Chapter 1
Automatic Configuration of Multi-objective Optimizers and Multi-objective Configuration

Leonardo C. T. Bezerra[1], Manuel López-Ibáñez[2], and Thomas Stützle[3]

Abstract

Heuristic optimizers are an important tool in academia and industry, and their performance-optimizing configuration requires a significant amount of expertise. As the proper configuration of algorithms is a crucial aspect in the engineering of heuristic algorithms, a significant research effort has been dedicated over the last years towards moving this step to the computer and, thus, make it automatic. These research efforts go way beyond *tuning* only numerical parameters of already fully defined algorithms, but exploit automatic configuration as a means for automatic algorithm design.

In this chapter, we review two main aspects where the research on automatic configuration and multi-objective optimization intersect. The first is the automatic configuration of multi-objective optimizers, where we discuss means and specific approaches. In addition, we detail a case study that shows how these approaches can be used to design new, high-performing multi-objective evolutionary algorithms. The second aspect is the research on multi-objective configuration, that is, the possibility of using multiple performance metrics for the evaluation of algorithm configurations. We highlight some few examples in this direction.

Key words: multi-objective optimization, automatic algorithm configuration, automatic configuration of multi-objective optimizers, automatic configuration considering multiple objectives

Instituto Metrópole Digital (IMD), Universidade Federal do Rio Grande do Norte (UFRN), Natal, RN, Brazil (leobezerra@imd.ufrn.br)
· Alliance Manchester Business School, University of Manchester, UK (manuel.lopez-ibanez@manchester.ac.uk)
· IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium (stuetzle@ulb.ac.be).

1.1 Introduction

Automatic algorithm configuration has shown to be a useful technique to relieve algorithm designers from tedious tasks in the tuning of different classes of algorithms, in particular stochastic local search (SLS) methods. Several software packages for configuring algorithms have been proposed in the literature, and among the most widely used ones we find ParamILS (Hutter et al, 2009), SMAC (Hutter et al, 2011), and irace (López-Ibáñez et al, 2016). The applications of these advanced automatic algorithm configuration tools are not limited to calibrating numerical parameters of already fully developed algorithms. When combined with flexible algorithmic frameworks, these tools can be used for design space exploration and for the generation of algorithms that have never been proposed previously in the literature. In fact, many algorithm packages for integer programming can be seen as such algorithmic frameworks where, by configuring a few parameters, specific routines or heuristics can be switched on or off, thus obtaining potentially new, previously unexplored algorithms. In a similar way, algorithmic frameworks for heuristic optimization can also benefit from this coupling with automatic configuration techniques. In recent research efforts, configurable unified frameworks have been proposed for classical decision problems, such as satisfiability (KhudaBukhsh et al, 2009), for optimization problems using metaheuristics (Liao et al, 2014; López-Ibáñez and Stützle, 2012; Dubois-Lacoste et al, 2011; Bezerra et al, 2016), and for machine learning tasks (Feurer et al, 2015; Thornton et al, 2013; Mendoza et al, 2016). In all cases, automatically configured algorithms obtained from these frameworks were shown to be competitive and often superior to the various algorithms from which the components for these algorithmic frameworks have been taken. In particular, metaheuristic approaches have been further extended to allow the composition of completely new, hybrid algorithms, which can be derived from a simple recursive framework (Marmion et al, 2013).

Automatic algorithm configuration tools have mainly been applied to configure single-objective algorithms, but they can also be used to automatically generate high-performance algorithms for tackling multi-objective optimization problems—when talking of multi-objective optimization here, we consider such problems in the most general sense, that is, trying to approximate the Pareto front. In fact, the authors of this chapter have dedicated a significant amount of their research towards (i) the development of the methodologies and tools for performing an efficient configuration of multi-objective optimizers; (ii) the generation of flexible algorithmic frameworks from which effective multi-objective optimizers can be generated, and; (iii) the elaboration of case studies that show the advantages of the proposed methodologies. The first work in this direction showed how to generate automatically multi-objective ant colony optimization (ACO) algorithms from a flexible algorithm framework (López-Ibáñez and Stützle, 2010b) and then later ex-

tended (López-Ibáñez and Stützle, 2012).[1] The configuration methodology consisted essentially in the performance analysis of the configurations through unary performance indicators such as the hypervolume, and the optimization of these indicators through off-the-shelf automatic algorithm configuration tools (see López-Ibáñez and Stützle (2012) for details). This methodology has also been applied to configure other multi-objective optimizers based on the two-phase plus Pareto local search framework by Dubois-Lacoste et al (2011), and to design multi-objective evolutionary algorithms, a major extension of this approach. This latter work was based on a new conceptual view of MOEA components that allows instantiating, from the same algorithmic template, a larger number of MOEAs from the literature than existing MOEA frameworks, and has led to substantially improved, automatically designed algorithms (Bezerra et al, 2016).

Alternatively, algorithm configuration can itself be interpreted as a multi-objective problem, where the aim is to produce a set of parameter configurations that are mutually nondominated with respect to multiple criteria (Dréo, 2009). From this point of view, a set of configurations should be obtained that builds a trade-off between different aspects of algorithm performance. Various automatic configuration methods have also been extended in that direction, the first one being the extension of the racing methods underlying the irace package (López-Ibáñez et al, 2016) to a multi-objective racing (Zhang et al, 2013).

In this chapter we review these two streams of research, namely (i) applying automatic algorithm configuration techniques to design multi-objective optimizers, and (ii) the search for algorithm configurations under multiple configuration objectives, focusing mainly on the former stream. The chapter is structured as follows. In the next section, we discuss some background on automatic algorithm configuration and then in Section 1.3, we discuss details on the configuration of multi-objective optimizers. Section 1.4 exemplifies the approach and possible results of automatically configuring multi-objective optimizers using the design of multi-objective evolutionary algorithms. Finally, in Section 1.5, we review various approaches that consider the configuration of algorithms from a multi-objective perspective and we conclude in Section 1.6.

1.2 Automatic algorithm configuration

The algorithm configuration task is concerned with the search for performance optimizing parameter settings of a parameterized algorithm. More formally, the task can be defined as follows. Let $\mathcal{A}$ be a parameterized algorithm with a set of n parameters $\Phi_{\mathcal{A}} = \{\phi_i\}, i = 1, \ldots, n$, each parameter

[1] In a different stream of research, Wessing et al (2010) had also applied tuning methods to tune the variation operator of a multi-objective evolutionary algorithm applied to a single problem instance.

having a domain $\mathcal{D}_{\phi_i}$. The *configuration space* $\Theta_{\mathcal{A}}$ of $\mathcal{A}$ is given by the cross-product $D_{\phi_1} \times D_{\phi_2} \times \ldots \times D_{\phi_n}$ of all parameter domains. A *configuration* $\theta_{\mathcal{A}} = \langle \theta_{\phi_i} \rangle \in \Theta_{\mathcal{A}}$ is a tuple comprising one value $\theta_{\phi_i} \in D_{\phi_i}$ for each parameter $\phi_i \in \Phi_{\mathcal{A}}$.

The goal of algorithm configuration is to find performance optimizing parameter settings w.r.t. some distribution of problem instances. More formally, given a specific distribution p_π of instances from a given problem Π, an algorithm $\mathcal{A}$ with a set of parameters $\Phi_{\mathcal{A}}$ and configuration space $\Theta_{\mathcal{A}}$, find the configuration $\theta_{\mathcal{A}} \in \Theta_{\mathcal{A}}$ that optimizes a given performance metric $\hat{c}(\mathcal{A}, p_\pi, \theta_{\mathcal{A}})$ – running $\mathcal{A}$ on p_π using configuration $\theta_{\mathcal{A}}$. In practice, the search of performance optimizing parameter settings is done on a set of training instances that are generated following some underlying distribution p_π.

The parameters of an algorithm are often of two main types: *numerical* and *categorical*. A parameter ϕ_i is *numerical* if its domain is $\mathcal{D}_{\phi_i} \subset \mathbb{R}$ (a *real*-valued parameter) or $\mathcal{D}_{\phi_i} \subset \mathbb{Z}$ (an *integer*-valued parameter). Examples are the temperature parameter in simulated annealing or the population size in evolutionary algorithms. A parameter ϕ_i is *categorical* if its domain $\mathcal{D}_{\phi_i} = \{\nu_1, \ldots, \nu_i\}$, where each ν_j is a discrete option and no ordering relation is defined for $\mathcal{D}_{\phi_i}$. For instance, a categorical parameter could model the set of different recombination operators available for an evolutionary algorithm. Sometimes the discrete values of a categorical parameter may be ordered acording to some criterion such as age or quality; we then speak of an *ordinal* parameter, an example being neighborhood operators for a local search algorithms that entail different neighborhood sizes.

It is also important to distinguish two other concepts, namely parameter *interaction* and *dependency*. Two (or more) parameters interact when the effect of simultaneous changes to these parameters differs from the effects of individually changing them. For instance, crossover and mutation rates in evolutionary algorithms jointly regulate the balance between intensification and diversification. Parameters that interact cannot be configured independently. Regarding dependency, some parameters are only used when specific values for other parameter(s) is (are) selected. For instance, if a clustering technique is needed and k-means is chosen, the parameter k needs to be specified; if another technique is chosen, parameter k does not arise. Parameters such as k are known as *conditional parameters*. In the case of conditional parameters, a valid configuration needs to have assigned values to all non-conditional parameters and to those conditional parameters whose condition is satisfied.

In the literature, the algorithm configuration (AC) task is often referred to also as *tuning task*, probably because in the early literature on parameter optimization fully designed algorithms were considered and often the "tuning" task was related to only setting appropriately the numerical algorithm parameters. However, as explained below, over the years this view has been much extended and algorithm configuration now also considers setting parameters that actually influence the algorithm design; often such parameters are either

categorical or ordinal ones. Because of this extension of scope, we prefer to refer to the task as algorithm configuration rather than tuning.

In many (early) research efforts, the algorithm configuration task has been tackled manually in an often tedious algorithm engineering process. However, given the complex nature of this task, a significant research effort over the last decade or more has been devoted to automate the AC process (Eiben and Smit, 2011; Hoos, 2012a). Given a configuration budget, typically the number of experiments or a maximum runtime, automated algorithm configuration approaches search the configuration space to find high-performing parameter settings considering a training set of problem instances. It is important to highlight here that the problem instances one has at hand are typically divided into a training set and a test set. The training instances are only used during the parameter optimization, while the test set is used to evaluate how generalizable the found parameter set is. The motivation for this separation is the same as in machine learning, where it is well-known that machine learning algorithms may overfit, i.e., be too specific to the previously observed training data and generalize poorly to unseen data. This link between machine learning and automatic algorithm configuration is explicitly described in Birattari (2004).

A number of *automatic algorithm configuration* tools, or *configurators* for short, have been proposed, including iterated F-race (Birattari et al, 2010) and irace (López-Ibáñez et al, 2016), ParamILS (Hutter et al, 2009), GGA (Ansótegui et al, 2009), SMAC (Hutter et al, 2011) or the SPO package (Bartz-Beielstein et al, 2010). An emblematic example is probably the commercial mixed-integer programming solver IBM-ILOG-CPLEX, which ships with an integrated configurator to help end users fine-tune the nearly hundred relevant parameters it presents (IBM, 2017). In fact, this initiative is a result of the significant improvements demonstrated by the automatic configuration community on the runtime required by CPLEX for solving particular problems once properly tuned (Hutter et al, 2010), sometimes surpassing 50-fold speedups over the default settings previously recommended by the CPLEX team. Another direct benefit of the automatic algorithm configuration methodology is encouraging developers to expose parameters that were previously hard-wired into the code, but that can be handled more appropriately by applying automatic configuration to the target domains, as advocated by the proponents of a software design approach known as *programming by optimization* (Hoos, 2012b). In its most advanced version, this design paradigm gives rise to automatic algorithm design, as we will later discuss in this section.

Configurator overview

An in-depth analysis of all proposals for automatic algorithm configuration tools would not fit in this section and escapes the goal of this chapter, and so the reader is referred to Eiben and Smit (2011) and Hoos (2012a) for detailed

overviews of the methods available. Here, we describe the three most widely used configurators.

ParamILS (Hutter et al, 2009) is an iterated local search (ILS) algorithm (Lourenço et al, 2002) that searches in the configuration space. For ParamILS, the quality of a parameter configuration ϕ is given directly by a measure $\hat{c}(\phi)$, such as mean runtime or mean solution quality. Starting from an initial parameter configuration, ParamILS iteratively alters the current incumbent configuration by modifying only one of its parameter values at a time. Whether a modified configuration ϕ' is considered better than the original configuration ϕ is determined in different ways, depending on whether one uses the BasicILS or the FocusedILS variant. In the BasicILS variant, two configurations are compared on the same number of instances. Differently, FocusedILS uses a dominance criterion. In particular, a configuration ϕ' dominates a configuration ϕ if at least as many runs have been executed on ϕ' as on ϕ and it holds that $\hat{c}(\phi') \leq \hat{c}(\phi)$ (considering a measure to be minimized) on the first $n(\phi)$ runs, where $n(\phi)$ is the number of times configuration ϕ has been executed. If dominance of the new configuration cannot be established, additional runs on new instances are performed for both ϕ' and ϕ. Hence, in FocusedILS there is no pre-determined fixed number of instances on which configurations are compared. To prevent getting trapped in local optima, ParamILS uses a *perturbation* procedure by altering simultaneously several parameter values of the incumbent solution. Moreover, a restart mechanism replaces the perturbation with a fixed probability, ensuring that the algorithm explores different regions of the configuration space. The main advantages of ParamILS are a fast convergence due to the used local search-based approach and the proposal of *capping*, i.e., terminating runs that take longer than a given captime when configuring algorithms based on runtime. As drawbacks, ParamILS is only able to handle discrete parameters (real-valued parameters need to be discretized), and capping is usually not effective when the configuration target is solution-quality.

irace (López-Ibáñez et al, 2011, 2016) is a software package that implements iterated racing algorithms for automatic algorithm configuration, among which the earlier proposed I/F-Race (Balaprakash et al, 2007). It is based on an estimation of distribution (EDA) algorithm that encodes its learning as probability distributions that are used to sample configurations and race them. More specifically, a solution in irace comprises a parent configuration and a set of parameter-wise probability distributions. At each iteration, offspring configurations are sampled from the parent configurations and their parameter-wise probability distributions (the better a parent configuration, the higher is on average the number of offsprings generated from it); each sampled offspring configuration inherits the distributions from its parent. These offspring configurations are then tested on a subset of instances by means of *racing*, i.e., comparing solutions on an instance-basis while discarding poor-performing ones, until enough statistical evidence is gathered to determine the subset of configurations that perform best. Once the race is finished, irace

learns: given a surviving solution s, it updates the probability distributions of s to add a bias in favor of the parameter values present in $\phi(s)$, its associated configuration. Effectively, the parameter-wise distributions of s are biased towards the regions of the configuration space where the performance measure of the configuration associated to s ($\hat{c}(\phi(s))$ is optimized. When a new iteration is started, a new set of offspring solutions is sampled based on these updated probability distributions; otherwise, the configurations of the surviving candidates are returned. The major advantages of irace are dealing with all parameter types and allowing on-the-fly processing of the performance of candidates, a useful feature in the context of multi-objective optimization. In addition, a number of applications have demonstrated its effectiveness in comparison to manual tuning. Recent extensions of irace also include a capping mechanism making it competitive also for run-time minimization (Pérez Cáceres et al, 2017).

SMAC (Hutter et al, 2011) uses the idea of sequential model-based optimization (SMBO), in which response-surface models (RSMs, Box and Draper (2007)) are constructed/refined at each iteration. More precisely, SMBO approaches initially sample configurations using a given method and fit an RSM. During consecutive iterations, novel configurations are sampled and evaluated according to the RSM. Selected configurations expected to be high-performing are raced against the best-so-far configuration found. At the beginning of each iteration, the RSM is refined to learn from the performance of the novel configurations. By the end of the iterative process, a high-performing configuration is returned. SMAC extends previous SMBO approaches by allowing (i) different machine learning methods to fit the RSM, but usually using random forests (Breiman, 2001) for the modeling, (ii) the inclusion of categorical parameters, and; (iii) using instance sets instead of a single instance. The main advantages of SMAC are its ability to (i) deal with numerical and categorical parameters, and (ii) explicitly account for instance features and parameter interactions. In addition, SMAC uses racing and has been tested with several different machine learning methods and shown to work well on most scenarios (Hutter et al, 2014). As a drawback, the major bulk of the research on SMAC is restricted to runtime as performance metric, and its effectiveness is based on the quality of instance features, not readily available for all NP-hard problems.

It is important to remark that all configurators described above implement the principle of *sharpening*. Sharpening refers to methods that increase the sample size over the run of a configurator mainly to reduce the variance of the performance estimate. This increased sample size, which is usually obtained by evaluating configurations on more instances or repeating runs if not sufficient instances are available, makes also the comparison between high-performing configurations more accurate.

Automated algorithm design

Applying automatic configuration tools to the context of automated design is a natural consequence of exposing parameters that were previously hardwired into the code (Hoos, 2012b). In particular, an augmented algorithm configuration approach expands the configuration space to be searched by configurators such that design choices can also be considered. More precisely, the *design space* of this type of automatic algorithm design approach is defined with the help of a human-designed structural pattern, e.g. a template or a grammar, delimiting how low-level components can be combined to produce reasonable algorithmic designs. In this context, there is a connection between categorical parameters and design choices. On one hand, the latter is generally represented as the former, and the traditional examples of categorical parameters such as the choice of a local search or a crossover operator can be considered design choices. On the other hand, automatic algorithm design approaches propose a much more high-level perspective to existing algorithms from a given field. In particular, the process of crafting a template or a grammar depends on finding design patterns in the existing literature on a given topic, and proposing flexible ways of recombining these components in human-reasonable ways. Often, these works propose novel unified models, revealing the equivalence and interchangeability of components that had been independently proposed.

Since template- and grammar-based approaches differ considerably as to their nature, we next review the main insights and proposals from each group individually.

- **Template-based approaches** comprise the union of configurators with flexible, template-based algorithmic frameworks. Specifically, they are implemented by adding the configurable algorithmic components of the framework to the configuration space to be searched by the configurator. In a template-based approach, a design choice derives from deconstructing existing algorithms into algorithmic component patterns, thus providing different categorical choices to be selected by the configurator. Since the first proposal of this kind (**SATenstein**, KhudaBukhsh et al (2009)), many promising applications of template-based approaches have been proposed (Lindauer et al, 2015; Thornton et al, 2013; López-Ibáñez and Stützle, 2010b, 2012; Dubois-Lacoste et al, 2011; Feurer et al, 2015; Kotthoff et al, 2016; Mendoza et al, 2016). In particular, the proposals related to multi-objective optimization will be discussed in the next section.
- **Grammar-based approaches** encode the possible combinations of algorithmic components within some grammar (or equivalently as a finite state machine). When compared to template-based approaches, grammar-based approaches offer a more expressive approach that can, for example, also consider recursive design components. The combination of grammar-based approaches with automatic configuration tools has been first proposed by Mascia et al (2013, 2014b) and been used to configure hybrid metaheuristics from a flexible framework (Marmion et al, 2013; Mascia et al, 2014a). More precisely, in

grammar-based automated algorithm configuration, the design space searched by the configurator is defined in function of a (context-free) grammar. While earlier approaches to design heuristic algorithms such as local-search based SAT heuristics (Fukunaga, 2004, 2008) or iterated greedy heuristics for bin packing (Burke et al, 2012) have made use of genetic programming or evolutionary approaches to grammars such as grammatical evolution, the works mentioned above tackle automated algorithm design using configurators. This is done by first computing the set of possible derivations allowed by the grammar and then translating this into a parametric space. Therefore, any parameter instantiation selected by the configurator corresponds to a valid grammar derivation and can be evaluated on the target problem. Overall, grammar-based approaches provide both benefits and drawbacks. As major advantage, algorithm designers are given enhanced expressivity, being able to produce complex algorithmic designs based on recursive rules, the most prominent being hybridization. By contrast, the maximum height of the (implicit) derivation trees produced from the selected grammar must be kept small to prevent the parameter space from growing beyond practicality.

In conclusion, augmented algorithm configuration is both a feasible and effective approach to the automatic design of algorithms. More importantly, a number of insights are produced throughout the deconstruction, assembling, and design phases. For instance, during deconstruction it is common to identify equivalent algorithms or algorithmic components that had been independently proposed by different research groups. During assembling, it is not uncommon to envision novel applications of existing components once a more high-level template has been identified. Finally, the automatically designed algorithms are much more reasonable from a human perspective, and hence it is possible to analyze why these designs work well and how they could be further fiddled with to produce yet more insights or become more effective.

1.3 Automatic configuration of multi-objective optimizers

As discussed in the previous section, the automatic configuration process requires a performance metric to be optimized. However, the performance assessment of multi-objective optimizers is a complex task that has been subject to a significant research effort, and many metrics have been proposed for evaluating multi-objective algorithms. In this section, we first review the most relevant assessment approaches. Next, we review the main proposals of automatic configuration of multi-objective optimizers, and how they have addressed the performance assessment issue.

Performance assessment of multi-objective optimizers

The concept of high-quality approximation fronts is not straightforward. In the best-case scenario, every front generated by one optimizer Φ_1 strictly dominates every front generated by another optimizer Φ_2. In practice, however, this is rarely the case. In order to evaluate approximation fronts, several methodologies can be used, such as *dominance rankings* (Knowles et al, 2006) and quality metrics, or *indicators* (Zitzler et al, 2003; Jiang et al, 2014; Ishibuchi et al, 2015; Schütze et al, 2012).

- **Dominance rankings:** A rigorous comparison between the approximation fronts produced by different multi-objective optimizers is to use dominance rankings. In more detail, let Φ_1 and Φ_2 be two algorithms one wants to compare and $A^1_{\Phi_1}, A^2_{\Phi_1}, \ldots, A^r_{\Phi_1}$ and $A^1_{\Phi_2}, A^2_{\Phi_2}, \ldots, A^r_{\Phi_2}$ the approximation fronts they respectively produce over a series of r runs, comprising a collection $\mathcal{C}$. Each of these fronts is assigned a dominance ranking depicting how many fronts from $\mathcal{C}$ are better in terms of Pareto optimality (Zitzler et al, 2003) than it. In this way, both algorithms can now be evaluated based on the ranking values their approximation fronts achieve. In particular, statistical analysis can investigate whether this transformed sample for Φ_1 is significantly different from the sample for Φ_2, and post-hoc tests can indicate which algorithm produces better quality fronts if difference is observed (Knowles et al, 2006). As previously mentioned, this is a fairly strict approach since it is only possible to discriminate between algorithms that present very different performance.
- **Quality indicators:** These metrics either (i) analytically measure a given (set of) characteristic(s) a high-quality front should present, or (ii) analytically assess the difference between two fronts, in which case it is possible to evaluate how well a front approximates the Pareto front. In the former case, a metric is said to be unary, whereas in the latter it is said to be binary. As we will see later, some binary quality indicators can be used to construct unary quality indicators. More importantly, a requirement that should be observed by indicators concerns their agreement with Pareto dominance, formally defined as follows. Let $\mathcal{I}\colon \Omega \to \mathbb{R}$ be a quality indicator, which is to be maximized. $\mathcal{I}$ is said to be Pareto-compliant if, and only if, for every pair of approximation fronts $(A, B) \in \Omega$ for which $I(A) \geq I(B)$, it also holds that $B \not\preceq A$.
  As discussed by Zitzler et al (2003), true unary quality indicators display a limited potential for comparing sets of solutions while respecting Pareto dominance. By contrast, some binary indicators can deliver Pareto compliance, but the amount of data produced when analyzing a set of algorithms with multiple runs using binary indicators can be overwhelming. For these reasons, the most appropriate approaches to quality indicators are either to (i) use binary indicators as an auxiliary method for computing dominance rankings, or; (ii) reformulate binary indicators as unary indicators considering the comparison between an approximation front and a reference front.
  The ideal reference fronts to be used by quality indicators are true Pareto fronts. However, in combinatorial optimization it is often the case that one cannot

compute these directly given the NP-hardness of many of the original single-objective problems. For continuous problems, the artificially designed problems typically considered for multi-objective optimization present backdoors that allow Pareto optimal solutions to be easily generated. However, these Pareto fronts can be too large for practical purposes depending on the tolerance level used and the correlation between the different objectives. Whenever Pareto fronts are not available, reference sets can be assembled by merging all approximation fronts found by all optimizers being assessed. At this point, one can either (i) filter these supersets to leave only nondominated solutions or (ii) generate "average fronts" via different methods (Knowles et al, 2006). Although this approach is far from ideal, reference sets can become rich information sources as long as they are continuously refined by adding solutions found by high-performing algorithms.

Automatic configuration and multi-objective optimization

In the literature, the research on automatic configuration and multi-objective optimization intersect in two different ways. The first concerns the *automatic configuration of multi-objective optimizers*, and many different approaches have been proposed to this end. In particular, the most widely adopted approach considers a single metric to be optimized, typically a quality indicator (Wessing et al, 2010; López-Ibáñez and Stützle, 2010b). For this purpose, every time a configuration is run on an instance, the approximation front it produces is transformed into a scalar measure by means of a quality indicator. The multi-objective nature of the optimizers being configured is therefore transparent to the configurator.

The second way in which the fields of automatic configuration and multi-objective optimization intersect concerns the research on *multi-objective configuration* (Dréo, 2009; Zhang et al, 2013). Proposers of such approaches consider that the configuration task is, in itself, an optimization problem that can present multiple and often conflicting criteria. For instance, the major bulk of the research on automatic configuration concerns either solution quality or runtime, two objectives that are often difficult to simultaneously optimize. Therefore, different research groups have recently proposed adaptations of configurators to configure algorithms based on multiple criteria, even if the underlying algorithm being configured tackles a single-objective optimization problem. This second group of approaches will be further discussed in Section 1.5.

Next, we review the most important approaches of the former kind, namely the automatic configuration of multi-objective optimizers.

Main applications of the automatic configuration of multi-objective optimizers

Given that the field of automatic configuration is recent, it is not surprising that not many approaches concerning multi-objective optimization can be identified. In particular, this is further explained by the characteristics of the automatic configurators available, as only irace allows the computation of performance metrics on-the-fly in a complex way, as required by multi-objective optimization. We next describe the most relevant proposals we identify in the literature, as they have been the first to propose a feasible approach to this task:

- **The MOACO framework** (López-Ibáñez and Stützle, 2012) was the first proposal of template-based automatic algorithm design applied to a multi-objective optimization scenario. In particular, the authors expanded an existing *ant colony optimization* (ACO) framework (Stützle, 2002) to deal with a bi-objective optimization problem, namely the traveling salesman problem (TSP). More importantly, the different design choices used to assemble this framework were gathered by deconstructing the most relevant multi-objective ACO (MOACO) proposals from the literature and experimentally analyzing various of these components (López-Ibáñez and Stützle, 2010a,c). The augmented configuration space was defined based on the MOACO template that underlies this framework. Configurations were evaluated based on the unary hypervolume indicator, adopting objective-wise normalization and on-the-fly reference front assembling and bound computation at the end of each iteration. In addition, this was also the first work to consider a separation between multi-objective components and underlying algorithms, with the two most used ACO algorithms from the literature, Ant colony system (Dorigo and Gambardella, 1997) and $\mathcal{MAX}$-$\mathcal{MIN}$ ant system (Stützle and Hoos, 2000), being available as design choices. The results were remarkable, with the automatically designed MOACO algorithms outperforming by a large margin the MOACO algorithms from which the framework components were gathered. Later, this work was extended to tackle combinatorial problems other than the TSP (Bezerra et al, 2012), adopting the same procedure to evaluate candidate configurations.
- **The TP+PLS framework** (Dubois-Lacoste et al, 2011) built on the same idea of a hypervolume-based evaluation of candidate configurations, with normalization and dynamically computed reference fronts and bounds. More importantly, this was the first proposal of a template-based automatic design approach that considered hybrid metaheuristics. In particular, the two most commonly adopted SLS methods for bi-objective optimization, Pareto local search (PLS, Paquete et al (2004)) and Two-phase local search (TPLS, Paquete and Stützle (2003)), were selected and the most relevant proposals for each method were identified and deconstructed to provide components for a hybrid framework; for more details on such hybrid frameworks see Dubois-Lacoste et al (2013). We remark that, although the algorithm design was done automatically,

the hybridization between metaheuristics was an *a priori* human-designed stage, represented by the given template. The authors considered the bi-objective PFSP as application problem and also used irace as configurator. Results once again showed the improved performance of automatically designed algorithms when compared to manually designed ones.

- **The AutoMOEA framework** (Bezerra et al, 2016) refined the methodology used by previous investigations on hypervolume-based configuration of multi-objective optimizers. Concretely, the authors identified that pre-establishing bounds based on known objective-wise desired solution quality helped the normalization process, effectively discarding strong outliers that made it harder for configurators to distinguish between high-performing configurations. They applied this refined methodology to the context of multi-objective evolutionary algorithms (MOEAs), proposing a configurable framework from which a significant number of existing algorithms could be instantiated, in addition to the various novel algorithmic designs that could be produced by the configurator. This work also built on the idea of separating underlying algorithms from multi-objective components, and in recent, yet unpublished versions, the AutoMOEA framework has been incremented to encompass all of the most relevant MOEA search paradigms (Bezerra, 2016). Finally, these recent, ongoing investigations using the AutoMOEA framework have considered different quality indicators as configuration metric, and have empirically confirmed that the disagreements between metrics demand configuration approaches that consider the multi-objective nature of the tuning process.

In the next section, we further detail the characteristics of the AutoMOEA research on the automatic configuration of multi-objective optimizer.

1.4 Case study: automatic configuration of MOEAs

As previously discussed, template-based design approaches consist in identifying algorithmic design patterns. In this context, a design pattern comprises individual algorithmic components proposed in different algorithms from a given class that have the same function. Thus, these components can be seen as interchangeable, and a component from a given algorithm belonging to this class could be replaced by alternative procedures taken either from other algorithms or by newly devised ones sharing the same goal. In the context of MOEAs, examples are the *fitness* and *diversity* components that appear in many algorithms (Bezerra, 2016). This component-wise view has two main benefits. First, it allows algorithm designers to identify the various options available for each algorithmic component and whether a particular combination of components, i.e., an algorithm "design", has already been proposed. Second, it allows users to adapt the design of MOEAs to their particular application scenario.

In this section, we detail the conceptual view of MOEA components behind the AutoMOEA framework, which allowed instantiating, from the same algorithmic template, a larger number of MOEAs from the literature than existing MOEA frameworks. For example, using the AutoMOEA framework one is able to instantiate at least ten well-known MOEAs from the literature, considering also the hybridization between some of the most relevant MOEA search paradigms. On a higher level, this is achieved by considering MOEAs from a component-wise perspective, separating between the high-level algorithmic components specifically related to multi-objective optimization (MO), and the traditional algorithmic components related to search in optimization problems, i.e., the underlying evolutionary algorithm (EA). From a more detailed perspective, the representativeness of the proposed framework is further enhanced by reformulating the traditional distinction between fitness and diversity components (Van Veldhuizen and Lamont, 2000; Liefooghe et al, 2011) as preferences composed by set-partitioning, quality, and diversity metrics (Zitzler et al, 2010), and by allowing different preferences to be used for mating and environmental selection. Finally, our proposal also formalizes the distinction between internal and external populations and archives, which allows us to describe, using alternative options for the same components, algorithms as different as some of the MOEAs we can instantiate from the framework.

A component-wise view of MOEAs

The AutoMOEA framework is built on three fundamental pillars. The first concerns preference relations built from algorithmic components obtained from different search paradigms. Effectively, our modeling of these components allows designers to combine, in a single algorithm, components originally proposed for search paradigms as different as dominance- and indicator-based MOEAs (Bezerra et al, 2016). In Bezerra (2016), this has been extended to instantiate also decomposition-based MOEAs. A second pillar of the AutoMOEA framework is the separation between components related to the multi-objective aspects and the underlying EAs used. Concretely, our template allows the same set of MO components to be coupled with different EAs by simply changing the value of categorical parameters. By doing so, we increase the representativeness of the AutoMOEA template, since one can instantiate many MOEAs that are based on differential evolution (Abbass et al, 2001; Abbass, 2002; Madavan, 2002; Robič and Filipič, 2005; Kukkonen and Lampinen, 2005; Tušar and Filipič, 2007), for instance. Finally, our modeling of populations and archives allows us to instantiate, from a single framework, algorithms that may or may not use internal and external archivers, with preference relations customized for each archiver considered. This way, we deem equivalent environmental selection and archive truncation approaches, allowing

Algorithm 1 AutoMOEA template.

```
1: pop ← Initialization ()
2: if type(pop_ext) ≠ none then
3:     pop_ext ← pop
4: repeat
5:     pool ← BuildMatingPool (pop)
6:     pop_new ← Variation (pool)
7:     pop_new ← Evaluation (pop_new)
8:     pop     ← Replacement (pop, pop_new)
9:     if type(pop_ext) = bounded then
10:        pop_ext ← Replacement_Ext (pop_ext, pop_new)
11:    else if type(pop_ext) = unbounded then
12:        pop_ext ← pop_ext ∪ pop
13: until termination criteria met
14: if type(pop_ext) = none then
15:    return pop
16: else
17:    return pop_ext
```

Table 1.1 Composite components implemented in the AutoMOEA framework.

Component	Parameters
Preference	⟨ SetPart, Refinement, Diversity ⟩
BuildMatingPool	⟨ Preference_{Mat}, Selection ⟩
Replacement	⟨ Preference_{Rep}, Removal ⟩
Replacement_{Ext}	⟨ Preference_{Ext}, Removal_{Ext} ⟩
UnderlyingEA	⟨ BuildMatingPool, Variation ⟩

instantiations of structurally different algorithms as well as novel designs with combinations of components that had never been envisioned before.

The AutoMOEA template is depicted in Algorithm 1, and its main components are listed in Table 1.1, which can be briefly summarized as follows:

- Preference is a *composite* component that encapsulates a sequence of three *atomic* components used in the following order. The first, SetPart, partitions solutions into dominance-equivalent clusters. The second component, Refinement, ranks solutions within each partition, in general by means of quality indicators. Finally, component Diversity is used to keep the population well-spread across the objective space. A Preference component can also contain less than three atomic components since SetPart, Refinement, and Diversity can be set to *none*.
- BuildMatingPool uses traditional Selection operators to assemble a mating pool. In the case of tournaments, solutions are compared based on a preference relation Preference_{Mat}.
- Replacement and Replacement_{Ext} components, respectively, define environmental selection and external archive truncation (if an archive is used). Both Replacement components ensure elitism, and comprise two other components:

a preference relation ($\mathsf{Preference}_{Rep}$ and $\mathsf{Preference}_{Ext}$, respectively), used to compare solutions, and Removal, a policy that determines the frequency with which Preference is computed.
- Initialization and Variation encapsulate problem-specific components; in particular, how to generate an initial population and the variation operators used to produce novel solutions.
- UnderlyingEA is a high-level component that allows the AutoMOEA framework to freely combine MO components and different underlying EAs. In particular, we consider the two most relevant EAs used for continuous optimization, namely *genetic algorithms* (GAs, Goldberg (1989)) and *differential evolution* (DE, Price et al (2005)). We model the underlying EA as a composite component, which comprises composite components BuildMatingPool and Variation (see Table 1.1). In particular, we do so as EAs differ not only as to the operators used for variation, but also on how to select individuals to undergo variation.

Since the focus of this chapter is rather on the automatic configuration aspect of our work, we refrain from further low-level implementation details. For that matter, the reader is referred to Bezerra (2016) and Bezerra et al (2016).

Automatic MOEA configuration

Our experimental investigation, presented originally in Bezerra et al (2016), had two main goals. The first was to assess how automatically designed MOEAs (hereafter called AutoMOEAs) perform compared to several standard MOEAs that can be instantiated from our framework. Second, we wanted to investigate how much the structure of the AutoMOEAs vary depending on the benchmark and the number of objectives considered. The benchmark problems that have been considered at the design time of an algorithm may implicitly or explicitly bias the algorithm design. We studied this effect by considering two different benchmark sets, the DTLZ set (Deb et al, 2005) and the WFG set (Huband et al, 2006). Each benchmark set was used with two, three and five objectives. We separated between different number of objectives as it is known before running an algorithm and, obviously, an algorithm configuration that performs well for a low number of objectives (e.g. 2 or 3) need not perform well for more objectives (e.g. 5). We then designed AutoMOEAs for each of the six scenarios obtained from the combinations of benchmark set (DTLZ and WFG) and number of objectives (2, 3, and 5), as we discuss next. For further details on the experimental setup, the reader is referred to Bezerra et al (2016).

The designs of the AutoMOEAs selected by irace for each of the scenarios we consider are provided as supplementary material of the original paper (Bezerra et al, 2015). Although patterns can be observed, it is hard to establish general guidelines for selecting components when we consider a specific benchmark or

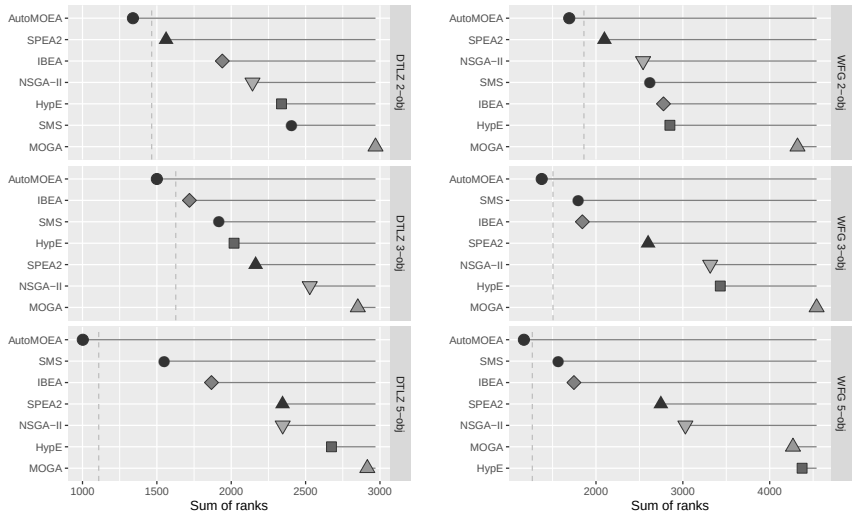

Fig. 1.1 Sum of ranks depicting the performance of MOEAs on DTLZ (left) and WFG (right) sets according to the I_H^{rpd}. Smaller values indicate better or more robust performance. The vertical dashed line in each plot indicates the critical rank sum difference for Friedman's test with 99% confidence.

a specific number of objectives. However, the I_H^{rpd} rank sum analysis given in Figure 1.1 shows that each of these AutoMOEA variants perform very well on the scenarios for which they were designed. This result is consistent with our expectations that different scenarios should demand different components, and that the component-wise design proposed here provides enough flexibility to meet this need.

To further validate the effectiveness of the automatic configuration approach adopted in this investigation, we conducted three additional sets of experiments:

- **Runtime-constrained setup:** As shown by the results discussed above, standard MOEAs tend to perform better on the scenarios for which they have been properly tuned. Besides the benchmark set and the number of objectives considered, another major factor that affects the performance of algorithms is the stopping criterion used to terminate their runs. In continuous optimization, a maximum number of function evaluations (FE) is typically used because some applications present computationally costly FEs. As a result, algorithm designers tend to devise algorithms that are able to reach high-quality solutions with as few FEs as possible. Moreover, the time spent by the algorithms computing metrics or discarding solutions is not considered an issue in these scenarios and, hence, very fast and very slow algorithms are often considered equal. For instance, SMS-EMOA requires almost 10 minutes for executing 10 000 FEs in our computer environment, while IBEA terminates

in seconds. However, in many practical situations the computational cost of the FEs may not be high enough to justify large computation times. In such scenarios, fast algorithms such as IBEA or NSGA-II could likely outperform slow ones such as SMS-EMOA by seeing many more solutions within a maximum runtime.
By contrast, our design approach is able to deal with such changes naturally. Overall, the results confirm that the overhead incurred by MOEA components can greatly impair their efficiency when facing a problem that is not computationally expensive, but requires a constrained runtime. Nonetheless, when adequate algorithmic components are available in the framework, the automatic configuration process is able to identify the designs that are more suitable for the given setup.

- **Cross-benchmark setup**: We additionally investigated the cross-benchmark performance generalization of the AutoMOEAs, by comparing the various MOEA algorithms on the benchmarks for which they have not been tuned. More precisely, the algorithms tuned on the WFG training set of functions were run on the DTLZ benchmark set, and vice versa. This analysis considered only the setup with a maximum number of FEs to use and the results of the rank sum analysis of the I_H^{rpd} are given in Figure 1.2. In most cases, the relative order among the algorithms remains very similar to the one encountered in Figure 1.1. In five out of six cases the AutoMOEA algorithms remain the best performing ones. The results for the $I_{\epsilon+}$ indicator are consistent with these ones for all scenarios, despite minor differences.
- **Combinatorial optimization**: To further validate the proposed methodology on an application domain that is rather different from the domain for which MOEAs were originally conceived, we devised AutoMOEAs for tackling four multi-objective permutation flow shop problems (MO-PFSP), a well-known class of multi-objective combinatorial problems. Although MOEAs are not always designed with combinatorial optimization problems in mind, many of the MOEAs we considered in our investigation have been adapted to such problems using problem-specific variation operators (Minella et al, 2008). Indeed, the designs of the AutoMOEAs devised for the PFSP differed in many aspects from those devised for continuous optimization problems. Nonetheless, the performance displayed by the AutoMOEAs confirmed the efficacy of the automatic MOEA design also for combinatorial optimization. This further highlights the importance of having a flexible and representative MOEA framework.

The experiments reported in this section have confirmed the importance of the automatic design methodology for developing MOEAs for continuous and combinatorial optimization, highlighting both its effectiveness and flexibility. Under all application scenarios and setups considered here, the AutoMOEAs were able to present a robust behavior and often outperform all standard MOEAs, even if these are configured using the same setup and the same configuration budget. At the same time, the performance of these standard MOEAs varied considerably. Although IBEA performed well on most setups

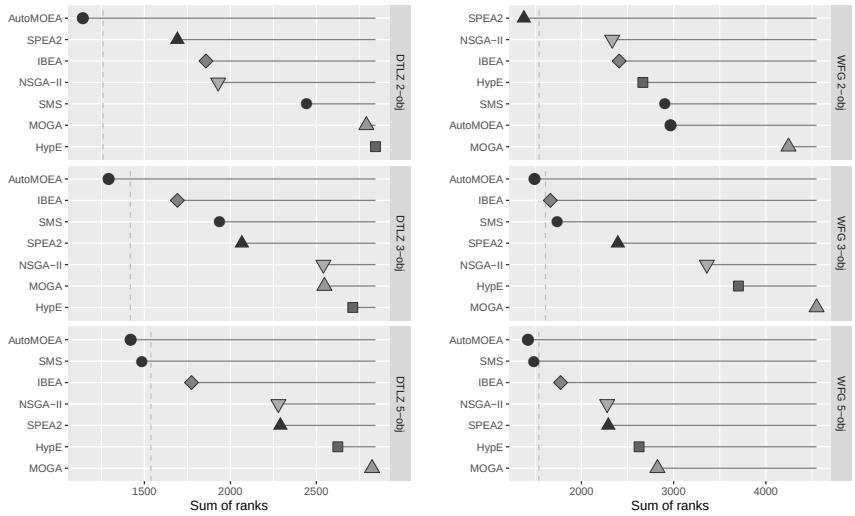

Fig. 1.2 Sum of ranks depicting the performance of MOEAs on DTLZ (left) and WFG (right) sets when tuned for the opposite set according to the I_H^{rpd}. Smaller values indicate better or more robust performance. The vertical dashed line in each plot indicates the critical rank sum difference for Friedman's test with 99% confidence.

we adopted, the AutoMOEAs were able to consistently outperform it in the majority of cases.

1.5 Multi-objective configuration of algorithms

The second main use of multiple objectives in the context of configuration arises when more than one objective is to be considered for defining the quality of an algorithm configuration. In this case, the aim becomes producing a set of parameter configurations that are mutually non-dominated w.r.t. the multiple quality measures for evaluating configurations. A first discussion of the potential usefulness of automatically tuning algorithms for multiple objectives is due to Dréo (2009). Further motivation for the development of multi-objective configuration techniques is also given by Dang Thi Thanh and De Causmaecker (2014). In Dréo's prototypical experimental studies he used as measures the speed (that is, execution time measured either in CPU time or number of objective function evaluations) of an algorithm and as a second the precision (that is, the solution quality reached). The examples studied considered setting a single parameter and exploring this (small) configuration space using NSGA-II and evaluating each configuration a same number of times.

Zhang et al (2013) proposed to tackle multi-objective configuration tasks using an extension of racing procedures to the multi-objective context. They propose S-race, where each of the candidate configurations is evaluated according to multiple criteria. Each of the alive candidate configurations is then evaluated on one or several training instances and candidate configurations that are dominated by some other candidate configuration are eliminated from the race. The elimination test in S-race is done by the sign test for matched pairs. The sign test checks for two candidate configurations θ_i and θ_j on how many instances one dominates the other and vice versa (n_{ij} and n_{ji}, respectively). Dominance is then based on pairwise sign tests among all surviving candidate configurations using Holm's procedure for correcting the α-level for the multiple testing. S-race was evaluated on a task of selecting support vector machine configurations, where 50 configurations have been randomly sampled using seven possible parameters. A comparison of the results of S-race showed significant time saving w.r.t. a brute-force evaluation; additional experiments with S-race are provided by Zhang et al (2016). In the original S-race, however, evaluations where two configurations are mutually non-dominated, are discarded for the statistical testing. As a remedy for this case and for improving the testing procedure, Zhang et al (2015) extend S-race to use a sequential probability ratio test instead of the sign test, and an additional indifference zone to allow early stopping of comparisons of two configurations in case they are mutally non-dominated. An extension of S-race to an iterated version in the same spirit as Iterated F-race extends F-race is proposed by Miranda et al (2015), who reported significant improvements by the iterated version over the underlying S-race used as a stand-alone procedure.

Another approach to multi-objective configuration is the multi-objective extension of ParamILS proposed by Blot et al (2016), which is called MO-ParamILS. The main extension in MO-ParamILS when compared to ParamILS comprises the usage of an archive of configurations that plays the same role as an archive of solutions in Pareto local search approaches (Paquete et al, 2004). For comparing configurations a dominance criterion is defined between configurations that uses for each of the configuration the estimates of the respective objective values. As in ParamILS, also in MO-ParamILS configurations are either evaluated on a fixed number of problem instances (BasicILS variant) or the number of instances on which the two configurations are compared is increased in analogy to the FocusedILS variant. For initializing the archive, a number of default configurations can be defined and r randomly chosen configurations are added. Care is taken that configurations dominated according to the used dominance criterion are eliminated from the initial archive. In the local search process, the neighborhood of each configuration in the archive is examined until a neighbor is found that dominates it. Non-dominated neighbors found in this process are also added to the archive. Apart from these details, ParamILS's main design features are maintained: in a given iteration, a single configuration selected uniformly at random from

the current archive is perturbed by doing $s = 3$ random steps and becomes a new initial archive for the local search process; with a probability of $p = 0.01$ a restart is applied instead of a perturbation and a random configuration is chosen as a new initial archive. Experiments on multi-objective configuration tasks involving time and quality objectives for a mixed integer programming solver (CPLEX) and the memory versus running time objectives for a SAT solver (CLASP) showed a clear advantage of the MO-ParamILS variant that extends FocusedILS over the one relying on the BasicILS variant. More recently, authors apply the MO-ParamILS automatic configurator to configure a multi-objective local search algorithm based on the Pareto local search (PLS) paradigm and test it in the context of bi-objective permutation flow-shop problems (Blot et al, 2017). They evaluate the performance of the configured PLS algorithm using the hypervolume measure and the spread indicator that measures the distribution of a set of solutions in bi-objective problems. They showed that using the multi-objective configuration approach results in a larger number of trade-off configurations than either optimizing only a single measure or a weighted sum of the two quality measures for evaluating configurations.

1.6 Summary

Multi-objective optimization and automatic algorithm configuration are two fields that have consistently demonstrated their relevance in the computational intelligence research community. In this chapter, we have briefly reviewed the two main ways in which these fields intersect. The first concerns the automatic configuration of multi-objective optimizers, where the multi-objective nature of the problem being solved by the optimizer is made transparent to the configurator through quality metrics that assign scalar values to the quality of approximation fronts. The second refers to the multi-objective configuration of optimizers, where the goal of the configurator is to simultaneously optimize multiple criteria, such as solution quality or resource consumption. In this context, either single- or multi-objective optimizers can be configured, as demonstrated by the works previously presented.

More importantly, we have detailed works that demonstrate the potential of such approaches to redefine the traditional algorithm engineering process, where algorithms are designed in isolation of their parameter configurations or focusing on a single performance metric one attempts to optimize. By adopting the heuristic engineering process repeatedly illustrated here, it is possible to conceive algorithms from a more high-level perspective, without making premature assumptions about the effectiveness of given algorithmic components, or spending a significant amount of resources in the traditional test-redesign loop.

Acknowledgements This work received support from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a research director.

References

Abbass HA (2002) The self-adaptive Pareto differential evolution algorithm. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), IEEE Press, Piscataway, NJ, pp 831–836

Abbass HA, Sarker R, Newton C (2001) PDE: a Pareto-frontier differential evolution approach for multi-objective optimization problems. In: Proceedings of the 2001 Congress on Evolutionary Computation (CEC'01), IEEE Press, Piscataway, NJ, pp 971–978

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

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, Blesa MJ, Blum C, Naujoks B, Roli A, Rudolph G, Sampels M (eds) Hybrid Metaheuristics, Lecture Notes in Computer Science, vol 4771, Springer, Heidelberg, Germany, pp 108–122

Bartz-Beielstein T, Lasarczyk C, Preuss M (2010) The sequential parameter optimization toolbox. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, pp 337–360

Bezerra LCT (2016) A component-wise approach to multi-objective evolutionary algorithms: from flexible frameworks to automatic design. PhD thesis, IRIDIA, École polytechnique, Université Libre de Bruxelles, Belgium

Bezerra LCT, López-Ibáñez M, Stützle T (2012) Automatic generation of multi-objective ACO algorithms for the biobjective knapsack. In: Dorigo M, et al (eds) Swarm Intelligence, 8th International Conference, ANTS 2012, Lecture Notes in Computer Science, vol 7461, Springer, Heidelberg, Germany, pp 37–48

Bezerra LCT, López-Ibáñez M, Stützle T (2015) Automatic component-wise design of multi-objective evolutionary algorithms. `http://iridia.ulb.ac.be/supp/IridiaSupp2014-010/`

Bezerra LCT, López-Ibáñez M, Stützle T (2016) Automatic component-wise design of multi-objective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 20(3):403–417

Birattari M (2004) The problem of tuning metaheuristics as seen from a machine learning perspective. PhD thesis, IRIDIA, École polytechnique, Université Libre de Bruxelles, Belgium

Birattari M, Yuan Z, Balaprakash P, Stützle T (2010) F-race and iterated F-race: An overview. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, pp 311–336

Blot A, Hoos HH, Jourdan L, Kessaci-Marmion ME, Trautmann H (2016) MO-ParamILS: A multi-objective automatic algorithm configuration framework. In: Festa P, Sellmann M, Vanschoren J (eds) Learning and Intelligent Optimization, 10th International

Conference, LION 10, Lecture Notes in Computer Science, vol 10079, Springer, Cham, Switzerland, pp 32–47

Blot A, Pernet A, Jourdan L, Kessaci-Marmion ME, Hoos HH (2017) Automatically configuring multi-objective local search using multi-objective optimisation. In: Trautmann H, Rudolph G, Klamroth K, Schütze O, Wiecek MM, Jin Y, Grimme C (eds) Evolutionary Multi-criterion Optimization, EMO 2017, Lecture Notes in Computer Science, Springer International Publishing, Cham, Switzerland, pp 61–76

Box GEP, Draper NR (2007) Response surfaces, mixtures, and ridge analyses. John Wiley & Sons

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Burke EK, Hyde MR, Kendall G (2012) Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation 16(7):406–417

Dang Thi Thanh N, De Causmaecker P (2014) Motivations for the development of a multi-objective algorithm configurator. In: Vitoriano B, Pinson E, Valente F (eds) ICORES 2014 - Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems, SciTePress, pp 328–333

Deb K, Thiele L, Laumanns M, Zitzler E (2005) Scalable test problems for evolutionary multiobjective optimization. In: Abraham A, Jain L, Goldberg R (eds) Evolutionary Multiobjective Optimization, Advanced Information and Knowledge Processing, Springer, London, UK, pp 105–145

Dorigo M, Gambardella LM (1997) Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53–66

Dréo J (2009) Using performance fronts for parameter setting of stochastic metaheuristics. In: Rothlauf F (ed) GECCO (Companion), ACM Press, New York, NY, pp 2197–2200

Dubois-Lacoste J, López-Ibáñez M, Stützle T (2011) Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In: Krasnogor N, Lanzi PL (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, ACM Press, New York, NY, pp 2019–2026

Dubois-Lacoste J, López-Ibáñez M, Stützle T (2013) Combining two search paradigms for multi-objective optimization: Two-Phase and Pareto local search. In: Talbi EG (ed) Hybrid Metaheuristics, Studies in Computational Intelligence, vol 434, Springer Verlag, pp 97–117

Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1(1):19–31

Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and robust automated machine learning. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds) Advances in Neural Information Processing Systems (NIPS 28), pp 2962–2970

Fukunaga AS (2004) Evolving local search heuristics for SAT using genetic programming. In: Deb K, et al (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2004, Part II, Lecture Notes in Computer Science, vol 3103, Springer, Heidelberg, Germany, pp 483–494

Fukunaga AS (2008) Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1):31–61

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Boston, MA, USA

Hoos HH (2012a) Automated algorithm configuration and parameter tuning. In: Hamadi Y, Monfroy E, Saubion F (eds) Autonomous Search, Springer, Berlin, Germany, pp 37–71

Hoos HH (2012b) Programming by optimization. Communications of the ACM 55(2):70–80

Huband S, Hingston P, Barone L, While L (2006) A review of multiobjective test problems and a scalable test problem toolkit. IEEE Transactions on Evolutionary Computation 10(5):477–506

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36:267–306

Hutter F, Hoos HH, Leyton-Brown K (2010) Automated configuration of mixed integer programming solvers. In: Lodi A, Milano M, Toth P (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Lecture Notes in Computer Science, vol 6140, Springer, Heidelberg, Germany, pp 186–202

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello Coello CA (ed) Learning and Intelligent Optimization, 5th International Conference, LION 5, Lecture Notes in Computer Science, vol 6683, Springer, Heidelberg, Germany, pp 507–523

Hutter F, Xu L, Hoos HH, Leyton-Brown K (2014) Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206:79–111

IBM (2017) ILOG CPLEX optimizer. `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`

Ishibuchi H, Masuda H, Tanigaki Y, Nojima Y (2015) Modified distance calculation in generational distance and inverted generational distance. In: Gaspar-Cunha A, Antunes CH, Coello Coello CA (eds) Evolutionary Multi-criterion Optimization, EMO 2015 Part I, Lecture Notes in Computer Science, vol 9018, Springer, Heidelberg, Germany, pp 110–125

Jiang S, Ong YS, Zhang J, Feng L (2014) Consistencies and contradictions of performance metrics in multiobjective optimization. IEEE Transactions on Cybernetics 44(12):2391–2404

KhudaBukhsh AR, Xu L, Hoos HH, Leyton-Brown K (2009) SATenstein: Automatically building local search SAT solvers from components. In: Boutilier C (ed) Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), AAAI Press, Menlo Park, CA, pp 517–524

Knowles JD, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK-Report 214, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, revised version

Kotthoff L, Thornton C, Hoos HH, Hutter F, Leyton-Brown K (2016) Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. Journal of Machine Learning Research 17:1–5

Kukkonen S, Lampinen J (2005) GDE3: the third evolution step of generalized differential evolution. In: Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005), IEEE Press, Piscataway, NJ, pp 443–450

Liao T, Stützle T, Montes de Oca MA, Dorigo M (2014) A unified ant colony optimization algorithm for continuous optimization. European Journal of Operational Research 234(3):597–609

Liefooghe A, Jourdan L, Talbi EG (2011) A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. European Journal of Operational Research 209(2):104–112

Lindauer MT, Hoos HH, Hutter F, Schaub T (2015) AutoFolio: an automatically configured algorithm selector. Journal of Artificial Intelligence Research 53:745–778

López-Ibáñez M, Stützle T (2010a) An analysis of algorithmic components for multiobjective ant colony optimization: A case study on the biobjective TSP. In: Collet P, Monmarché N, Legrand P, Schoenauer M, Lutton E (eds) Artificial Evolution: 9th International Conference, Evolution Artificielle, EA, 2009, Lecture Notes in Computer Science, vol 5975, Springer, Heidelberg, Germany, pp 134–145

López-Ibáñez M, Stützle T (2010b) Automatic configuration of multi-objective ACO algorithms. In: Dorigo M, et al (eds) Swarm Intelligence, 7th International Conference, ANTS 2010, Lecture Notes in Computer Science, vol 6234, Springer, Heidelberg, Germany, pp 95–106

López-Ibáñez M, Stützle T (2010c) The impact of design choices of multi-objective ant colony optimization algorithms on performance: An experimental study on the biobjective TSP. In: Pelikan M, Branke J (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2010, ACM Press, New York, NY, pp 71–78

López-Ibáñez M, Stützle T (2012) The automatic design of multi-objective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16(6):861–875

López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58

Lourenço HR, Martin O, Stützle T (2002) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer Academic Publishers, Norwell, MA, pp 321–353

Madavan NK (2002) Multiobjective optimization using a Pareto differential evolution approach. In: Proceedings of the 2002 World Congress on Computational Intelligence (WCCI 2002), IEEE Press, Piscataway, NJ, pp 1145–1150

Marmion ME, Mascia F, López-Ibáñez M, Stützle T (2013) Automatic design of hybrid stochastic local search algorithms. In: Blesa MJ, Blum C, Festa P, Roli A, Sampels M (eds) Hybrid Metaheuristics, Lecture Notes in Computer Science, vol 7919, Springer, Heidelberg, Germany, pp 144–158

Mascia F, López-Ibáñez M, Dubois-Lacoste J, Stützle T (2013) From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: Pardalos PM, Nicosia G (eds) Learning and Intelligent Optimization, 7th International Conference, LION 7, Lecture Notes in Computer Science, vol 7997, Springer, Heidelberg, Germany, pp 321–334

Mascia F, López-Ibáñez M, Dubois-Lacoste J, Marmion ME, Stützle T (2014a) Algorithm comparison by automatically configurable stochastic local search frameworks: A case study using flow-shop scheduling problems. In: Blesa MJ, Blum C, Voß S (eds) Hybrid Metaheuristics, Lecture Notes in Computer Science, vol 8457, Springer, Heidelberg, Germany, pp 30–44

Mascia F, López-Ibáñez M, Dubois-Lacoste J, Stützle T (2014b) Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Computers & Operations Research 51:190–199

Mendoza H, Klein A, Feurer M, Springenberg JT, Hutter F (2016) Towards automatically-tuned neural networks. In: Workshop on Automatic Machine Learning, pp 58–65

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

Miranda P, Silva RM, Prudêncio RB (2015) I/S-Race: An iterative multi-objective racing algorithm for the SVM parameter selection problem. In: 22st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning, Bruges, April 23-24-25, 2014, ESANN, pp 573–578

Paquete L, Stützle T (2003) A two-phase local search for the biobjective traveling salesman problem. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary Multi-criterion Optimization, EMO 2003, Lecture Notes in Computer Science, vol 2632, Springer, Heidelberg, Germany, pp 479–493

Paquete L, Chiarandini M, Stützle T (2004) Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Gandibleux X, Sevaux M, Sörensen K, T'Kindt V (eds) Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol 535, Springer, Berlin, Germany, pp 177–200

Pérez Cáceres L, López-Ibáñez M, Hoos HH, Stützle T (2017) An experimental study of adaptive capping in irace. In: Battiti R, Kvasov DE, Sergeyev YD (eds) Learning and Intelligent Optimization, 11th International Conference, LION 11, Lecture Notes in Computer Science, vol 10556, Springer, Cham, Switzerland, pp 235–250

Price K, Storn RM, Lampinen JA (2005) Differential Evolution: A Practical Approach to Global Optimization. Springer, New York, NY

Robič T, Filipič B (2005) DEMO: Differential evolution for multiobjective optimization. In: Coello Coello CA, Aguirre AH, Zitzler E (eds) Evolutionary Multi-criterion Optimization, EMO 2005, Springer, Heidelberg, Germany, Lecture Notes in Computer Science, vol 3410, pp 520–533

Schütze O, Esquivel X, Lara A, Coello Coello CA (2012) Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation 16(4):504–522

Stützle T (2002) ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. URL `http://www.aco-metaheuristic.org/aco-code/`

Stützle T, Hoos HH (2000) $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8):889–914

Thornton C, Hutter F, Hoos HH, Leyton-Brown K (2013) Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Dhillon IS, Koren Y, Ghani R, Senator TE, Bradley P, Parekh R, He J, Grossman RL, Uthurusamy R (eds) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, ACM Press, New York, NY, pp 847–855

Tušar T, Filipič B (2007) Differential evolution versus genetic algorithms in multiobjective optimization. In: Obayashi S, et al (eds) Evolutionary Multi-criterion Optimization, EMO 2007, Springer, Heidelberg, Germany, Lecture Notes in Computer Science, vol 4403, pp 257–271

Van Veldhuizen DA, Lamont GB (2000) Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. Evolutionary Computation 8(2):125–147

Wessing S, Beume N, Rudolph G, Naujoks B (2010) Parameter tuning boosts performance of variation operators in multiobjective optimization. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G (eds) Parallel Problem Solving from Nature, PPSN XI, Lecture Notes in Computer Science, vol 6238, Springer, Heidelberg, Germany, pp 728–737

Zhang T, Georgiopoulos M, Anagnostopoulos GC (2013) S-Race: A multi-objective racing algorithm. In: Blum C, Alba E (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2013, ACM Press, New York, NY, pp 1565–1572

Zhang T, Georgiopoulos M, Anagnostopoulos GC (2015) SPRINT: Multi-objective model racing. In: Silva S, Esparcia-Alcázar AI (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, ACM Press, New York, NY, pp 1383–1390

Zhang T, Georgiopoulos M, Anagnostopoulos GC (2016) Multi-objective model selection via racing. IEEE Transactions on Cybernetics 46(8):1863–1876

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

Zitzler E, Thiele L, Bader J (2010) On set-based multiobjective optimization. IEEE Transactions on Evolutionary Computation 14(1):58–79