., and Anagnostopoulos, G. C. (2016). Multi-objective model selection via racing. *IEEE Transactions on Cybernetics*, 46(8):1863–1876.
(citations on pages 32, 98, and 111)

Zhang, T., Georgiopoulos, M., and Anagnostopoulos, G. C. (2018). Pareto-optimal model selection via SPRINT-Race. *IEEE Transactions on Cybernetics*, 48(2):596–610. (citations on pages 32, 98, and 111)

Zitzler, E. and Künzli, S. (2004). Indicator-based selection in multiobjective search. In Yao et al. (2004), pages 832–842. (citations on pages 12 and 13)

Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-report*, 103. (citation on page 13)

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms - A comparative case study. In Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H., editors, *Parallel Problem Solving from Nature – 5th International Conference, PPSN V. Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–304. Springer. (citation on page 15)

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271. (citations on pages 12, 13, 14, 15, and 48)

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.
(citations on pages 14, 15, and 122)

Title. Reacting and Adapting to the Environment – Designing Autonomous Methods for Multi-Objective Combinatorial Optimisation

Keywords. automatic algorithm design, multi-objective optimisation, combinatorial optimisation, metaheuristics, local search algorithms

Abstract. Large-scale optimisation problems are usually hard to solve optimally. Approximation algorithms such as metaheuristics, able to quickly find sub-optimal solutions, are often preferred. This thesis focuses on multi-objective local search (MOLS) algorithms, metaheuristics able to deal with the simultaneous optimisation of multiple criteria. As many algorithms, metaheuristics expose many parameters that significantly impact their performance. These parameters can be either predicted and set before the execution of the algorithm, or dynamically modified during the execution itself.

While in the last decade many advances have been made on the automatic design of algorithms, the great majority of them only deal with single-objective algorithms and the optimisation of a single performance indicator such as the algorithm running time or the final solution quality. In this thesis, we investigate the relations between automatic algorithm design and multi-objective optimisation, with an application on MOLS algorithms.

We first review possible MOLS strategies ans parameters and present a general, highly configurable, MOLS framework. We also propose MO-ParamILS, an automatic configurator specifically designed to deal with multiple performance indicators. Then, we conduct several studies on the automatic offline design of MOLS algorithms on multiple combinatorial bi-objective problems. Finally, we discuss two online extensions of classical algorithm configuration: first the integration of parameter control mechanisms, to benefit from having multiple configuration predictions; then the use of configuration schedules, to sequentially use multiple configurations.

Titre : Réagir et s'adapter à son environnement – Concevoir des méthodes autonomes pour l'optimisation combinatoire à plusieurs objectifs

Mots-clés : design automatique d'algorithmes, optimisation multi-critères, optimisation combinatoire, métaheuristiques, algorithmes de recherche locale

Résumé : Les problèmes d'optimisation à grande échelle sont généralement difficiles à résoudre de façon optimale. Des algorithmes d'approximation tels que les métaheuristiques, capables de trouver rapidement des solutions sous-optimales, sont souvent préférés. Cette thèse porte sur les algorithmes de recherche locale multi-objectif (MOLS), des métaheuristiques capables de traiter l'optimisation simultanée de plusieurs critères. Comme de nombreux algorithmes, les MOLS exposent de nombreux paramètres qui ont un impact important sur leurs performances. Ces paramètres peuvent être soit prédits et définis avant l'exécution de l'algorithme, soit ensuite modifiés dynamiquement.

Alors que de nombreux progrès ont récemment été réalisés pour la conception automatique d'algorithmes, la grande majorité d'entre eux ne traitent que d'algorithmes mono-objectif et l'optimisation d'un unique indicateur de performance. Dans cette thèse, nous étudions les relations entre la conception automatique d'algorithmes et l'optimisation multi-objective.

Nous passons d'abord en revue les stratégies MOLS possibles et présentons un framework MOLS général et hautement configurable. Nous proposons également MO-ParamILS, un configurateur automatique spécialement conçu pour gérer plusieurs indicateurs de performance. Nous menons ensuite plusieurs études sur la conception automatique de MOLS sur de multiples problèmes combinatoires bi-objectifs. Enfin, nous discutons deux extensions de la configuration d'algorithme classique : d'abord l'intégration des mécanismes de contrôle de paramètres, pour bénéficier de multiples prédictions de configuration ; puis l'utilisation séquentielle de plusieurs configurations.