Benchmarking in Optimization: Best Practice and Open Issues

Thomas Bartz-Beielstein[1], Carola Doerr[2], Jakob Bossek[3], Sowmya Chandrasekaran[1], Tome Eftimov[4], Andreas Fischbach[1], Pascal Kerschke[5], Manuel López-Ibáñez[6], Katherine M. Malan[7], Jason H. Moore[8], Boris Naujoks[1], Patryk Orzechowski[8,9], Vanessa Volz[10], Markus Wagner[3], and Thomas Weise[11]

[1]*Institute for Data Science, Engineering, and Analytics, TH Köln, Germany*
[2]*Sorbonne Université, CNRS, LIP6, Paris, France*
[3]*Optimisation and Logistics, School of Computer Science, The University of Adelaide, Adelaide, Australia*
[4]*Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia*
[5]*Statistics and Optimization Group, University of Münster, Münster, Germany*
[6]*School of Computer Science and Engineering, University of Málaga, Málaga, Spain*
[7]*Department of Decision Sciences, University of South Africa, Pretoria, South Africa*
[8]*Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, PA, USA*
[9]*Department of Automatics, AGH University of Science and Technology, Krakow, Poland*
[10]*modl.ai, Copenhagen, Denmark*
[11]*Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University, Hefei, China*
benchmarkingbestpractice@gmail.com

July 8, 2020

Abstract

This survey compiles ideas and recommendations from more than a dozen researchers with different backgrounds and from different institutes around the world. Promoting best practice in benchmarking is its main goal. The article discusses eight essential topics in benchmarking: clearly stated goals, well-specified problems, suitable algorithms, adequate performance measures, thoughtful analysis, effective and efficient designs, comprehensible presentations, and guaranteed reproducibility. The final goal is to provide well-accepted guidelines (rules) that might be useful for authors and reviewers. As benchmarking in optimization is an active and evolving field of research this manuscript is meant to co-evolve over time by means of periodic updates.

Contents

1 Introduction

Introducing a new algorithm without testing it on a set of benchmark functions appears to be very strange to every optimization practitioner, unless there is a strong theoretical motivation justifying the interest in the algorithm. Taking theory-focused papers aside, from the very beginning in the 1960s nearly every publication in Evolutionary Computation (EC) was accompanied by benchmarking studies. One of the key promoters of the EC research domain, Hans-Paul Schwefel [1975], wrote in his PhD thesis:

> The extremely large and constantly increasing number of optimization methods inevitably leads to the question of the best strategy. There does not seem to be a clear answer. Because, if there were an optimal optimization process, all other methods would be superfluous ... [1]

Famous studies, e.g., from Moré et al. [1981], were performed in this period and established well-known test functions that are known to every algorithm developer. They can still be found in the portfolio of recent benchmark studies, e.g., Rosenbrock's function [Rosenbrock, 1960]. In the 1960s, experiments could be rerun only a very limited number of times, using different starting points or random seeds. This situation has changed drastically: nowadays, new algorithms can be run a hundred or even a thousand times. This enables very complex and sophisticated benchmark suites such as those available in the Comparing Continuous Optimizers (COCO) [Hansen et al., 2016b] and the Nevergrad [Rapin and Teytaud, 2018] platforms. However, the questions to be answered by benchmarking remain basically the same, e.g.,

- how well does a certain algorithm perform on a given problem?
- why does an algorithm succeed/fail on a specific test problem?

Specifying the goal of a benchmark study is as important as the study itself, as it shapes the experimental setup—i.e., the choice of the problem instances, algorithm instances, performance criteria, and statistics. Typical goals that a user or a researcher wishes to answer through a benchmarking study are discussed in Section 2.

But not only computational power has increased significantly in the last decades. Theory made important progress as well. In the 1980s, some researchers claimed that there is an algorithm that is able to outperform all other algorithms on average [Golden et al., 1986]. A set of no free lunch theorems (NFLTs), presented by Wolpert and Macready [1997] radically changed this situation [Adam et al., 2019]. Brownlee [2007] summarizes NFLT consequences as follows:

> 1) bound claims of algorithm or parameter suitability to the problem instances being tested, 2) research into devising problem classes and matching suitable algorithms to classes is a good thing, 3) be cautious about generalizing performance to other problem instances, and 4) be very cautious about generalizing performance to other problem classes or domains.

Whitley et al. [2002] examine the meaning and significance of benchmarks in light of theoretical results such as NFLT. Problems caused by NFLT are still subject of current research, e.g., Liu et al. [2019] discuss paradoxes in numerical comparison of optimization algorithms based on NFLT. Perhaps the most relevant NFLT consequence reads as follows: statements about the performance of algorithms must be coupled with the problem class or even the problem instances. Or, as stated by Haftka [2016]:

> Improving an algorithm for one class of problem is likely to make it perform more poorly for other problems.

As a NFLT consequence, benchmarking gains a central role, both for theory and practice. It has to take care of the following aspects:

1. performance measure

[1] German original quote: "Die überaus große und ständig steigende Zahl von Optimierungsmethoden führt zwangsläufug zu der Frage nach der besten Strategie. Eine eindeutige Antwort scheint es nicht zu geben. Denn, gäbe es ein optimales Optimierungsverfahren, dann würden sich alle anderen Methoden erübrigen..."

2. problem (instance)

3. algorithm (instance).

Excellent papers on how to set up a good benchmark test exist for many years. Hooker and Johnson are only two authors that published papers still worth reading today [Hooker, 1994, 1995, Johnson et al., 1989, 1991, Johnson, 2002b]. McGeoch [1986] can be considered as a milestone in the field of experimental algorithmics, which builds the cornerstone for benchmark studies. Gent and Walsh [1994] stated that the empirical study of algorithms is a relatively immature field—and we claim, that this situation has unfortunately not significantly changed in the last 25 years. Reasons for this unsatisfactory situation in EC are manifold. For example, EC has not agreed upon general methodology for performing benchmark studies like the fields of statistical Design of Experiments (DOE) or data mining [Chapman et al., 2000, Montgomery, 2017]. These fields provide a general methodology to encourage the practitioner to consider important issues before performing a study.

The question remains: why are minimum standards not considered in every paper submitted to EC conferences and journals? One answer might be: setting up a sound benchmark study is very complicated. There are many pitfalls, especially stemming from complex statistical considerations [Črepinšek et al., 2014]. So, to do nothing wrong, computer scientists oftentimes report only average values decorated with corresponding standard deviations, p-values, or boxplots.

Another answer might be: practical guidelines are missing. Researchers from computer science would apply these techniques, if examples are available. This paper tries to fill this gap. It is a joint initiative from several researchers in EC that was established during the Dagstuhl seminar 19431 on *Theory of Randomized Optimization Heuristics*, which took place in October 2019. Since then, we have been compiling ideas covering a broad range of disciplines, all connected to EC.

We are aware that every version of this paper represents a snapshot, because the field is evolving. New theoretical results such as no-free lunch might come up from theory, new algorithms (quantum computing, heuristics supported by deep learning techniques, etc.) appear on the horizon, and new measures, e.g., based on extensive resampling (Monte Carlo) can be developed in statistics.

It is more than necessary to include as many researchers as possible into this process. We invite everybody who would like to contribute to contact us. Therefore, we have set up an email account: benchmarkingbestpractice@gmail.com.

We consider this paper as a starting point, as a first trial to support the EC community in improving the quality of benchmark studies. Surely, this paper cannot cover every single aspect related to benchmarking. Although this paper focuses on single-objective, unconstrained optimization problems, its findings can be easily transferred to other domains, e.g, multi-objective or constrained optimization. The objectives in other problem domains may differ slightly and may require different performance measures—but the content of most sections should be applicable, too. Each of the following sections presents references to best-practice examples and discusses open topics. The following aspects, which are considered relevant to every benchmark study, are covered in the subsequent sections:

1. Goals: what are the reasons for performing benchmark studies?

2. Problems: how to select suitable problems?

3. Algorithms: how to select a portfolio of algorithms to be included in the benchmark study?

4. Performance: how to measure performance?

5. Analysis: how to evaluate results?

6. Design: how to set up a study, e.g., number of runs? It also discusses the imperative of tuning an algorithm before benchmarking.

7. Presentation: how to describe results?

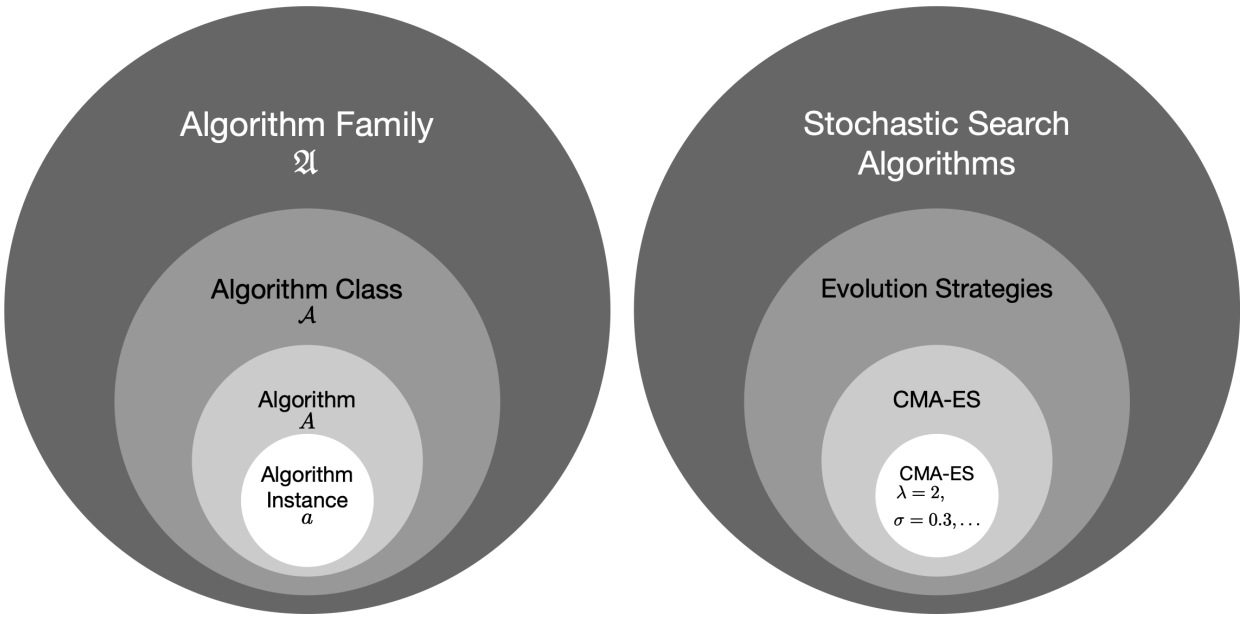

Figure 1: Algorithms. Illustration of the terminology used in this paper. General terms are shown on the *left*, whereas on the *right* one specific example is shown. In a benchmark study, the implementation of algorithm instances belonging to the same or to different algorithm families are considered.

8. Reproducibility: how to guarantee scientifically sound results and how to guarantee a lasting impact, e.g., in terms of comparability?

Notational Conventions. We typically benchmark a *portfolio* (or *set*) of different *algorithm instances*. Each algorithm instance a is the specification of an *algorithm* A from which it is obtained by fixing the parameters and possibly other decisions. Algorithm instances are also referred to as "algorithm configurations" in the literature. That is, we never benchmark an algorithm, but only one or several of its instances. We may even say that we only benchmark a certain *implementation* of an algorithm instance, which is subject to a concrete choice of implementation language, compiler and operating system optimizations, and concrete versions of software libraries.

The algorithms represented in a benchmark study may belong to the same or to different *algorithm classes* $\mathcal{A}$, which, in turn, are specifications of an algorithm family. Figure 1 illustrates this notation (left half) and provides an example (right half). Concretely, we could consider the family of stochastic search algorithms, and within this family the class of *evolution strategies*. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Hansen, 2000] is an algorithm belonging to this class. In order to benchmark "the" CMA-ES, a concrete instance has to be chosen.

Algorithm instances may be (and in the context of this survey often are) randomized, so that the performance of the algorithm instance on a given problem instance is a series of (typically highly correlated) random variables, one for each step of the algorithm. A concrete performance profile [Dean and Boddy, 1988, Zilberstein, 1996] can be reproduced by using the same sequence of numbers that were generated by the random number generator. In practice, such replicability is often achieved by fixing the random number generator and storing the random seed, which plays an important role to guarantee reproducibility as discussed in Sec. 9.

The same terminology introduced above for the space of algorithms can also be applied to the space of possible benchmark problems: we execute the algorithm instances $a_1, a_2, \ldots$ on concrete *problem instances* $\pi_1, \pi_2, \ldots$, which are specifications of a *problem* Π. Most benchmark studies consider performance of algo-

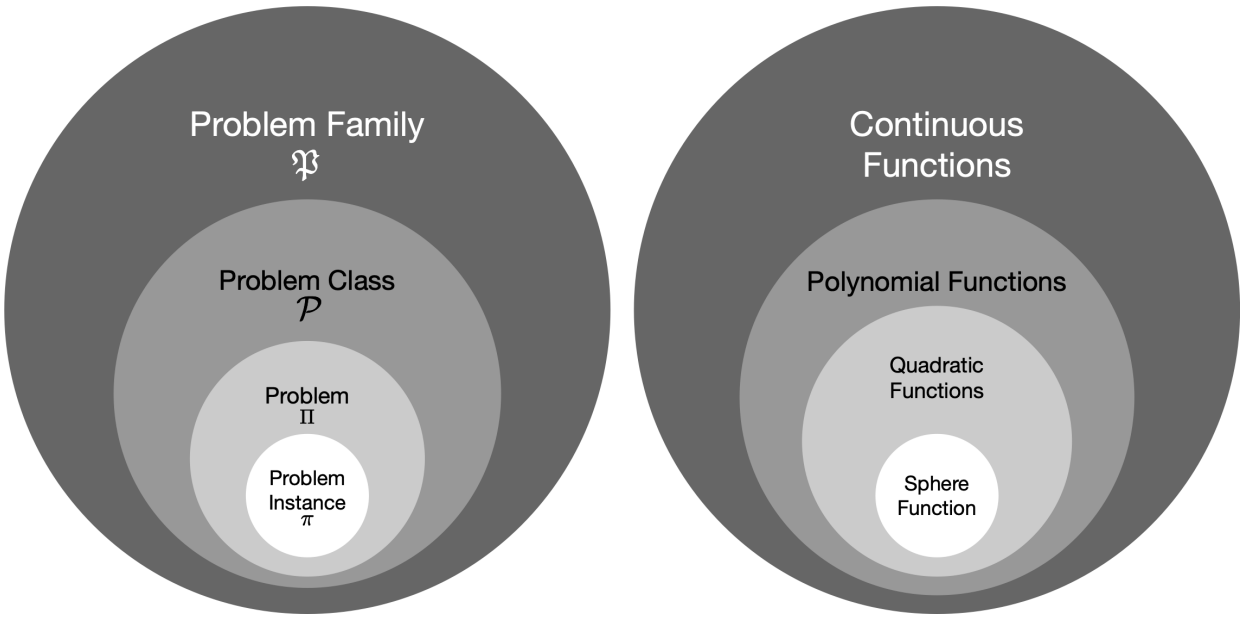

Figure 2: Problems. Illustration of the terminology used in this paper. General terms are shown on the *left*, whereas on the *right* one specific example is shown. It is of great importance that we distinguish between problems and problem instances, because these terms are mixed in many benchmark studies. As a consequence of this imprecise use of terminology, it remains unclear whether the authors refer to one specific problem instance or to the whole class. Furthermore, we will use the terms *set*, *suite*, and *portfolio* to describe a collection of problem instances, that do not necessarily share a common property. As discussed in the text, the example also shows that different classifications are possible, so that great care has to be taken with respect to which problems have been used in the benchmark study.

rithm instances on several problems, which may or may not belong to the same *problem class*, sometimes also referred to as *problem suite*. The problem classes are specifications of the *problem family* $\mathfrak{P}$ they belong to. Thus, finally, the problem instances considered in the benchmark study may stem from a portfolio (set) of different problem families. As is the case for the algorithms, also the problem instances can be randomized, in which case we speak of *stochastic* or *noisy* problem instances.

The notation is illustrated in Figure 2, together with an example on the right: the 5-dimensional sphere function $f\colon \mathbb{R}^5 \to \mathbb{R}, x \mapsto \sum_{i=1}^{5} x_i^2$ centered in the origin is a concrete problem instance that can be subject to optimization. The sphere function is a quadratic function, which belongs to the class of polynomial functions, which, in turn, are a class within the family of continuous functions. However, this example also illustrates that the taxonomy is not used very consistently in the literature: One could also argue that the function f defined above is an instance of the sphere problem (which, for example, may comprise all translated, shifted, and/or scaled versions of f, i.e., functions of the type $g(x) = Cf(x+y)+c$ with $C, c \in \mathbb{R}$ and $y \in \mathbb{R}^5$). One could then argue that the quadratic functions are the class of problems to which the sphere problem belongs, etc. A third interpretation could consider f as the problem instance, the set of all functions $f^d : \mathbb{R}^d \to \mathbb{R}, x \mapsto \sum_{i=1}^{d} x_i^2$ as the sphere problem, the set of translated, shifted, and/or scaled versions of f^d, $d \in \mathbb{N}$, as the problem class and the quadratic functions as a problem family. This discussion also demonstrates that we need to be careful to report and read very precisely which problem instances have been used in the benchmark study.

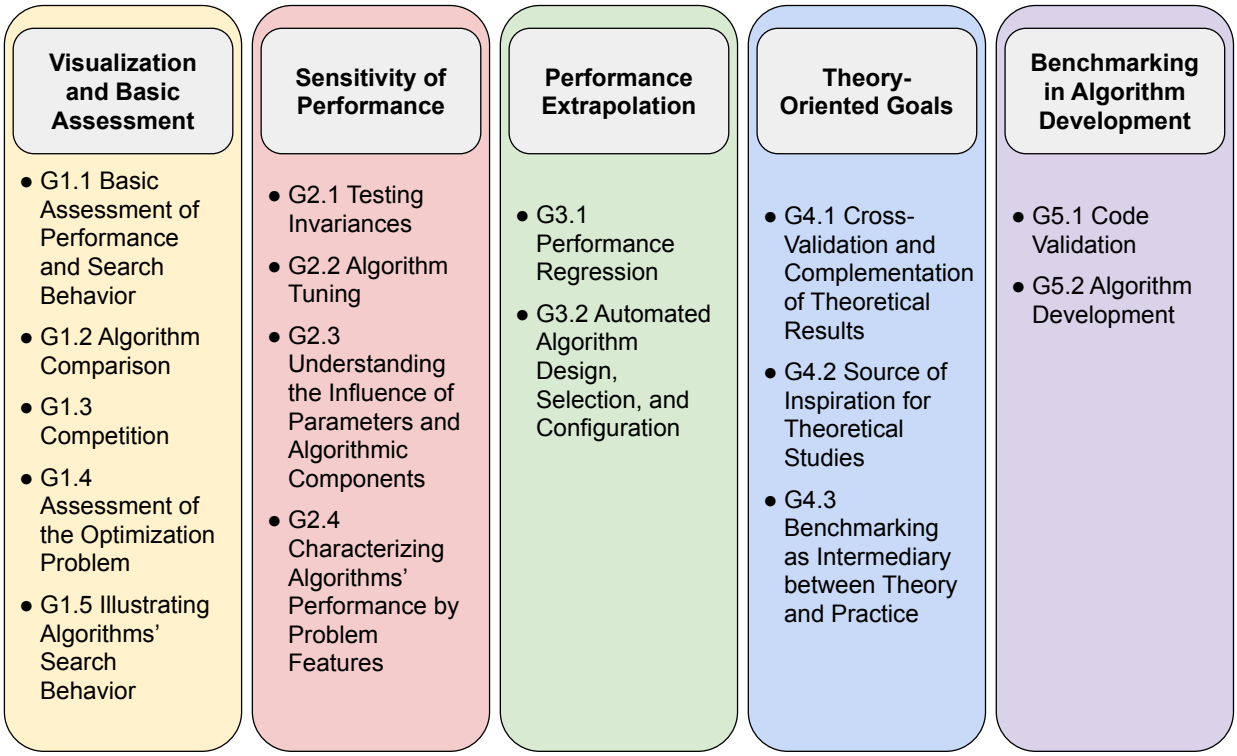

Figure 3: Summary of common goals of benchmark studies.

2 Goals of Benchmarking Activities

The motivation to perform benchmark studies on optimization algorithms is as manifold as the algorithms and the problems that are being used in these studies. In this section, we summarize the most common goals in benchmarking. Figure 3 summarizes these goals. The relevance of these goals can differ within different sub-streams of optimization. Also, the grouping is not necessarily unique, but should be understood as an attempt to find something that represents the benchmarking objectives well within the broader scientific community.

2.1 Visualization and Basic Assessment of Algorithms and Problems

(G1.1) *Basic Assessment of Performance and Search Behavior.*
The arguably most basic research question that one may want to answer with a benchmark study is how well a certain algorithm performs on a given problem. In the absence of mathematical analyses, and in the absence of existing data, the most basic approach to gain insight into the performance is to run the algorithm (ideally several times, if the algorithm or the problem are stochastic) on one or more problem instances, and to observe the behavior of the algorithm in decision and objective space. With this data, one can analyze what a "typical" performance profile looks like, how the solution quality evolves over time, how many evaluations are needed to find solutions of a certain

quality, how much time the execution of the algorithm takes, how robust the performance is, etc. The evaluation criteria are manifold, as we shall discuss in Section 5. But what is inherent to all studies falling into this goal G1.1, is that they are aimed to answer a rather basic question "How well does this particular algorithm perform on this particular problem?".

(G1.2) *Algorithm Comparison.*
The great majority of benchmark studies do not focus on a single algorithm, but rather *compare* the performance and/or the search behavior of two or more algorithms. The comparison of algorithms serves, most notably, the purpose of understanding strengths and weaknesses of different algorithmic approaches for different types of problems or problem instances during the different stages of the optimization process. These insights can be leveraged for algorithm design, selection, and configuration, as we shall discuss below.

(G1.3) *Competition.*
One particular motivation to compare algorithms is to determine a "winner", i.e., an algorithm that performs better than any of its competitors, for a given performance measure and on a given set of problems. Benchmarking is of great value in selecting the most adequate algorithm especially in real-world optimization settings [Beiranvand et al., 2017]. The role of competitions for benchmarking is discussed quite controversially [Hooker, 1995], as competitions may promote algorithms that overstate the importance of the problems that they are tested upon, and thereby create over-fitting. At the same time, however, one cannot neglect that competitions have been an important incentive for contributions to the design and to the development of new algorithmic ideas.

(G1.4) *Assessment of the Optimization Problem.* In many real-world problems, especially those related to bioinformatics, the ground truth is unknown, in other problems it is necessary to deal with limited knowledge, or lack of explicit formulas. In those situations, computer simulations or even physical experiments are required to evaluate the quality of a given solution candidate. In addition, even if a problem is explicitly modelled by a mathematical formula, it can nevertheless be difficult to grasp its structure or to derive a good intuition for how its "fitness landscape" looks like. Similarly, when problems consist of several instances, it can be difficult (or even impossible) to understand in what respect these different instances are alike and in which aspects they differ. Benchmarking simple optimization heuristics can help to visualize the optimization problem and to derive some basic features. Commonly applied solvers in this context are local solvers and non-adaptive random search, but also space-filling sampling strategies like Latin Hypercube Designs (LHDs) [McKay et al., 1979].

(G1.5) *Illustrating Algorithms' Search Behavior.*
Understanding how an optimization heuristic operates on a problem can be difficult to grasp when only looking at the algorithm and problem description. One of the most basic goals that benchmarking has to offer are numerical and graphical illustrations of the optimization process, both in performance space but also the decision space. With these numbers and visualizations, a first idea about the optimization process can be derived. This also includes an assessment of the stochasticity when considering several runs of a randomized algorithm or an algorithm operating upon a stochastic problem. In the same vein, benchmarking offers a hands-on way of visualizing effects that are difficult to grasp from mathematical descriptions. That is, where mathematical expressions are not easily accessible to everyone, benchmarking can be used to illustrate the effects that the mathematical expression describe.

2.2 Sensitivity of Performance with Respect to Algorithm Design and Problem Characteristics

(G2.1) *Testing Invariances.*
Several researchers argue that, ideally, the performance of an optimization algorithm should be

invariant with respect to certain aspects of the problem embedding, such as the scaling of the function values, translation of the function values, or a rotation of the search space (see Hansen [2000] and references therein for a general discussion and Lehre and Witt [2012], Rowe and Vose [2011] for examples formalizing the notion of *unbiased* algorithms).

Whereas certain invariances, such as comparison-baseness, are typically easily inferred from a pseudocode description of the algorithm, other invariances (e.g., invariance with respect to translation or rotation) might be harder to grasp. In such cases, benchmarking can be used to test, empirically, whether the algorithm possesses the desired invariances.

(G2.2) *Algorithm Tuning.*
Most optimization heuristics are configurable, i.e, we are able to adjust their search behavior (and, hence, performance) by modifying their parameters. Typical parameters of algorithms are the number of points kept in the memory, the number of solution candidates that are evaluated in each iteration, parameters determining the distribution from which new samples are generated (e.g., the mean, variance, and direction of the search), the choice of points to keep in memory, and the stopping criterion. Optimization heuristics applied in practice often comprise tens of parameters that need to be tuned.

Finding the optimal configuration of an algorithm for a given problem instance is referred to as offline parameter tuning [Eiben and Jelasity, 2002, Eiben and Smith, 2015]. Tuning can be done manually or with the help of automated configuration tools [Akiba et al., 2019, Bergstra et al., 2013, Olson and Moore, 2016], Benchmarking is a core ingredient of the parameter tuning process. A proper design of experiment is an essential requirement for tuning studies [Bartz-Beielstein, 2006, Orzechowski et al., 2018, 2020]. Parameter tuning a necessary step before comparing a viable configuration of a method with others, as we're disregarding those combinations of parameters, which do not yield promising results.

It can help to shed light on suitable choices of parameters and algorithmic modules. Selecting a proper parameterization for a given optimization problem is a tedious task [Fialho et al., 2010]. Besides the selection of the algorithm and the problem instance, tuning requires the specification of a *performance measure*, e.g., best solution found after a pre-specified number of function evaluations (to be discussed in Sec. 5) and a *statistic*, i.e., number of repeats, which will be discussed in Sec. 7.

Another important concern with respect to algorithm tuning is the robustness of the performance with respect to these parameters, i.e., how much does the performance deteriorate if the parameters are mildly changed? In this respect, parameter recommendations with a better robustness might be preferable over less robust ones, even if compromising on performance [Paenke et al., 2006]. Such as robustness study that later gave rise to a rigorous theoretical assessment of parameter robustness in [Doerr et al., 2019] can be found in [Doerr and Wagner, 2018].

(G2.3) *Understanding the Influence of Parameters and Algorithmic Components.*
While algorithm tuning focuses on finding the best configuration for a given problem, *understanding* refers to the question: *why* does one algorithm perform better than a competing one? Understanding requires additional statistical tools, e.g., analysis of variance or regression techniques. Questions such as "Does recombination have a significant effect on the performance?" are considered in this approach. Several tools that combine methods from statistics and visualization are integrated in the software package Sequential Parameter Optimization Toolbox (SPOT), which was designed for understanding the behavior of optimization algorithms. SPOT provides a set of tools for model based optimization and tuning of algorithms. It includes surrogate models, optimizers and DOE approaches [Bartz-Beielstein et al., 2017].

(G2.4) *Characterizing Algorithms' Performance by Problem Features and Vice Versa.*
Whereas *understanding* as discussed in the previous paragraph tries to get a deep insight into the elements and working principles of algorithms, *characterization* refers to the relationship between

algorithms and problems. That is, the goal is to link features of the problem with the performance of the algorithm(s).

Problem features can be high-level features such as its dimension, the structure of the search space, and other basic properties of the problem. Low-level features of the problem, such as its multi-modality, its separability, or its ruggedness can either be derived from the problem formulation or via an exploratory sampling approach [Kerschke and Trautmann, 2019a,b, Malan and Engelbrecht, 2013, Mersmann et al., 2010, 2011, Muñoz Acosta et al., 2015a,b].

A classical example for a question answered by the characterization approach is how the performance of an algorithm scales with the number of decision variables.

2.3 Benchmarking as Training: Performance Extrapolation

(G3.1) *Performance Regression.*
The probably most classical hope associated with benchmarking is that the generated data can be used to extrapolate the performance of an algorithm for other, not yet tested problem instances. This extrapolation is highly relevant for *selecting* which algorithm to choose and how to *configure* it, as we shall discuss in the next section. Performance extrapolation requires a good understanding of how the performance depends on problem characteristics, the goal described in G2.4.

In the context of machine learning, performance extrapolation is also referred to as *transfer learning* [Pan and Yang, 2010]. It can be done manually or via sophisticated regression techniques. Regardless of the methodology used to extrapolate performance data, an important aspect in this regression task is a proper selection of the instances on which the algorithms/configurations are tested. For performance extrapolation based on supervised learning approaches, a suitable selection of feature extraction methods is another crucial requirement for a good fit between extrapolated and true performance.

(G3.2) *Automated Algorithm Design, Selection, and Configuration.*
When the dependency of algorithms' performance with respect to relevant problem characteristics is known and performance can be reasonably well extrapolated to previously unseen problem instances, the benchmarking results can be used for designing, selecting, or configuring an algorithm for the problem at hand. That is, the goal of the benchmark study is to provide training data from which rules can be derived that help the user choose the best algorithm for her optimization task. These guidelines can be human-interpretable such as proposed in Bartz-Beielstein [2006], Liu et al. [2020] or they can be implicitly derived by AutoML techniques [Hutter et al., 2019, Kerschke and Trautmann, 2019a, Kerschke et al., 2019a, Olson and Moore, 2016].

2.4 Theory-Oriented Goals

(G4.1) *Cross-Validation and Complementation of Theoretical Results.*
Theoretical results in the context of optimization are often expressed in terms of asymptotic running time bounds [Auger and Doerr, 2011, Doerr and Neumann, 2020], so that it is typically not possible to derive concrete performance values from them, e.g., for a concrete dimension, target values, etc. To analyze the behavior in small dimensions and/or to extend the regime for which the theoretical bounds are valid, a benchmarking study can be used to complement existing theoretical results.

(G4.2) *Source of Inspiration for Theoretical Studies.*
Notably, empirical results derived from benchmarking studies are an important source of inspiration for theoretical works. In particular when empirical performance does not match our intuition, or when we observe effects that are not well understood by mathematical means, benchmarking studies can be used to pinpoint these effects, and to make them accessible to theoretical studies, see [Doerr et al., 2019] for an example.

(G4.3) *Benchmarking as Intermediary between Theory and Practice.*
The last two goals, G4.1 and G4.2, together with G1.1 and G1.2 highlight the role of benchmarking as an important intermediary between empirically-oriented and mathematically-oriented subcommunities within the domain of heuristic optimization [Müller-Hannemann and Schirra, 2010]. In this sense, benchmarking plays a similar role for optimization heuristics as *Algorithm Engineering* [Kliemann and Sanders, 2016] does for classical algorithmics.

2.5 Benchmarking in Algorithm Development

(G5.1) *Code Validation.*
Another important aspect of benchmarking is that it can be used to verify that a given code performs as it is expected to. To this end, algorithms can be assessed on problems with known, simple structures. If the algorithm does not behave as expected, some debugging is required.

(G5.2) *Algorithm Development.*
In addition to *understanding* performances, benchmarking is also used to identify weak spots with the goal to develop better performing algorithms. This also includes first empirical comparisons of new ideas to gain first insights into whether or not it is worth investigating further. This can result in a loop of empirical and theoretical analysis. A good example for this is parameter control: it has been observed early on that a dynamic choice of algorithms' parameters can be beneficial over static ones [Karafotias et al., 2015]. This led to the above mentioned loop of evaluating parameters empirically and stimulated theoretical investigations.

2.6 Open Issues and Challenges

Several of the goals listed above require fine-grained records about the traces of an algorithm in performance and/or in the decision space, raising the issue of storing, sharing, and re-using the data from the benchmark studies. Several benchmark environments offer a data repository to allow users to re-use previous experimental results. However, compatibility between the data formats of different platforms is rather weak, and a commonly agreed-upon standard would be highly desirable, both for a better comparability and for a resource-aware benchmarking culture. As long as such standards do not exist, tools that can flexibly interpret different data formats can be used. For example, the performance assessment module IOHanalyzer of the IOHprofiler benchmarking environment [Doerr et al., 2018] can deal with various different formats, including those from the two most widely adopted benchmarking environments in EC, Nevergrad [Rapin and Teytaud, 2018] and COCO [Hansen et al., 2016b].

Coming back to a resource-aware benchmarking culture, we repeat a statement already made in the introduction: two of the most important steps of a benchmarking study are the formulation of a clear research question that shall be answered, and the design of an experimental setup that best answers this question through a well-defined set of experiments. It is often surprising to see how many scientific reports do not clearly explain the main research question that the study aims to answer, (n)or how the reported benchmarking data supports the main claims.

Finally, we note that also the goals themselves undergo certain "trends", which are not necessarily stable over time. The above collection of goals should therefore be seen as a snapshot of what we observe today, some of the goals mentioned above may gain or loose in relevance.

3 Problem Instances

A critical element of algorithm benchmarking is the choice of problem instances, because it can heavily influence the results of the benchmarking. Assuming that we (ultimately) aim at solving real-world problems, ideally, the problem set should be representative of the real-world scenario under investigation, otherwise it is not possible to derive general conclusions from the results of the benchmarking. In addition, it is important

that problem sets are continually updated to prevent the over-tuning of algorithms to particular problem sets.

This section discusses various aspects related to problem sets used in benchmarking. The four questions we address are:

1. What are the desirable features of a good problem set?
2. How do you evaluate the quality of a problem set?
3. What benchmark problem sets are publicly available?
4. What are the open problems in research related to problem sets for benchmarking?

3.1 Desirable Characteristics of a Problem Set

This section describes some of the general features that affect the usefulness of suites of problems for benchmarking, see Whitley et al. [1996] and Shir et al. [2018] for position statements.

(B1.1) *Comprehensive.*
A good benchmark suite should contain problems with a range of difficulties [Olson et al., 2017]. However, what is difficult for one algorithm could be easy for another algorithm and for that reason, it is desirable for the suite to contain a wide variety of problems with different characteristics. In this way, a good problem suite can be used to highlight the strengths and weaknesses of different algorithms. Competition benchmark problems are frequently distinguished based on a few simple characteristics such as modality and separability, but there are many other features that can affect the difficulty of problems for search [Kerschke and Trautmann, 2019b, Malan and Engelbrecht, 2013, Muñoz Acosta et al., 2015b] and the instances in a problem suite should collectively capture a wide range of characteristics.

(B1.2) *Representative.*
At the end of a benchmarking exercise, claims are usually made regarding algorithm performance. The more representative the benchmarking suite is of the class of problems under investigation, the stronger the claim about algorithm performance will hold. The problem instances should therefore include the difficulties that are typical of real world instances of the problem class under investigation.

(B1.3) *Scalable and Tunable.*
Ideally a benchmark set/framework includes the ability to tune the features of the problem instances. For example, it could be useful to be able to set the dimension of the problem, the level of dependence between variables, the number of objectives, and so on.

(B1.4) *Known solutions.*
If the optimal solution(s) of a benchmark problem are known, then it makes it easier to measure exact performance of algorithms in relation to the known optimal performance.

While these properties are *desirable,* we cannot and should not insist on them under all circumstances, e.g., the Low Auto-correlation Binary Sequence (LABS) problem is a quite simple optimization problem for which optimal solutions are not known even for relatively small dimensions, see [Packebusch and Mertens, 2016] for a discussion.

3.2 Evaluating the Quality of a Problem Set

Although it is trivial to assess whether a problem suite provides information on the optimal solution or is tunable, it is not as obvious to assess whether a problem set is comprehensive or representative. In this section, we provide a brief overview of two ways of evaluating the quality of problem sets.

(B2.1) *Feature space.*
Smith-Miles and colleagues introduced the notion of an *instance space* for the mapping of problems to a two-dimensional visualization, e.g., via t-SNE [Maaten and Hinton, 2008] of the instance feature space. Such an approach can be useful for visualising the spread of a set of problem instances across the space of features. Among the first here have been the characterizations of algorithm footprints for the Traveling Salesperson Problem (TSP) [Smith-Miles and Tan, 2012] and for continuous problems [Muñoz and Smith-Miles, 2017].

Škvorc et al. [2020] propose a methodology for visualising numerical search spaces in two dimensions based on Exploratory Landscape Analysis (ELA) features [Mersmann et al., 2011] combined with clustering and a t-distributed stochastic neighbour embedding visualization algorithm.

(B2.2) *Performance space.*
Simple aggregating statistics such as mean and total performance aggregate much information without always enabling the discrimination of two or more algorithms. For example, two algorithms can be very similar (and thus perform comparably) or they might be structurally very different but the aggregated scores might still be comparable. From the area of algorithm portfolios, we can employ ranking-based concepts such as the marginal contribution of individual algorithm to the total portfolio, as well as the Shapley values, which consider all possible portfolio configurations [Fréchette et al., 2016]. Still, for the purpose of benchmarking and better understanding of the effect of design decisions on an algorithm's performance, it might be desirable to focus more on instances that enable the user to tell the algorithms apart in the performance space.

This is where the targeted creation of instances comes into play. Among the first articles that evolved small TSP instances that are difficult or easy for a single algorithm is that by Mersmann et al. [2013], which was then followed by a number of articles also in the continuous domain as well as for constrained problems. Recently. this was extended to the explicit discrimination of pairs of algorithms for larger TSP instances [Bossek et al., 2019], which required more disruptive mutation operators. In combination with the already mentioned algorithm footprints in the feature space, this can be a powerful tool for investigations.

3.3 Available Benchmark Sets

Over the years, competitions and special sessions at international conferences have provided a wealth of resources for benchmarking of optimization algorithms. Some studies on metaheuristics have also made problems available to be used as benchmarks. This section briefly outlines some of these resources, mostly in alphabetical order of their key characteristic. We have concentrated on benchmark problems that are fundamentally different in nature, and that have documentation and code available online. It is also due to our focus on fundamental differences that we typically do not go into the details of configurable instances and parameterized instance generators.

(B2.1) *Artificial discrete optimization problems.*
Subjectively, this area is among those with the largest amount of benchmark sets. Here, many are inspired by problems encountered in the real world, which then have given rise to many fundamental problems in computer science. Noteworthy subareas of discrete optimization are combinatorial optimization, integer programming and constraint programming – and for many of them large sets of historically grown sets of benchmarks exist. For example, the satisfiability and maximum satisfiability communities hold regular competitions[2] and invite the submission of new instances, and large instances sets for the travelling salesman problem[3] and for mixed integer programming[4], as do for many, many more.

[2] http://www.satcompetition.org/, https://maxsat-evaluations.github.io/
[3] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ and http://www.math.uwaterloo.ca/tsp/index.html
[4] https://miplib.zib.de/

In contrast to these instance-driven sets are the more abstract models that define variable interactions at the lowest level (i.e., independent of a particular problem) and then construct an instance based on fundamental characteristics. Noteworthy examples here are (for binary representations) the NK landscapes [Kauffman, 1993] (which has the idea of tunable ruggedness at its core), the W-Model [Weise and Wu, 2018] (with configurable features like length, neutrality, epistasis, multi-objectivity, objective values, and ruggedness), and the Pseudo-Boolean Optimization (PBO) suite of 23 binary benchmark functions by Doerr et al. [2020], which covers a wide range of landscape features and which extends the W-model in various ways (in particular, superposing its transformations to other base problems).

(B2.2) *Artificial real-parameter single-objective problems.*
Benchmark suites have been defined for special sessions, workshops and competitions at both the Association for Computing Machinery (ACM), Genetic and Evolutionary Computation Conference (GECCO) and the Institute of Electrical and Electronics Engineers (IEEE) Congress on Evolutionary Computation (CEC). Documentation and code are available online – for GECCO Black-Box-Optimization-Benchmarking (BBOB)[5], and for CEC[6].

(B2.3) *Artificial constrained real-parameter problems.*
A set of 29 scalable artificial constrained real-parameter problems were defined for a CEC 2017 competition[7].

(B2.4) *Artificial mixed representation problems.*
Benchmark suites combining discrete and continuous variables include mixed-integer NK landscapes [Li et al., 2006], mixed binary and real encoded multi-objective problems [McClymont and Keedwell, 2011], mixed integer problems based on the CEC functions [Liao et al., 2013], and a mixed integer suite based on the BBOB functions (bbob-mixint) with a bi-objective formulation (bbob-biobj-mixint) [Tušar et al., 2019].

(B2.5) *Artificial multi-objective problems.*
Huband et al. [2006] provide a useful review of early multi-objective problem suites and propose the WFG function suite. More recent test suites include the BBOB bi-objective test suite [Brockhoff et al., 2016], the suite defined for the CEC 2020 Special Session on Multimodal Multi-Objective Optimization[8], and the real-world inspired set by Tanabe and Ishibuchi [2020]. In addition, Meneghini et al. [2020] propose the Generalized Position-Distance (GPD) problem generator with parameters for controlling features such as the number of variables and objectives, bias and deceptiveness.

(B2.6) *Black-box optimization problems.*
For all benchmarks listed here, the problem formulation and the instances typically are publicly available, which inevitably leads to a specialization of algorithms to these. The Black-Box Optimization Competition[9] has attempted to address this shortcoming with its single- and multi-objective, continuous optimization problems. Having said this, in 2019, the evaluation code has been made available.

(B2.7) *Dynamic single-objective optimization problems.*
Branke [2001] covers the analysis of Evolutionary Algorithms (EAs) in dynamic environments.

(B2.8) *Dynamic multi-objective optimization problems.*
The 2018 CEC competition[10] introduced dynamic problems that have been based on popular static, multi-objective optimization problems, such as the DTLZ and the ZDT families. Complementary

[5] https://coco.gforge.inria.fr
[6] https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark
[7] https://github.com/P-N-Suganthan/CEC2017-BoundContrained
[8] https://github.com/P-N-Suganthan/2020-Multimodal-Multi-Objective-Benchmark
[9] https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/
[10] http://homepages.cs.ncl.ac.uk/shouyong.jiang/cec2018/cec2018.html

to this is the slightly older survey [Raquel and Yao, 2013] that covers test problems, performance measures and optimization approaches.

(B2.9) *Expensive optimization problems.*
The GECCO 2020 Industrial Challenge provides a suite of discrete-valued electrostatic precipitator problems with expensive simulation-based evaluation[11].

(B2.10) *Multimodal optimization (niching).*
Benchmark problem sets for niching include the GECCO and CEC competitions on niching methods for multimodal optimization problems[12] and the single-objective multi-niche competition problems[13].

(B2.11) *Noisy optimization.*
A benchmark collection covering various of the here-mentioned categories is the Nevergrad platform [Rapin and Teytaud, 2018],[14]. The original version had a strong focus on noisy problems, but the platform covers today discrete, continuous, mixed-integer problems with and without constraints, with and without noise, explicitly modelled problems and true black-box problems, etc.

(B2.12) *Problems with interdependent components.*
While much research tackles combinatorial optimization problems in isolation, many real-world problems are combinations of several [Bonyadi et al., 2019]. For example, the pit to port optimization in the mining industry consists of multiple interdependent components from scheduling, packing and routing. The Travelling Thief Problem [Bonyadi et al., 2013] has been created as an academic platform to systematically study the effect of the interdependence, and the 9 720 instances [Polyakovskiy et al., 2014][15] vary in four dimensions. A number of single- and multi-objective as well as static and dynamic extensions of the Travelling Thief Problem have been proposed since then [Sachdeva et al., 2020].

(B2.13) *Real-world discrete optimization.*
The GECCO competition on the optimal camera placement problem (OCP) and the unicost set covering problem (USCP) include a set of discrete real-world problem instances[16]. The Mazda Benchmark Problem is a scalable, multi-objective, discrete-valued, constrained benchmark problem based on real-world car structure design[17].

(B2.14) *Real-world numerical optimization.*
A set of 57 single-objective real-world constrained problems were defined for competitions at a number of conferences[18]. The Smart Grid Problems Competitions provide testbeds for real-world problems in the energy domain[19], whereas the Game Benchmark for EAs [Volz et al., 2019] provide a suite of test functions inspired by game-related problems[20].

3.4 Open Issues

What gaps do we see in terms of research on problem sets for benchmarking?

- Real-world: Besides the CEC real-world numerical optimization and the Mazda Benchmark Problem there seems to be a lack of real-world relevant benchmark problems, especially for continuous optimization. When there are some proper real-world problems available (e.g., data sets for combinatorial

[11] https://www.th-koeln.de/informatik-und-ingenieurwissenschaften/gecco-challenge-2020_72989.php
[12] http://epitropakis.co.uk/gecco2020/
[13] https://github.com/P-N-Suganthan/CEC2015-Niching
[14] https://github.com/facebookresearch/nevergrad
[15] https://cs.adelaide.edu.au/~optlog/research/combinatorial.php
[16] http://www.mage.fst.uha.fr/brevilliers/gecco-2020-ocp-uscp-competition/
[17] http://ladse.eng.isas.jaxa.jp/benchmark/
[18] https://github.com/P-N-Suganthan/2020-RW-Constrained-Optimisation
[19] http://www.gecad.isep.ipp.pt/ERM-competitions/home/
[20] http://www.gm.fh-koeln.de/~naujoks/gbea/gamesbench.html

problems, or the CEC problems mentioned), they often are single-shot optimizations, i.e., only a single run can be conducted, which then makes it difficult to retrieve generalizable results. A recent push towards a collection and characterization has been made with a survey[21] by the Many Criteria Optimization and Decision Analysis (MACODA) working group.

- Availability: the availability of diverse instances, source code (of fitness functions, problem generators, but also of algorithms) leaves much to be desired. Ideal are large collections of instances, their features, algorithms, and their performance—the Algorithm Selection Library (ASlib)[22] [Bischl et al., 2016] has such data, although for a different purpose. As a side-effect, these (ideally growing) repositories offer a means against the reinvention of the wheel and the benchmarking against so-called "well-established" algorithms that are cited many times – but maybe just cited many times because they can be beaten easily.

- Watching our claims: how can one feed the intended outcome "my approach beats your approach" back to the instances? Or to paraphrase this: what conclusions can one actually attempt to draw, given the performance is always "modulo the given test suite"?

- It is an advantage of test problem suites that they can provide an objective means of comparing systems. However, there are also problems related to test poblem suites: Whitley et al. [2002] discuss the potential disadvantage that systems can become overfitted to work well on benchmarks and therefore that good performance on benchmarks does not generalize to real-world problems. Fischbach and Bartz-Beielstein [2020] list and discuss several drawbacks of these test suites, namely: (i) problem instances are somehow artificial and have no direct link to real-world settings; (ii) since there is a fixed number of test instances, algorithms can be fitted or tuned to this specific and very limited set of test functions; (iii) statistical tools for comparisons of several algorithms on several test problem instances are relatively complex and not easily to analyze.

- Handling noise: Branke et al. [2001] discuss strategies how to cope with noise. Jin and Branke [2005] present a good survey.

4 Algorithms

As discussed in Section 2, most benchmarking studies aim at analyzing or understanding algorithms' behavior on different types of problem instances. The selection of the portfolio of algorithms to be tested is therefore an important step in benchmarking. We present in this section a classification of common query-based optimization heuristics. We illustrate this classification by listing examples from the family of optimizers for continuous problems.

As we shall see in the description of the algorithm classes, a characteristic that is common to all of them is a high level of parametrization of the algorithms, giving rise to the question which algorithm instances to compare through the benchmark study. We will argue in Section 4.2 that (hyper-)parameter optimization is an important ingredient of most benchmark studies – the use of automated configuration tools is therefore highly recommended.

Section 4.3 summarizes recommendations for the selection of the portfolio of algorithms. A discussion of open issues can be found in Section 4.4.

4.1 Algorithm Families

Extending the work of Stork et al. [2018], we discuss five different families of algorithms:

(a) One-Shot Optimization algorithms

[21] https://sites.google.com/view/macoda-rwp/home
[22] https://github.com/coseal/aslib_data

Table 1: Properties of algorithms determining their family

	One-shot	**Hill-climbers**	**Trajectory**	**Population**	**Surrogates**
Adaptive sampling		✓	✓	✓	✓
Single solution		✓			
Greedy selection		✓			
Single- or multiple solutions			✓		
Exploration			✓	✓	✓
Self-adaptive acceptance functions			✓		
Search-space partitioning			✓		
Multiple solutions concurrently				✓	
Cooperative or competitive selection				✓	
Surrogates for objective	(✓)				✓
Surrogates for candidate distributions					✓

(b) Hill-climbing algorithms

(c) Trajectory algorithms

(d) Population-based algorithms

(e) Surrogate-based algorithms

Arguably these families are given in an order with increasing complexity. A summarized representation of the algorithm concepts that leads to the association to an algorithm family can be found in Table 1.

One-Shot Optimization Algorithms When dealing with problems that are very costly to evaluate, or when simply working under high time pressure, the possibility for a sequential optimization may not be given. In such cases, the decision maker has to resort to *one-shot optimization algorithms,* which typically sample the decision space in a space-filling manner. Several different criteria to measure the "space-fillingness" exist, and each one of them implies different one-shot designs. The debate about which variant to favor under which circumstances is very closely related to similar discussion around the DOE (see Section 7 for a short discussion).

It may be important to note that quasi-random designs such as LHD and low-discrepancy point sets [Dick and Pillichshammer, 2010, Matoušek, 2009] are often (but not always!) found to be superior over uniform sampling, see [Bergstra and Bengio, 2012, Bousquet et al., 2017, Cauwet et al., 2019] for related theoretical works. It is worth noting also that even when the prior distribution of the optimum is exactly known, the best best one-shot distribution may be a different one [Meunier et al., 2020b] - a phenomenon related to the "Stein phenomenon" [James and Stein, 1961, Stein, 1956].

We also classify as one-shot optimization algorithms those strategies in which the final decision may differ from the evaluated points. That is, in this setting, the user evaluates n alternatives, but is then free to use

the information provided by these n points to decide for an alternative that is not included in this set. The final decision can be derived by optimizing a surrogate built on top of the n points [Bossek et al., 2020a], or by simply averaging some of the best points [Meunier et al., 2020a].

Thus, summarizing this paragraph, the distinguishing property of one-shot optimization algorithms is the fact that no adaptive sampling is permitted, i.e., all points to be evaluated by the algorithm have to be decided on independently of the quality of other search points.

Hill-Climbing Algorithms Besides deterministic or stochastic hill-climbing algorithms, this family covers gradient-based algorithms. The concept of exploration is most often not part of these algorithms, which means that escaping local optima is most likely not possible. Consequently, hill-climbers are often embedded in more sophisticated global search strategies to enable fast convergence in a local optimum. Examples for hill-climbing algorithms include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [Shanno, 1970], conjugate gradients [Fletcher, 1976], and Nelder-Mead [Nelder and Mead, 1965]. Hill-climbers typically start with a single solution and employ a greedy search strategy. The performance of these algorithms relies heavily on the starting point, which is, due to the absence of individual operators, selected at random or by utilizing domain knowledge. Therefore, restarts with different (random or quasi-random) starting points are a common approach to use hill-climbers effectively. In comparison to the range of the search space, new candidate solutions are searched within the vicinity of the current solution. Gradient-based methods compute or approximate the gradients of the objective function to find the potentially best direction and step size for the improvement of the solution. The step size is the most critical control parameter, as it controls the speed of convergence. Using an adaptive strategy to change the step size online is considered as state of the art nowadays, e.g., with the famous $1/5$-th success rule, see [Devroye, 1972, Rechenberg, 1994, Schumer and Steiglitz, 1968].

Main characteristics of algorithms of this family:

- comparison of candidates in a greedy manner
- single-solution based

Trajectory Algorithms Algorithms of this family extend the idea of hill-climbing algorithms, such that they are aware that multiple places of interest exist. Exploration is added as a concept of this family and the search-space is systematically evaluated by one or more candidate solutions. Algorithms of this family consider operators to guide the search globally in specific directions. This guidance can be divided into two different sub-strategies, i.e., the *exploring trajectory*, which utilizes parameterized acceptance functions, and the *systematic trajectory*, which separates the search space into different areas. An example for the *exploring trajectory* subfamily is the well-known Simulated Annealing (SANN) introduced by Kirkpatrick et al. [1983]. SANN implements an acceptance function that allows the escape of local optima with a certain probability, i.e., the acceptance of a worse solution, which enables the balancing of exploration and exploitation. Modern SANN variants implement self-adaptive acceptance functions, which allows, e.g., alternating phases of exploitation and exploration. The subfamily of *systematic trajectory* algorithms use the information of former iterations of the search to divide the search space into partitions. Partitions can be marked for either interesting areas or avoidable areas. One outstanding example algorithm of this family is *Tabu Search*, introduced by Glover [1989], which maintains a list of visited points of the search space to avoid revisiting these areas in future iterations.

Main characteristics of algorithms of this family:

- exploration
- acceptance-functions or search-space partitioning
- single- or multiple candidate solutions

Population-based Algorithms Population-based algorithms utilize several candidate solutions concurrently. The generation and selection strategies differ from those of the Hill-climbers and Trajectory algorithms, as a whole population of candidates is employed. Examples are Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995, Shi and Eberhart, 1998], ant colony optimization for continuous domains ($ACO_{\mathbb{R}}$) [Dorigo et al., 2006, Socha and Dorigo, 2008], and several EAs. EAs can arguably be considered as the state of the art of the population-based algorithm family, as their search concepts are fundamental for this field of algorithms. EAs mimic the natural idea of evolution, reproduction, and the selection concept of survival of the fittest. Candidate solutions can take the role of parent individuals, which can be mate and recombined to create offspring individuals. To improve the solutions, the population is evolved over several generations, which iteratively repeats variation by recombination and mutation, evaluation, and selection. EAs generally have several tunable parameters, e.g., the selection pressure, variation step size, or mutation, and recombination rate. For an overview of the field of EAs, we refer to Bäck et al. [1997], Eiben and Smith [2015], for an overview of existing methods to tune EAs, we refer to Eiben and Smit [2011], Hutter et al. [2019].

Within the family of population-based algorithms, Estimation of Distribution Algorithms (EDAlgos) come into focus. They employ mathematical or statistical models of the underlying candidate distributions. One famous example is the CMA-ES [Hansen et al., 2003]. EDAlgos introduce several additional control parameters as well. The CMA-ES for example adapts parameters over time, evaluating prior iterations.

Main characteristics of algorithms of this family:

- multiple candidate solutions regarded concurrently
- cooperative or competitive generation and selection mechanisms

Surrogate-based Algorithms This family distinguishes two subfamilies of algorithms using surrogates. Surrogate-based algorithms, which utilize a surrogate model of the objective function for variation and selection, and the Surrogate-assisted algorithms, which employ surrogate models to improve the selection of candidate solutions in population-based algorithms. For each group, the quality of the surrogate model is crucial for the efficiency of the algorithm. Assuming that different modeling techniques have different advantages and disadvantages, it is difficult to choose the best fitting model. Famous examples of surrogate model techniques are regression models (linear, quadratic, polynomial), Gaussian processes, regression trees, artificial neural networks, and symbolic regression models. Recent research focuses on ensembles of surrogate models, to exploit advantages of different models at the same time, to compute predictors superior to a (or any) single predictor. A well-known example of surrogate-based optimization is the Efficient Global Optimization (EGO) algorithm, see [Jones et al., 1998]. Several control parameters are introduced, e.g., the size and sampling strategy used to generate the initial solutions, the employed model technique, the infill-criterion for the candidate selection, the optimizer that searches the surrogate for these candidate solutions, and the adaptation technique to converge from an exploratory infill criterion to an exploitative one.

Main characteristic of algorithms of this family:

- one ore more surrogate models to approximate the objective function or sample candidate solutions more efficiently

4.2 Hyperparameter Handling in Benchmarks

All previously discussed families of algorithms have in common to have one or more important control parameters. To fairly compare the performance of algorithms and judge their efficiency, it is crucial to avoid bad parameter configurations by tuning the algorithms properly [Beiranvand et al., 2017, Eiben and Smit, 2011]. Even a well-working parameter configuration for a certain setup, i.e., a fixed budget, may work comparably worse on a significantly different budget. Several tools developed for automatic parameter configuration are available, e.g., iterated racing (irace) [López-Ibáñez et al., 2016], Iterated Local Search in Parameter Configuration Space (ParamILS) [Hutter et al., 2009], SPOT [Bartz-Beielstein et al., 2005], Sequential Model-based Algorithm Configuration (SMAC) [Hutter et al., 2011], GGA [Ansótegui et al., 2015],

and hyperband [Li et al., 2017] to name a few. As manual tuning can be biased, especially for algorithms unknown to the experimenter, automated tuning is state of the art and highly recommended. Giving rise to the large amount of research in the field of automated algorithm configuration and hyperparameter optimization, there exist several related benchmarking platforms, like the algorithm configuration library (ACLib) [Hutter et al., 2014] or the hyperparameter optimization library (HPOlib) [Eggensperger et al., 2013], which deal particularly with this topic.

As mentioned in Section 2 under goal (G2.2), the robustness of algorithms with respect to their hyperparameters can be an important characteristic for users, in which case this question should be integrated into (or even be the subject of) the benchmarking study.

4.3 Algorithm Portfolio Selection Guideline

Dependent on the goal of the benchmark study to perform, several of the following recommendations should be taken into consideration:

- Add a simple (e.g., random) search as a baseline comparison.
- Add a state of the art algorithm which is known to perform well on the chosen problem instances.
- Compare your algorithm to the most recent variants or versions of the competing algorithms.
- Do not add to many algorithms, as interpretation and visualization of results can easily become very unhandy.

4.4 Discussion and Open Issues

- Each new family (or each new algorithm) introduce new concepts or methods to overcome difficulties regarding the search behaviour or the selection of candidates and consequently introduces new control parameters. The algorithms efficiency rely heavily on the parameter choices, which may depend on the test instances to solve. Consequently, automated algorithm tuning is highly recommended for nearly all benchmarking scenarios.
- If a benchmark study has the goal to compare different algorithms (G1.2), e.g., a new version or implementation of an algorithm with the preceding version or simply compare different algorithms, it is important to take the most recent versions of the algorithms to gain unbiased results. Additionally, a baseline comparison, e.g., a random search, help to quickly analyse the performance of all algorithms.
- Sometimes, e.g., when historical data is available, it might be beneficial or even necessary to initialize (i.e., to seed) the algorithms with one or more candidate solutions. As discussed above, algorithms starting points, e.g., for local search algorithms, have an impact on the performance of the algorithms. Some implementations of certain algorithms may not provide a suitable interface to properly initialize the algorithms with previously available or computed starting points. When comparing different algorithms (e.g., G1.2 and G1.3) or analyze the algorithms search behavior (G1.1), it is important to let all candidate algorithms use the same starting points.
- To retrieve unbiased results, it is useful to let not the algorithms itself chose their starting points, except when the goal is to analyze the algorithms seeding itself, e.g., when the selected seeding method is probably beneficial for the problem to solve. In this situation, the results and their interpretation have to be taken with care.
- Not all algorithms support the configuration of the same stopping criteria, which may influence the search as well [Beiranvand et al., 2017]. The experimenter has to consider that when it comes to the interpretation of the results.
- For (random) seed handling and further reproducibility handling we refer to Section 9 Reproducibility.

- Programming paradigms change and the complexity of algorithms grows. For example, ensemble methods and deep learning gain more and more importance [Bartz-Beielstein and Zaefferer, 2017, Goodfellow et al., 2018]. Compare a classical, gradient-based optimization method from the 1960s (or a simple (1+1)-Evolution Strategy (ES)) with a modern complex ensemble-based optimization algorithm that implements a surrogate model based on deep learning. This not only pushes the need for computational resources to conduct benchmark studies but also has important implications on the replicability and the reusability of the results (see also Section 9 for a discussion).

5 How to Measure Performance?

5.1 General Concepts

The performance of algorithms depends on several factors, with solution quality and consumed budget being the most obvious. In fact, benchmarking usually examines algorithms by asking one of two questions: "How fast can the algorithms achieve a given solution quality?" or "What solution quality can the algorithms achieve with a given budget?" Since budget and solution quality mutually influence each other, good algorithms ideally optimize a Multi-Objective Optimization (MOO) problem, where the consumed runtime and attained solution quality are two of the objectives [Bossek et al., 2020b, Weise et al., 2014]. Robustness is a third goal, i.e., to achieve good quality fast and with a low variance of both time and quality. Depending on the application, time is measured in different ways. The most intuitive way is to measure it by means of clock or CPU time. As such, it is the default in several combinatorial optimization problems like solving TSP [Kerschke et al., 2018b] or Boolean Satisfiability (SAT) problems [Xu et al., 2008]. However, as CPU time is highly sensitive to a variety of external factors – such as hardware, programming language, work load of the processors – results from experiments that relied on CPU time are much less replicable and thus hardly comparable. In an attempt of mitigating this issue, Johnson and McGeoch [2002], as well as Weise et al. [2014] proposed a normalized time, which is computed by dividing the runtime of an algorithm by the time a standardized implementation of a standardized algorithm requires for the same (or at least a comparable) problem instance.

The alternative way of measuring time are Function Evaluations (FEs), i.e., the number of generated and fully evaluated candidate solutions. In fact, in case of query-based optimization, e.g., in classical continuous optimization, this machine-independent metric is *the* way of measuring algorithmic performances [Hansen et al., 2016a] – although from the perspective of actual clock time, FEs are less comparable as different FEs might be of different time complexity [Weise et al., 2014]. In such cases, counting algorithm steps in a domain-specific method with a higher resolution may be useful. For instance, Weise et al. [2014] proposed to count the number of distance evaluations on the TSP, and in Hains et al. [2013] bit flips are counted on the Maximum Satisfiability (MAX-SAT) problem. However, within the EC community, FEs are clearly the gold standard for measuring performance of algorithms.

Thus, from a practical point of view, both options have a reason for existence. If the budget is given by means of clock time – e.g., models have to be trained until the morning of the next working day, or they need to be adjusted within seconds in case of predictions at the stock market – then results that rely on clock or CPU time are more meaningful. On the other hand, in case single FEs are expensive – for instance, in case every single FE corresponds to a physical experiment or a cost-intensive numerical simulation – the amount of required FEs is of much higher importance than the corresponding clock time .

Noticeably, many papers in the EC community also use *generations* as machine-independent time measures. However, this might not be a good idea, because the exact relationship between FEs and generations is not always clear, which makes results hard to compare with, e.g., local search algorithms.

As mentioned at the beginning of this section, once a quality metric for candidate solutions is defined, one usually faces the decision to either measure the quality that can be achieved with a certain computational budget *or* the budget required to achieve a certain quality. These scenarios correspond to vertical and horizontal cuts in a performance diagram as discussed by Hansen et al. [2012] and Finck et al. [2015], respectively (see Figure 4). The fixed-budget scenario has the advantage that the results are always well-

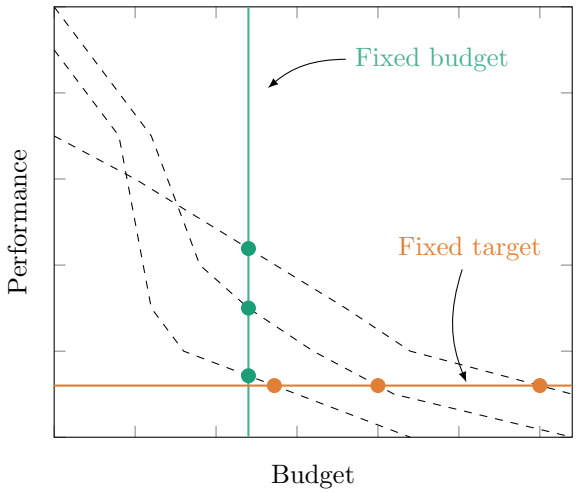

Figure 4: Visualization of a fixed-budget perspective (green line) and a fixed-target perspective (orange line) inspired by Figure 4 in [Hansen et al., 2012]. Dashed lines show three exemplary performance trajectories.

defined and any real computation always has a limited budget. Budgets based on consumed clock time are harder to compare or reproduce [Johnson, 2002a]. Horizontal cuts, i.e., fixing the desired solution quality, allows drawing conclusions that are easier to interpret; statements such as "algorithm instance b is ten times faster than algorithm instance a in solving this problem" are likely much more tangible compared to "the solution quality achieved by algorithm instance b is 0.2% better than the one of algorithm instance a" [Finck et al., 2015, Hansen et al., 2012].

Most optimization methods, ranging from metaheuristics over problem-specific local search to branch and bound algorithms are so-called anytime algorithms [Boddy and Dean, 1989]. An anytime algorithm starts with one solution and then tries to iteratively generate better solutions. This means that its performance does not correspond to a singular point, but instead to an entire curve in the time-quality diagrams. For different time budgets or targeted objective values, different algorithms may yield better results or shorter runtimes, respectively.

5.2 Quality Metrics

Single-objective optimization In single-objective optimization scenarios, the quality metrics are quite natural: the height/fitness in continuous single-objective optimization, the tour length in TSP, the accuracy of a classification algorithm in a machine learning task, or the number of ones in a binary bit string in case of OneMax. However, interpreting these objective values on their own is usually quite difficult and also very specific to the respective problem instance. Instead, one could ideally try to use more intuitive and less problem-dependent alternatives [Johnson, 2002a, Talbi, 2009].

If the optimal solution to a problem instance is known, the (relative) excess over the optimal solution quality could be used. Doing so may focus research on instances that can be solved exactly, i.e., the opposite type of problems for which metaheuristics have initially been designed for [Johnson, 2002a]. Alternatively, the lower bound for the optimal objective value could be used for normalizing the results. For instance, in case of the TSP, results are often compared to the Held-Karp lower bound [Johnson, 2002a]. As *absolute* differences are very specific to the scaling of the problem's objective values, it is highly recommended to rather look at the *relative* excess over the optimal solution – something that has been common practice in solving TSPs for decades [Christofides, 1976].

Another alternative is to use results of a clearly specified and ideally simple heuristic for normalization [Johnson, 2002a]. The relative excess over the best-known solution is also often reported. This is

typical in the Job Shop Scheduling domain, as shown by a large number of references listed in [Weise, 2019]. However, this requires an exact knowledge of the related work and may be harder to interpret later in the future [Johnson, 2002a]. For some problems, reference solutions may be available and the excess over their quality can be reported.

Multi-objective optimization In MOO, one is facing the additional challenge of ranking solutions based on their Pareto dominance. To determine the performance on a set of multi-objective solutions, it must first be decided whether the metric should be computed only on the (mostly non-dominated) solutions of the final population or on all solutions that were archived during the complete run. Here, performance indicators must cover not only the quality of the solutions themselves (usually expressed via some form of proximity metric to the true Pareto front), but also their distribution along the Pareto front [Bosman and Thierens, 2003, Liu et al., 2017]. In recent years the diversity of solutions, both in search and objective space, have become another important aspect when dealing with MOO problems [Kerschke et al., 2019b, Liefooghe et al., 2018, Paquete et al., 2004]. Another goal is to achieve consistency, i.e., to avoid situations where the obtained Pareto front contains outliers far away from the true front [Hadka and Reed, 2012]. As a result, several different metrics have emerged [Riquelme et al., 2015, Talbi, 2009].

A large hypervolume or S-metric [Knowles and Corne, 2002, Zitzler, 1999] indicates closeness to the true Pareto Front and somewhat a good spread [Santos and Xavier, 2018], however, the resulting values themselves are not human-interpretable. The Inverted Generational Distance (IGD) [Bezerra et al., 2017, Coello Coello and Reyes Sierra, 2004] measures both diversity and convergence, as a good approximated front should have points near each element of the reference set. IGD+ is an improved version of IGD which is weakly Pareto compliant [Bezerra et al., 2017, Ishibuchi et al., 2015]. In addition, IGDX [Zhou et al., 2009] allows to measure proximity of solutions in the search space. The additive epsilon indicator (ε_+-indicator) [Hadka and Reed, 2012] can discover the lack of consistency and it provides human-interpretable values. Popular is also the $R2$-indicator [Hansen and Jaszkiewicz, 1998, Knowles and Corne, 2002]. Recent surveys discussing many-objective optimization are presented by Chand and Wagner [2015] and Li et al. [2015].

Constraint optimization Under constraint optimization, a solution is either feasible or not, which is decided based on a set of constraints. Here, the absolute violations of each constraint can be summed up as a performance metric [Hellwig and Beyer, 2019, Kumar et al., 2020].

5.3 Aggregating Metrics over Multiple Runs

Many optimization algorithms are nowadays randomized search heuristics and as such their performances will vary if the experiment is repeated, i.e., if the algorithm is executed again with the same setup. Therefore, it is common to use performance metrics that aggregate the results of several (ideally independent) runs to provide reliable estimates of the algorithm performance.

Location: Measures of Central Behaviors In case of a fixed budget (i.e., the vertical cut approach) it is common practice to aggregate the solution qualities using the arithmetic mean. Alternatively, if one is interested in a more robust metric, the median is typically used [Weise et al., 2014]. Fleming and Wallace [1986] recommend using the geometric mean to summarize normalized benchmark results, because the arithmetic mean then leads to mistaken conclusions. In scenarios where the primary goal is to achieve a desired target quality (in line with the horizontal cut approach), it is potentially necessary to aggregate runs of which not all have succeeded in achieving the desired target value within a fairly generous budget T as alternative termination criterion.

In the latter scenario, two to three metrics are mostly used for aggregating performances across algorithm runs, and we will list them below.

The gold standard in (single-objective) continuous optimization is the Expected Running Time (ERT) [Auger and Hansen, 2005, Price, 1997], which computes the ratio between the sum of consumed

budget across all runs and the number of successful runs [Hansen et al., 2012]. Thereby, it estimates the average running time an algorithm needs to find a solution of the desired target quality (under the assumption of independent restarts every T time units until success).

Inc other optimization domains, like TSP, SAT, etc., the *Penalized Average Runtime (PAR)* [Bischl et al., 2016, Kerschke et al., 2018a] is more common. It penalizes unsuccessful runs with a multiple of the maximum allowed budget – penalty factors ten (PAR10) and two (PAR2) are the most common versions – and afterwards computes the arithmetic mean of the consumed budget across all runs. The *Penalized Quantile Runtime (PQR)* [Bossek et al., 2020b, Kerschke et al., 2018a] works similarly, but instead of using the arithmetic mean for aggregating across the runs, it utilizes quantiles – usually the median – of the (potentially penalized) running times. In consequence, PQR provides a robust alternative to the respective PAR scores.

Spread and Reliability A common measure of reliability is the *estimated success probability*, i.e., the fraction of runs that achieved a defined goal. Bossek et al. [2020b] propose a multi-objective view on performance measures, combining the probability of success and the average runtime. The quotient *SP* of the mean time consumed in successful runs and the success ratio can also be computed [Hellwig and Beyer, 2019].

As measures of dispersion of a given single performance metric, statistics like standard deviations as well as quantiles are used, whereas the latter are more robust.

For constraint optimization, a *feasibility rate* (FR) [Kumar et al., 2020, Wu et al., 2017] is defined as the fraction of runs discovering at least one feasible solution. The number of constraint violated by the median solution [Kumar et al., 2020] and the mean amount of constraint violation over the best results of all runs [Hellwig and Beyer, 2019] can also be used.

5.4 Open Issues

Although each of the different optimization domains has established its preferable performance measure, research in this field is still facing open issues. For instance, so far performance is usually measured either using a fixed budget or a fixed target. However, as the results then depend on the magnitude of the chosen budget or target quality, one should rather compare algorithms by means of their anytime behavior [Bossek et al., 2020c, Jesus et al., 2020, Weise et al., 2014]. But even without considering the aforementioned anytime aspect, the current performance metrics have to be extended in the future. Aside from facing challenges like measuring quality *and* time simultaneously, we also have to integrate costs for violating constraints (constraint optimization), quantify variation or uncertainty (robust/noisy optimization), measure the spread across the local optima (multimodal optimization), or capture the population's proximity to the problem's local and/or global optima in search *and* objective space (MOO).

6 How to Analyze Results?

6.1 Three-Level Approach

Once the performance measure for the algorithm's performance is selected by the user and all data related to it is collected in experiments, the next step is to analyse the data and draw conclusions from it. From the detailed characterization of possible benchmark goals in Section 2, we will focus on goals (G1.2) and (G1.3), i.e., algorithm comparison and competition of several algorithms. Therefore, we will consider:

- single-problem analysis and
- multiple-problem analysis.

In both scenarios, multiple algorithms will be considered, i.e., following the notation introduced in Section 2, there are at least two different algorithm instances, say, a_j and a_k from algorithm A or at least two different

algorithm instances $a_j \in A$ and $b_k \in B$, where A and B denote the corresponding algorithms. Single-problem analysis is a scenario where the data consists of multiple runs of the algorithms on a single problem instance $\pi_i \in \Pi$. This is necessary because many optimization algorithms are stochastic in nature, so there is no guarantee that the result will be the same for every run. Additionally, the path leading to the final solution is often different. For this reason, it is not enough to perform just a single algorithm run per problem, but many runs are needed to make a conclusion. In this scenario, the result from the analysis will give us a conclusion which algorithm performs the best on that specific problem.

Otherwise in the case of multiple-problem analysis, focusing on (G1.2), we are interested in comparing the algorithms on a set of benchmark problems. The best practices of how to select a representative value for multiple-problem analysis will be described in Section 5.

No matter of what we are performing, i.e., single-problem or multiple-problem analysis, the best practices to analyse the results of the experiments suggest that the analysis can be done as a three-level approach, which consists of the following three steps:

1. Exploratory Data Analysis (EDA)
2. Confirmatory Analysis
3. Relevance Analysis

This section focuses on analyzing the empirical results of an experiment using descriptive, graphical, and statistical tools, which can be used for the three-level approach for analysis. More information about various techniques and best practices to analyze the results of experiments can be found in Crowder et al. [1979], Golden et al. [1986], Barr et al. [1995], Bartz-Beielstein et al. [2004], Chiarandini et al. [2007], García et al. [2009], Bartz-Beielstein et al. [2010], Derrac et al. [2011], Eftimov et al. [2017], Beiranvand et al. [2017]. Mersmann et al. [2010], and more recently Kerschke and Trautmann [2019a], present methods based on ELA to answer two basic questions that arise when benchmarking optimization algorithms. The first one is: which algorithm is the 'best' one? and the second one: which algorithm should I use for my real world problem? In the following, we summarize the most accepted and standard practices to evaluate the considered algorithms stringently. These methods, if adhered to, may lead to wide acceptance and applicability of empirically tested algorithms and may be a useful guide in the jungle of statistical tools and methods.

6.2 Exploratory Data Analysis

6.2.1 Motivation

Exploratory Data Analysis (EDA) is an elementary tool that employs descriptive and graphical techniques to better understand and explore empirical results. It must be performed to validate the underlying assumptions about the distribution of the results, e.g., normality or independence, before implementing any statistical technique that will be discussed in Section 6.3.

We recommend starting with EDA to understand basic patterns in the data. It is useful to prepare (statistical) hypotheses, which are the basis of confirmatory analysis. In EDA, visual tools are preferred, whereas confirmatory analysis is based on probabilistic models. EDA provides a flexible way to analyse data without preconceptions. Its tools stem from descriptive statistics and use an inductive approach, because in the beginning, there is no theory that has to be validated. One common saying is "let the data speak", so data suggest interesting questions, e.g., unexpected outliers might indicate a severe bug in the algorithm. EDA is a very flexible way to generate hypotheses, which can be analyzed in the second step (confirmatory analysis). Although EDA might provide deeper understanding of the algorithms, it does not always provide definitive answers. Then, the next step (confirmatory analysis) is necessary. And, there is also the danger of overfitting: focussing on very specific experimental designs and results might cause a far too pessimistic (or optimistic) bias. Finally, it is based on experience, judgement, and artistry. So there is no standard cookbook available, but many different recipes.

The following are the key tools available in EDA. It can provide valid conclusions that are graphically presented, without requiring further statistical analysis. For further reading about EDA, the reader is referred to [Tukey, 1977].

6.2.2 The Glorious Seven

Descriptive statistics include the mean, median, best and worst (minimum and maximum, respectively), first and third quartile, and standard deviation of the performance measures of the algorithms. These seven statistics measure the central tendency and the variability of the results. Note, they might be sensitive to outliers, missing, or biased data. Most importantly, they do not provide a complete analysis of the performance, because they are based on a very specific data sample. For example, mean and standard deviation are affected by outliers, which might exist because of an algorithms' poor runs and variability. Both can be caused by an inadequate experimental design, e.g., selection of improper starting points for the algorithm or too few function evaluations. The median is more robust statistic than the mean if sufficiently many data points are available. The best and the worst value of the performance measure provide insights about the performance data, but they consider only one out of n data points, i.e., they are determined by one single data and therefore not very robust compared to the mean or median that are considering all data points. The quantiles are cut points which split a probability distribution into continuous intervals with equal probabilities. Similar to the median, they require a certain amount of data points and are probably meaningless for small data. Bartz-Beielstein [2006] presents a detailed discussion of these basic statistics.

6.2.3 Graphical Tools

Visualising final results. Graphical tools can provide more insight into the results and their distributions. The first set of graphical tools can be used to analyse the final results of the optimization runs. *Histograms* and *boxplots* are simple but effective tools and provide more information for further analysis of the results. Box plots visualize the distribution of the results. They illustrate the statistics introduced in Section 6.2.2 in a very compact and comprehensive manner and provide means to detect outliers. Histograms provide information about the shape of the distribution. Because the shape of histograms is highly affected by the size of the boxes, we strongly recommend combining histograms with density plots.

Visualising run-time behaviour. The second set of tools can be used to analyse the algorithm performance over time, i.e., information about the performance for every kth iteration is required. Suitable for the analysis of the performances of the optimization algorithms are *convergence plots* in which the performance of the algorithm can be evaluated against the number of function evaluations. This helps us to understand the dynamics of multiple algorithms in a single plot.

Histograms and box plots are also used in the graphical multiple problem analysis. Besides these common tools, specific tools for the multiple problem analysis were developed, e.g., *performance profiles* proposed in Dolan and Moré [2002]. They have emerged as an important tool to compare the performances of optimization algorithms based on the cumulative distribution function of a performance metric (CPU time, achieved optimum). It is the ratio of a performance metric obtained by each algorithm versus the best value the performance metric among all algorithms that is being compared. Such plots help to visualize the advantages (or disadvantages) of each competing algorithm graphically. Performance profiles are not applicable, if the (true or theoretical) optimum is unknown. However, there are solutions for this problem, e.g., using the best known solution so far or a guessed (most likely) optimum based on the user's experience.

As performance profiles are not evaluated against the number of function evaluations, they cannot be used to infer the percentage of the test problems that can be solved with some specific number of function evaluations. To attain this feature, the *data profiles* were designed for fixed-budget derivative free optimization algorithms [Moré and Wild, 2009]. It is appropriate to compare the best possible solutions obtained from various algorithms within a fixed budget.

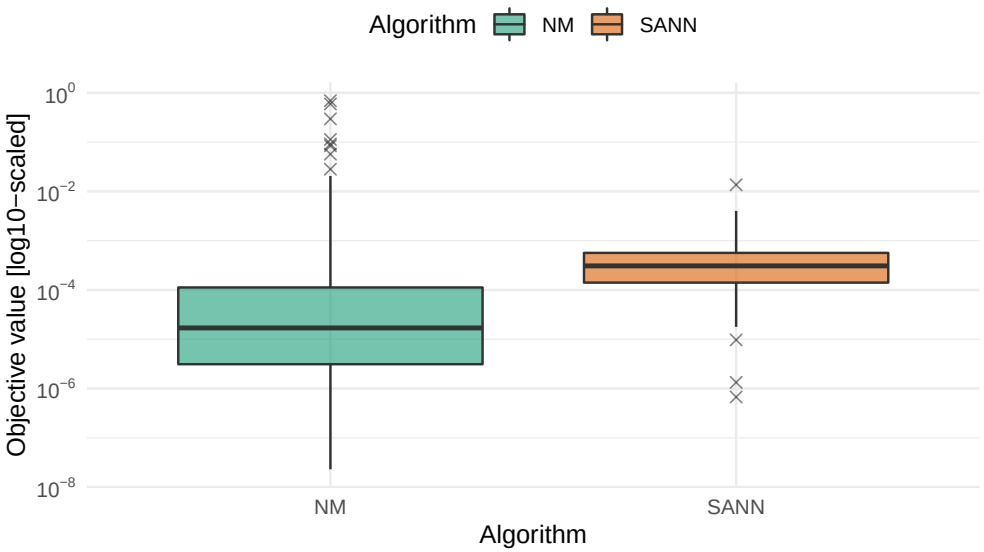

Figure 5: Boxplots of the achieved objective values (log-scaled) obtained by each 100 runs of NM and SANN on the Rosenbrock test function.

6.3 Confirmatory Analysis

6.3.1 Motivation

The second step in the three-level approach is referred to as *confirmatory analysis*, which is based in inferential statistics, because it implements a deductive approach: a given assumption (statistical hypothesis) is tested using the experimental data. Since the assumptions are formulated as statistical hypotheses, confirmatory analysis heavily relies on probability models. Its final goal is to provide definite answers to specific questions, i.e., questions for a specific experimental design. Because it uses probability models, its emphasis is on complex numerical calculations. Its main ingredients are hypothesis tests and confidence intervals. Confirmatory analysis usually generates more precise results for a specific context than EDA. But, if the context is not suitable, e.g., statistical assumptions are not fulfilled, a misleading impression of precision might might occur.

Often, EDA tools are not sufficient enough to clearly analyze the differences in the performances of algorithms, mainly when the differences are of smaller magnitude.

Example 6.1 (Comparing two algorithms)**.** *For example, let us consider the well known continuous Rosenbrock function [Rosenbrock, 1960]. We run the Nelder Mead (NM) algorithm [Nelder and Mead, 1965] and the SANN algorithm [Aarts and Korst, 1988] for each 100 runs. The numerical statistics of the results are shown in Table 2. Initial analysis of the summary statistics cannot reveal which algorithm is superior. Considering the mean of the performances SANN tends to outperform NM, but considering the median performances the vice versa can be observed. Also, from the boxplot in Figure 5 NM tends to outperform SANN. But it is not sufficient to support the conclusion that NM outperforms SANN in all respects. The presence of various outliers in NM can be seen in Figure 5. In such scenarios, a statistical analysis is recommended.*

The need to perform statistical analysis and various procedures involved in making decisions about selecting the best algorithm are widely discussed in [Amini and Barr, 1993, Barr et al., 1995, Carrano et al., 2011, Chiarandini et al., 2007, Eftimov et al., 2017, García et al., 2009, Golden et al., 1986, McGeoch, 1996]. The basic idea of statistical analysis is based on hypothesis testing. Before analysing the performance data, we should define two hypotheses i) the null hypothesis H_0 and ii) the alternative hypothesis H_1. The null hypothesis states that there is no statistical significant difference between the two algorithms'

Table 2: Overview of various key numerical metrics for the NM and SANN algorithms on the Rosenbrock test function.

Algorithm	Mean	Median	Standard Dev.	Best	Worst
NM	0.0200	1.7×10^{-5}	0.0960	2.3×10^{-8}	0.700
SANN	0.0006	3.0×10^{-4}	0.0015	6.7×10^{-7}	0.014

performances, while the alternative hypothesis directly contradicts the null hypothesis by indicating the statistical significance between the algorithms' performances. Hypothesis testing can be two-sided or one-sided. We will consider the one-sided case in the following, because it allows us to ask if algorithm instance a is better than algorithm instance b. Let $p(a)$ denote the performance of algorithm a. In the context of minimization, smaller performance values will be better, because we will compare the best solutions or the run times. The statement"a outperforms b" is equivalent to "$p(a) < p(b)$", which can be formulated as the statistical hypothesis $H_1 : p(b) - p(a) > 0$. It is a common agreement in hypotheses testing that this hypothesis H_1 will be tested against the null hypothesis $H_0 : p(b) - p(a) \leq 0$, which states that a is not better than b.

After the hypotheses are defined, we should select an appropriate statistical test, say T, for the analysis. The test statistic T which is a function of a random sample that allows researchers to determine the likelihood of obtaining the outcomes if the null hypothesis is true. The mean of the best found values from n repeated runs of an algorithm is a typical example of a test statistic. Additionally, a significance level α should be selected. Usually, a significance level of 95% is used. However, the selection of this value depends on the experimental design and the scientific question to be answered.

6.3.2 Assumptions for the Safe Use of the Parametric Tests

There are *parametric* and *non-parametric* statistical tests. To select between them, there are assumptions for the safe use of the parametric tests. Common assumptions include independence, normality, and homoscedasticity of variances. The independence assumption is directly met as the results of independent runs of the algorithm with randomly generated initial seeds are being compared. To check the normality assumption several tests can be performed including *Kolmogorov-Smirnov test* [Sheskin, 2003], *Shapiro-Wilk test* [Shapiro and Wilk, 1965], and *Anderson Darling test* [Anderson and Darling, 1952]. The normality assumption can be also checked by using graphical representation of the data using *histograms*, *empirical distribution functions* and *quantile-quantile* plots (Q-Q plots) [Devore, 2011]. The *Levene's test* [Levene, 1961] and *Bartlett's test* [Bartlett, 1937] can be performed to check if the assumption of equality of variances is violated. We should also mention that there are transformation approaches that may help to attain the normality, but this should be done with a great care, since we are changing the decision space. If the required assumptions are satisfied then we are selecting a parametric test since it has higher power than a non-parametric one, otherwise we should select a non-parametric one.

Additionally to the assumptions for the safe use of the parametric tests, before selecting an appropriate statistical test, we should take care if the performance data is paired or unpaired. Paired data is data in which natural or matched couplings occur. This means that each data value in one sample is uniquely paired to a data value in the other sample. The choice between paired and unpaired samples depends on experimental design, and researchers need to be aware of this when designing their experiment. Using Common Random Numbers (CRN) is a well-known technique for generating paired samples. If same seeds are used during the optimization, CRNs might reduce the variances and lead to more reliable statistical conclusions [Nazzal et al., 2012]. [Kleijnen, 1988].

6.3.3 A Pipeline for Selecting an Appropriate Statistical Test

A pipeline for selecting an appropriate statistical test for benchmarking optimization algorithms is presented in Figure 6. Further, we are going to explain some of them depending from the benchmarking scenario (i.e.,

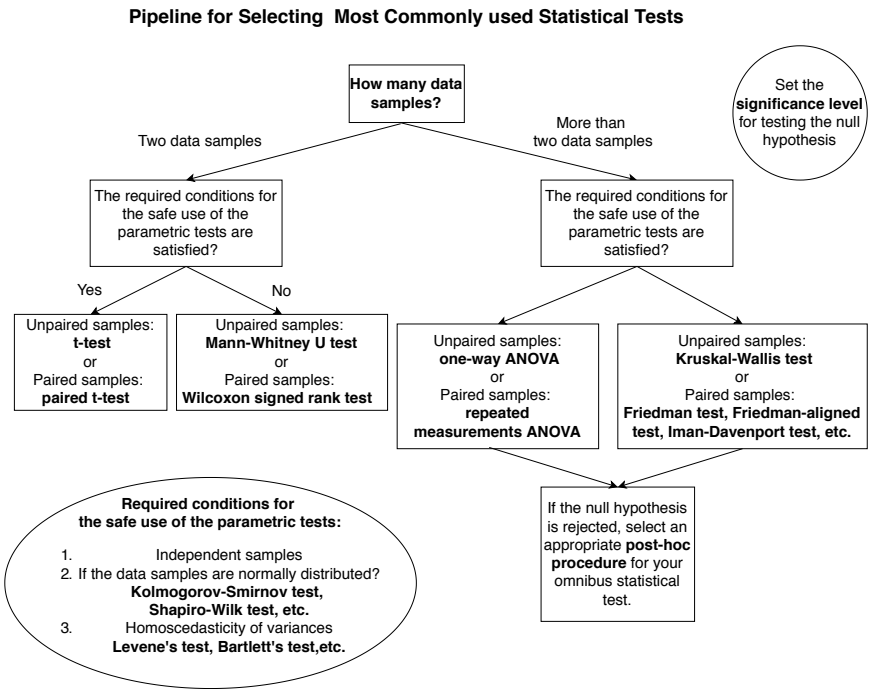

Figure 6: A pipeline for selecting an appropriate statistical test [Eftimov et al., 2020].

single-problem or multiple-problem analysis).

Single-problem analysis. As we previously mentioned, in this case, the performance measure data is obtained using multiple runs of k algorithm instances $a_1, \ldots, a_k$ on one selected problem instance π_j.

The comparison of samples in pairs is called a *pairwise comparison*. Note, that a pairwise comparison of algorithms does not necessary mean that the corresponding samples are paired. In fact, most pairwise comparisons use unpaired samples, because the setup for pairwise sampling is demanding, e.g., implementing random number streams etc. If more than two samples are compared at the same time, a *multiple comparison* is performed.

For pairwise comparison, the t test [Sheskin, 2003] is the appropriate parametric one, while its non-parametric version is the *Mann-Whitney U test* (i.e., *Wilcoxon-rank sum test*) [Hart, 2001]. In the case when more than two algorithms are involved, the parametric version is the one-way *ANOVA* [Lindman, 1974, Montgomery, 2017], while its appropriate non-parametric test is the *Kruskal-Wallis rank sum test* [Kruskal and Wallis, 1952]. Here, if the null hypothesis is rejected, then we should continue with a post-hoc procedure to define the pairs of algorithms that contribute to the statistical significance.

Multiple-problem analysis. The mean of the performance measure from multiple runs can be used as representative value of each algorithm on each problem. However, as stated above, averaging is sensitive to outliers, which needs to be considered especially because optimization algorithms could have poor runs. For this reason, the median of the performance measure from the multiple runs can also be used as more robust statistic.

Both mean and median are sensitive to errors inside some ϵ-neighborhood (i.e., small difference between their values that is not recognized by the ranking schemes of the non-parametric tests), which can additionally affect the statistical result. For these reasons, Deep Statistical Comparison (DSC) for comparing evolutionary algorithms was proposed [Eftimov et al., 2017]. Its main contribution is its ranking scheme, which is based on the whole distribution, instead of using only one statistic to describe the distribution, such as mean or

median.

The impact of the selection of the three above-presented transformations, which can be used to find a representative value for each algorithm on each problem, to the final result of the statistical analysis in the multiple-problem analysis is presented in [Eftimov and Korošec, 2018].

Statistical tests. No matter which transformation is used, once the data for analysis is available, the next step is to select an appropriate statistical test. For pairwise comparison, the *t test* is the appropriate parametric one [Sheskin, 2003], while its relevant non-parametric version is the *Wilcoxon signed rank test* [Wilcoxon, 1945]. In the case when more than two algorithms are involved, the parametric version is the repeated measurements *ANOVA* [Lindman, 1974, Montgomery, 2017], while its appropriate non-parametric tests are the *Friedman rank-based test* [Friedman, 1937], *Friedman-aligned test* [García et al., 2009], and *Iman-Davenport test* [García et al., 2009]. Additionally, if the null hypothesis is rejected, same as the single-problem analysis, we should continue with a post-hoc procedure to define the pairs of algorithms that contribute to the statistical significance.

Other non-parametric tests are the non-parametric rank based tests, which are suitable when the distribution assumptions are questionable [Sheskin, 2003]. Using them, the data is ranked and then the p-value is calculated for the ranks and not the actual data. This ranking helps to eliminate the problem of skewness and in handling extreme values. The *permutation test* [Pesarin, 2001] estimates the permutation distribution by shuffling the data without replacement and identifying almost all possible values of the test statistic. The *Page's trend test* [Derrac et al., 2014] is also a non-parametric test that can be used to analyse convergence performance of evolutionary algorithms.

Post-hoc procedures. When we have more than two algorithms that are involved in the comparison, the appropriate statistical test can find statistical significance between the algorithms' performances, but is is not able to define the pairs of algorithms that contribute to this statistical significance. For this reason, if the null hypothesis is rejected, we should continue with a *post-hoc* test.

The *post-hoc* testing can be done in two scenarios: i) all pairwise comparisons and ii) multiple comparisons with a control algorithm. Let us assume that we have k algorithms involved in the comparison, so in the first scenario we should perform $k(k-1)/2$ comparisons, and in the second one $k-1$.

In the case of all pairwise comparisons, the post-hoc test statistic should be calculated. It depends on the appropriate statistical test that is used to compare all algorithms together, which rejected the null hypothesis. After that, the obtained p-values are corrected with some post-hoc procedure. For example, if the null hypothesis in the *Friedman test*, *Friedman aligned-ranks test*, or *Iman–Davenport test*, is rejected, we can use the *Nemenyi*, *Holm*, *Shaffer*, and *Bergmann* correction to adapt the p-values.

The multiple comparisons with a control algorithm is the scenario when our newly developed is the control algorithm and we are comparing it with state-of-the-art algorithms. Same as the previous scenario, the post-hoc statistic depends on the appropriate statistical test that is used to compare all algorithms together, which rejected the null hypothesis, and the obtained p-values are corrected with some post-hoc procedure. In the case of *Friedman test*, *Friedman aligned-ranks test* , or *Iman–Davenport test*, appropriate post-hoc procedures are: *Bonferroni*, *Holm*, *Hochberg*, *Hommel*, *Holland*, *Rom*, *Finner*, and *Li*.

Another way to the multiple comparisons with a control algorithms is to perform all comparisons between the control algorithm and each other algorithm using some pairwise test. In this case, we should be careful when making a conclusion, since we are losing the control on the Family-Wise Error Rate(FWER) when performing multiple pairwise comparisons. All obtained p-values will come from independent pairwise comparisons. The calculation of the true statistical significance for combining pairwise comparisons is presented in [Eftimov et al., 2017, García et al., 2009].

More information about different post-hoc procedures and their application in benchmarking theory in evolutionary computation is presented in [García et al., 2009].

6.4 Relevance Analysis

6.4.1 Motivation

The third step of the recommended approach is related to the practical relevance of our statistical findings: are the differences really meaningful in practice or are they only statistical "artifacts" caused by an inadequate experimental design? A typical example for these artifacts is a difference in performance, say δ, which is statistically significant but of no practical relevance, because a value as small as δ cannot be measured in real-world scenarios. So, there is still a gap when transferring the learned knowledge from theory to practice. This happens because the statistical significance that exists is not scientifically meaningful in a practical sense.

Example 6.2 (Assembly line)**.** *Let us assume that two optimization algorithms that should minimize the average time of a production process, e.g., an assembly line, are compared. The mean difference in performance is* $\delta = 10^{-14}$*, which is statistically significant. However, this difference has no meaning in reality, because it is far below the precision of the assembly line timer.*

For this reason, when we are performing a statistical analysis, we should also try to find the relevance of the statistical significance to real world application. We should also mention that the practical significance depends from the specific problem being solved. Additionally, this is also true in benchmarking performed for scientific publications, where the comparisons of the performance measures can be affected by several factors such as computer accuracy (i.e., floating points), variable types (4-byte float, 8-byte float, 10-byte float), or even the stopping criteria that is the error threshold when the algorithms are stopped. All these factors can result in different values, which does not represent the actual performance of the algorithms even if statistical significance is found.

6.4.2 Severity: Relevance of Parametric Test Results

In order to probe the meaningfulness of the statistically significant result, it is suggested to perform a post-data analysis. One such post data analysis is the severity measure, a meta statistical principle [Bartz-Beielstein et al., 2010, Mayo and Spanos, 2006]. Severity describes the degree of support to decisions made using classical hypothesis testing. Severity takes into account the data and performs a post-data evaluation to scrutinize the decisions made by analyzing how well the data fits the testing framework. The severity is the actual power attained in the post data analysis and can be described separately for the decision of either rejecting or not rejecting the null hypothesis.

The conclusions obtained from the hypothesis testing is dependent on sample size and can suffer from the *problem of large n*. Severity deals with this problem directly by changing the intensity of the severity test. For an increased number of runs, the severity test is less severe. And for a smaller number of runs, the severity test is more severe [Mayo and Spanos, 2006].

6.4.3 Multiple-Problem Analysis

We present two approaches that investigate the scientific meaningfulness of statistically significant results in the multiple-problem setting. One approach is the Chess Rating Systems for Evolutionary Algorithms (CRS4EA), which is an empirical algorithm for comparing and ranking evolutionary algorithms [Veček et al., 2014]. It makes a chess tournament where the optimization algorithms are considered as chess players and a comparison between the performance measures of two optimization algorithms as the outcome of a single game. A draw limit that defines when two performance measure values are equal should be specified by the user and it is a problem specific. At the end, each algorithm has its own rating which is a result from the tournament and the statistical analysis is performed using confidence intervals calculated using the algorithms rating.

The second approach is the practical Deep Statistical Comparison (pDSC), which is a modification of the DSC approach used for testing for statistical significance [Eftimov and Korošec, 2019]. The basic idea is that the data on each problem should be pre-processed with some practical level specified by a user, and after

that involved with DSC to find relevant difference. Two pre-processing steps are proposed: i) sequential pre-processing, which pre-processes the performance measures from multiple runs in a sequential order, and ii) a Monte-Carlo approach, which pre-processes the performance measure values by using a Monte-Carlo approach to avoid the dependence of the practical significance on the order of the independent runs. A comparison between the CRS4EA and pDSC is presented in [Eftimov and Korošec, 2019]. Using these two approaches the analysis is made for a multiple-problem scenario. Additionally, the rankings from pDSC obtained on a single-problem level can be used for single-problem analysis.

6.5 Open Issues

An important aspect not addressed in this iteration of the document is the analysis of the benchmark problems themselves rather than the performance of the algorithms operating thereon. That is, which means exist for investigating structural characteristics of the benchmarking problem at hand? How can one (automatically) extract its most relevant information? How should those information be interpreted? There exists a variety of approaches for this, and each of them helps to improve the understanding of the respective problem, and in consequence may facilitate the design, selection and/or configuration of a suitable algorithm.

Linked to the above is a discussion of methods for visualizing problem landscapes. Visualizing the landscape of a continuous problem, or plotting approximate tours for a given TSP instance, usually improves our understanding of its inherent challenges and reveals landscape characteristics such as multimodality. Moreover, such visualizations also help to study the search behavior of the algorithms under investigation. Unfortunately, the vast majority of works treats the issue of visualizing problems very poorly, so we will make sure to address this particular issue in the continuation of this document.

7 Experimental Design

7.1 Design of Experiments (DoE)

Unfortunately, many empirical evaluations of optimization algorithms are performed and reported without addressing basic experimental design considerations [Brownlee, 2007]. An important step to make this procedure more transparent and more objective is to use DOE and related techniques. They provide an algorithmic procedure to make comparisons in benchmarking more transparent. Experimental design provides an excellent way of deciding which and how many algorithm runs should be performed so that the desired information can be obtained with the least number of runs.

DOE is planning, conducting, analyzing, and interpreting controlled tests to evaluate the influence of the varied factors on the outcome of the experiments. The importance and the benefits of a well designed planned experiment have been summarized by Hooker [1995]. Johnson [2002b] suggests to report not only the run time of an algorithm, but also explain the corresponding adjustment process (preparation or tuning before the algorithm is run) in detail, and therefore to include the time for the adjustment in all reported running times to avoid a serious underestimate.

The various key implications involved in the DOE are clearly explained in Kleijnen [2001]. A comprehensive list of the recent publications on design techniques can be found in Kleijnen [2017]. The various design strategies in the Design and Analysis of Computer Experiments (DACE) are discussed by Santner et al. [2003]. Wagner [2010] discusses important experimental design topics, e.g., "How many replications of each design should be performed?" or "How many different algorithm runs should be evaluated?"

This section discusses various important practical aspects of formulating the design of experiments for a stochastic optimization problem. The key principles are outlined. For a detailed reading of the DOE, the readers are referred to Montgomery [2017] and Kleijnen [2015].

7.2 Design Decisions

Design decisions can be based on geometric or on statistical criteria [Pukelsheim, 1993, Santner et al., 2003]. Regarding geometric criteria, two different design techniques can be distinguished: The samples can be

placed either (1) on the boundaries, or (2) in the interior of the design space. The former technique is used in DOE whereas DACE uses the latter approach. An experiment is called *sequential* if the experimental conduct at any stage depends on the results obtained so far. Sequential approaches exist for both variants.

We recommend using factorial designs or space-filling designs instead of the commonly used One-factor-at-a-time (OFAT) designs. When several factors are involved in an experiment, the OFAT design strategy is inefficient as it suffers from various limitations including huge number of experimental runs and inability to identify the interactions among the factors involved. It is highly recommended to use a multi-factorial design [Montgomery, 2017]. The Factorial designs are robust and faster when compared with OFAT. For a complete insight into Fully-Factorial and Fractional Factorial designs the readers are redirected to Montgomery [2017]. The Taguchi design [Roy, 2001] is a variation of the fractional factorial design strategy, which provides robust designs at better costs with fewer evaluations. The Plackett and Burman design [Plackett and Burman, 1946] are recommended for screening. The modern space-filling designs are sometimes more efficient and require fewer evaluations than the fractional designs, especially in case of non-linearity. Further information about space-filling designs can be found in Santner et al. [2003].

However, it is still an open question which design characteristics are important: "... extensive empirical studies would be useful for better understanding what sorts of designs perform well and for which models" [Santner et al., 2003, p. 161].

7.3 Designs for Benchmark Studies

In the context of DOE and DACE, runs of an optimization algorithm instance will be treated as experiments. There are many degrees of freedom when an optimization algorithm instance is run. In many cases optimization algorithms require the determination of parameters (e.g., the population size in ESs) before the optimization run is performed. From the viewpoint of an experimenter, design variables (factors) are the parameters that can be changed during an experiment. Generally, there are two different types of factors that influence the behavior of an optimization algorithm:

1. problem-specific factors, i.e., the objective function,
2. algorithm-specific factors, i.e., the population size of an ES and other parameters which need to be set to derive an executable algorithm instance.

We will consider experimental designs that comprise problem-specific factors and algorithm-specific factors. Algorithm-specific factors will be considered first. *Implicit parameters* can be distinguished from *explicit parameters* (synonymously referred to as *endogeneous* and *exogeneous* in [Beyer and Schwefel, 2002]). The latter are explicitly exposed to the user, whereas the former are often hidden, i.e., either made inaccessible to the user (e.g., when the algorithm code is not made available) or simply "hidden" in the implementation and not easily identifiable as a parameter that can be optimized.

An *algorithm design* is a set of parameters, each representing one specific setting of the design variables of an algorithm and defining an algorithm instance. A design can be specified by defining ranges of values for the design parameters. Note that a design can contain none, one, several, or even infinitely many design points, each point representing an algorithm instance. Consider the set of explicit strategy parameters for PSO algorithms with the following values: swarm size $s = 10$, cognitive parameter $c_1 \in [1.5, 2]$, social parameter $c_2 = 2$, starting value of the inertia weight $w_{\max} = 0.9$, final value of the inertia weight $w_{\text{scale}} = 0$, percentage of iterations for which $w_{\max}$ is reduced $w_{\text{iterscale}} = 1$, and maximum value of the step size $v_{\max} = 100$. This algorithm design contains infinitely many design points, because c_1 is not fixed.

Problem designs provide information related to the optimization problem, such as the available resources (number of function evaluations) or the problem's dimension.

An *experimental design* consists of a problem design and an algorithm design. Benchmark studies require complex experimental designs, because they are combinations of several problem and algorithm designs. Furthermore, as discussed in Section 5, one or several performance measures must be specified.

7.4 How to Select a Design for Benchmarking

The following points have to be considered when designing an benchmark study[23]:

- What are the main goals of the experiment? (see Section 2)
- What is/are the test problem(s) and which (type of) instances do we select? (see Section 3)
- How many algorithms are to be tested? (see Section 4.3)
- How many test problems/test classes are relevant for the study? (see Section 3)
- How tuning of algorithms has to be performed? (see Section 4.2)
- What validation procedures are considered to evaluate the results of the experiment? (see Section 5)
- How will the results be analyzed? (see Section 6)
- How will the results be presented? (see Section 8)
- How are randomization and replicability of the experiment achieved? (see Section 9)

7.5 Tuning Before Benchmarking

Brownlee [2007] discusses the importance of tuning an algorithm before benchmarking. Bartz-Beielstein and Preuss [2010] state that comparisons of tuned versus untuned algorithms are not fair and should be avoided. During a benchmark study the employed parameter settings are extremely important as they largely define the obtained performance. Depending on the availability of code for the algorithms under scope and time for parameter searches, there are different possibilities to make a fair comparison:

- In the best case, the code for all methods is available. It is then possible to perform a parameter search for each problem and each algorithm via a tuning method. Taking the best parameter sets for each method for each problem ensures comparing the algorithms at their peak performance.
- If algorithm runs on the chosen problems take too long for a full tuning process, one may however perform a simple space-filling design on the parameter space, e.g., a LHD or a low-discrepancy point set [Matoušek, 2009] with only a few design points and repeats. This prevents misconfigurations of algorithms as one probably easily gets into the "ball park" [De Jong, 2007] of relatively good parameter settings. Most likely, neither algorithm works at its peak performance level, but the comparison is still fair.
- If no code other than for one's own algorithm is available, one has to resort to comparing with default parameter values. For a new algorithm, these could be determined by a common tuning process over the whole problem set. Note however, that such comparison deliberately abstains from setting good parameters for specific problems, even if this would be attempted for any real-world application.

7.6 Open Issues

(O7.1) *Best Designs.*
Same authors consider LHDs as the default choice, even if for numerous applications a superiority of other space-filling or low-discrepancy designs has been demonstrated Santner et al. [2003]. The question when to prefer i.i.d. uniform sampling, LHDs, low-discrepancy point sets, other space-filling designs, or sets minimizing some other diversity criterion is largely open.

[23] At the moment, this is only a list, which will be extended in forthcoming versions of this survey.

(O7.2) *Multiple Objectives.*
Sometimes properties of the objective function are used to determine the quality of a design. Therefore, it remains unclear how to measure the quality in settings where the objective function is unknown. Furthermore, problems occur if wrong assumptions about the objective function, e.g., linearity, are made. And, last but not least, in MOO, where no single objective can be specified, finding the optimal design can be very difficult [Santner et al., 2003].

8 How to Present Results?

8.1 General Recommendations

Several papers have been published in the last years, which give recommendations on how to report results. As Gent and Walsh [1994] already stated in 1994, after having generated some good results in your benchmark study, there are still many mistakes to make. They give the following recommendations:

1. present statistics
2. report negative results
3. do not push deadlines

Barr et al. [1995] in their classical work on reporting empirical results of heuristics specify a loose experimental setup methodology with the following steps:

1. define the goals of the experiment,
2. select measure of performance and factors to explore,
3. design and execute the experiment,
4. analyze the data and draw conclusions, and finally
5. report the experimental results.

They then suggest eight guidelines for reporting results, in summary they are; reproducibility, specify all influential factors (code, computing environment, etc.), be precise regarding measures, specify parameters, use statistical experimental design, compare with other methods, reduce variability of results, ensure results are comprehensive. They then go on to clarify these points with examples.

8.2 Reporting Methodologies

Besides recommendations, that provide valuable hints on how to report results, there exist also methodologies, which employ a scientific methodology, e.g., based on hypothesis testing [Popper, 1959, 1975]. Such a methodology was proposed by Bartz-Beielstein and Preuss [2010]. They propose organizing the presentation of experiments into seven parts, as follows:

(R.1) *Research question*
Briefly names the matter dealt with, the (possibly very general) objective, preferably in one sentence. This is used as the report's "headline" and related to the primary model.

(R.2) *Pre-experimental planning*
Summarizes the first—possibly explorative—program runs, leading to task and setup (R-3 and R-4). Decisions on employed benchmark problems or performance measures should be taken according to the data collected in preliminary runs. The report on pre-experimental planning should also include negative results, e.g., modifications to an algorithm that did not work or a test problem that turned out to be too hard, if they provide new insight.

(R.3) *Task*
Concretizes the question in focus and states scientific claims and derived statistical hypotheses to test. Note that one scientific claim may require several, sometimes hundreds, of statistical hypotheses. In case of a purely explorative study, as with the first test of a new algorithm, statistical tests may not be applicable. Still, the task should be formulated as precisely as possible. This step is related to the experimental model.

(R.4) *Setup*
Specifies problem design and algorithm design, including the investigated algorithm, the controllable and the fixed parameters, and the chosen performance measuring. The information provided in this part should be sufficient to replicate an experiment.

(R.5) *Results/Visualization*
Gives raw or produced (filtered) data on the experimental outcome and additionally provides basic visualizations where meaningful. This is related to the data model.

(R.6) *Observations*
Describes exceptions from the expected, or unusual patterns noticed, without subjective assessment or explanation. As an example, it may be worthwhile to look at parameter interactions. Additional visualizations may help to clarify what happens.

(R.7) *Discussion*
Decides about the hypotheses specified in R-3, and provides necessarily subjective interpretations of the recorded observations. Also places the results in a wider context. The leading question here is: What did we learn?

This methodology was extended and refined in Preuss [2015]. It is important to divide parts R-6 and R-7, to facilitate different conclusions drawn by others, based on the same results/observations. This distinction into parts of increasing subjectiveness is similar to the suggestions of Barr et al. [1995], who distinguish between results, their analysis, and the conclusions drawn by the experimenter.

Note that all of these parts are already included in current good experimental reports. However, they are usually not separated but wildly mixed. Thus we only suggest to insert labels into the text to make the structure more obvious.

We also recommend to keep a journal of experiments with single reports according to the above scheme to enable referring to previous experiments later on. This is useful even if single experiments do not find their way into a publication, as it improves the overview of subsequent experiments and helps to avoid repeated tests.

8.3 Open Issues

The former addressed presentation of negative results (cf. Gent and Walsh [1994] and above in 2) is not adequately accepted in the research community. While a paper improving existing experimental results or outperforming a different algorithm regularly gets accepted for publication, papers presenting negative results regularly will not.

9 How to Guarantee Reproducibility?

Reproducibility has been a topic of interest in the experimental analysis of algorithms for many decades. Classical works [Johnson, 2002a] advise *ensuring reproducibility*, but caution that the classical understanding of reproducibility in computer science, i.e., running exactly the same code on the same machine returns exactly the same measurements, differs substantially from the understanding in other experimental sciences, i.e., a different implementation of the experiment under similar conditions returns measurements that lead to the same conclusions.

For example, the "Reproducibility guidelines for AI research"[24] intended to be adopted by the Association for the Advancement of Artificial Intelligence (AAAI) are clearly focused on the concept of reproducibility in computer science.

Trying to clearly define various reproducibility concepts, the ACM distinguishes among:[25]

Repeatability (Same team, same experimental setup) The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.

Replicability (Different team, same experimental setup) The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.

Reproducibility (Different team, different experimental setup) The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

The above classification helps to identify various levels of reproducibility, reserving the term "Reproducibility" to the most scientifically useful, yet hardest to achieve. There are many practical guidelines and software systems available to achieve Repeatibility and Replicability [Gent et al., 1997, Johnson, 2002a], including code versioning tools (Subversion and Git), data repositories (Zenodo), reproducible documents (Rmarkdown and Jupyter notebooks), and reproducible software environments (OSF[26], CodeOcean and Docker). However, it is not so clear how to successfully achieve Reproducibility. For achieving reproducibility, one must give up on exactly reproducing the results and provide statistical guidelines that are commonly accepted by the field to provide sufficient evidence for a conclusion, even under different, but similar, experimental conditions. What constitutes *similar* experimental conditions is dependent on the experiment and there is no simple answer when benchmarking algorithms.

10 Summary and Outlook

10.1 Summary

This survey compiles ideas and recommendations from more than a dozen researchers with different backgrounds and from different institutions around the world. Promoting best practice in benchmarking is its main goal. This first version is a result from long and fruitful discussions among the authors. The authors agreed on eight essential topics, that should be considered in every benchmark study (goals, problems, algorithms, performance, analysis, design, presentation, and reproducibility). These topics defined the section structure of this article.

However, this is only the first version. It is hopefully a good starting point, but definitely not a textbook that explains every single approaches in detail. It is basically a guide (similar to the famous hitch-hiker's guide to EC [Heitkötter and Beasley, 1994]) and has a long list of references, which covers classical papers as well as the most recent ones.

Every section presents recommendations, best practice examples, and open issues.

Although we tried to include the most relevant contributions, we are aware that important contributions are missing. Because the acceptance of the proposed recommendations is crucial, we would like to invite

[24] http://folk.idi.ntnu.no/odderik/reproducibility_guidelines.pdf

[25] Quoting from:
https://www.acm.org/publications/policies/artifact-review-badging

[26] https://osf.io/

more researchers to share their knowledge with us. Moreover, as the field of benchmarking is constantly changing, this article will be regularly updated and published on arXiv. Interested readers can use the associated e-mail address for this project: benchmarkingbestpractice@gmail.com.

There are several other initiatives that are trying to improve benchmarking standards in query-based optimization fields, e.g., the Benchmarking Network[27], an initiative built to consolidate and to stimulate activities on benchmarking iterative optimization heuristics.

10.2 Outlook

As mentioned above, this survey is only the beginning of a wonderful journey. It can serve as a starting point for many activities that improve the quality of benchmark studies and enhance the quality of research in EC and related fields. Next steps can be as follows:

1. offer tutorials and organize workshops,
2. compile videos, which explain how to set up the experiments, analyze results, and report important findings,
3. provide software tools,
4. develop a comprehensible check-list, especially for beginners in benchmarking,
5. include a discussion section in every section, which describes controversial topics and ideas.

The final goal is to provide well-accepted guidelines (rules) that might be useful for authors and reviewers. Consider the following (rudimentary and incomplete) checklist, that can serve as a guideline for authors and reviewers:

1. goals: did the authors clearly state the reasons for this study?
2. problems: is the selection of problem instances well motivated and justified?
3. algorithms: do comparisons include relevant competitors?
4. performance: is the choice of the performance measure adequate?
5. analysis: are standards from statistics considered?
6. design: does the experimental setup enable efficient and fair experimentation? What measures are taken to avoid "cherry-picking results"?
7. presentation: are the results well organized and explained?
8. reproducibility: data and code availability?

Transparent, well accepted standards will improve the review process in EC and related fields significantly. These common standards might also accelerate the review process, because it improves the quality of submissions and helps reviewers to write objective evaluations. Most importantly, it is not our intention to dictate specific test statistics, experimental designs, or performance measures. Instead, we claim that publications in EC would improve, if authors explain, *why* they have chosen this specific measure, tool, or design. And, last but not least, authors should describe the goal of their study.

In our opinion, starting this discussion is very important. Maybe, this survey poses more questions than answers, which is fine. Therefore, we conclude this article with a famous saying that is attributed to Robert Feynman[28]:

> I would rather have questions that can't be answered than answers that can't be questioned.

[27] https://sites.google.com/view/benchmarking-network/

[28] https://en.wikiquote.org/w/index.php?title=Talk:Richard_Feynman&oldid=2681873#%22I_would_rather_have_questions_that_cannot_be_answered%22

Acknowledgments

This work has been initiated at Dagstuhl seminar 19431 on Theory of Randomized Optimization Heuristics,[29] and we gratefully acknowledge the support of the Dagstuhl seminar center to our community.

We thank C. M. Fonseca for his important input and our fruitful discussion, which helped us shape the section on performance measures.

C. Doerr acknowledges support from the Paris Ile-de-France region and from a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

T. Weise acknowledges support from the National Natural Science Foundation of China under Grant 61673359 and the Hefei Specially Recruited Foreign Expert program.

T. Eftimov acknowledges support from the Slovenian Research Agency under research core funding No. P2-0098 and project No. Z2-1867.

A. Fischbach and T. Bartz-Beielstein acknowledge support from the German Federal Ministry of Education and Research in the funding program Forschung an Fachhochschulen under the grant number 13FH007IB6 (KOARCH).

S. Chandrasekaran and T. Bartz-Beielstein acknowledge support from the Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen in the funding program FH Zeit für Forschung under the grant number 005-1703-0011 (OWOS).

B. Naujoks and T. Bartz-Beielstein acknowledge support from the European Commission's H2020 programme, H2020-MSCA-ITN-2016 UTOPIAE (grant agreement No. 722734), as well as the DAAD (German Academic Exchange Service), Project-ID: 57515062 "Multi-objective Optimization for Artificial Intelligence Systems in Industry".

J. Bossek acknowledges support by the Australian Research Council (ARC) through grant DP190103894.

J. Bossek and P. Kerschke acknowledge support by the European Research Center for Information Systems (ERCIS).

We also acknowledge support from COST action 15140 on Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO).

[29] https://www.dagstuhl.de/19431

Glossary

AAAI Association for the Advancement of Artificial Intelligence. 38

ACM Association for Computing Machinery. 15, 38

ASlib Algorithm Selection Library. 17

BBOB Black-Box-Optimization-Benchmarking. 15

BFGS Broyden-Fletcher-Goldfarb-Shanno. 19

CEC Congress on Evolutionary Computation. 15–17

CMA-ES Covariance Matrix Adaptation Evolution Strategy. 6, 20

COCO Comparing Continuous Optimizers. 4, 12

CRN Common Random Numbers. 29

CRS4EA Chess Rating Systems for Evolutionary Algorithms. 32, 33

DACE Design and Analysis of Computer Experiments. 33, 34

DOE Design of Experiments. 5, 10, 18, 33, 34

EA Evolutionary Algorithm. 15, 16, 20

EC Evolutionary Computation. 4, 5, 12, 22, 38, 39

EDA Exploratory Data Analysis. 26–28

EDAlgo Estimation of Distribution Algorithm. 20

EGO Efficient Global Optimization. 20

ELA Exploratory Landscape Analysis. 14, 26

ERT Expected Running Time. 24

ES Evolution Strategy. 22, 34

FE Function Evaluation. 22

GECCO Genetic and Evolutionary Computation Conference. 15, 16

GPD Generalized Position-Distance. 15

IEEE Institute of Electrical and Electronics Engineers. 15

irace iterated racing. 20

LABS Low Auto-correlation Binary Sequence. 13

LHD Latin Hypercube Design. 9, 18, 35

MACODA Many Criteria Optimization and Decision Analysis. 17

MAX-SAT Maximum Satisfiability. 22

MOO Multi-Objective Optimization. 22, 24, 25, 36

NFLT no free lunch theorem. 4

NM Nelder Mead. 28, 29

OFAT One-factor-at-a-time. 34

ParamILS Iterated Local Search in Parameter Configuration Space. 20

PBO Pseudo-Boolean Optimization. 15

pDSC practical Deep Statistical Comparison. 32, 33

PSO Particle Swarm Optimization. 20, 34

SANN Simulated Annealing. 19, 28, 29

SAT Boolean Satisfiability. 22

SMAC Sequential Model-based Algorithm Configuration. 20

SPOT Sequential Parameter Optimization Toolbox. 10, 20

TSP Traveling Salesperson Problem. 14, 22, 23, 25, 33

References

Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines.* New York, NY; John Wiley and Sons Inc., 1988.

Stavros P Adam, Stamatios-Aggelos N Alexandropoulos, Panos M Pardalos, and Michael N Vrahatis. No Free Lunch Theorem: A Review. In *Approximation and Optimization*, pages 57 – 82. Springer, 2019.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

Mohammad M Amini and Richard S Barr. Network reoptimization algorithms: A statistically designed comparison. *ORSA Journal on Computing*, 5(4):395–409, 1993.

Theodore W Anderson and Donald A Darling. Asymptotic theory of certain" goodness of fit" criteria based on stochastic processes. *The annals of mathematical statistics*, pages 193–212, 1952.

Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. Model-based genetic algorithms for algorithm configuration. In *Proc. of International Conf. on Artificial Intelligence (IJCAI'15)*, pages 733–739. AAAI, 2015.

Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics.* World Scientific, 2011.

Anne Auger and Nikolaus Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1777–1784. IEEE, 2005.

Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation.* IOP Publishing Ltd., GBR, 1st edition, 1997.

Richard S Barr, Bruce L Golden, James P Kelly, Mauricio GC Resende, and William R Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

Maurice Stevenson Bartlett. Properties of sufficiency and statistical tests. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 160(901):268–282, 1937.

Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism.* Natural Computing Series. Springer, 2006.

Thomas Bartz-Beielstein and Mike Preuss. The Future of Experimental Research. In Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 17–46. Springer, Berlin, Heidelberg, New York, 2010.

Thomas Bartz-Beielstein and Martin Zaefferer. Model-based methods for continuous and discrete global optimization. *Applied Soft Computing*, 55:154 – 167, 2017.

Thomas Bartz-Beielstein, Konstantinos E Parsopoulos, and Michael N Vrahatis. Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis & Computational Mathematics*, 1(2):413–433, 2004.

Thomas Bartz-Beielstein, Christian WG Lasarczyk, and Mike Preuss. Sequential Parameter Optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 773 – 780. IEEE, 2005.

Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss. *Experimental methods for the analysis of optimization algorithms.* Springer, 2010.

Thomas Bartz-Beielstein, Lorenzo Gentile, and Martin Zaefferer. In a nutshell: Sequential parameter optimization. Technical report, TH Köln, 2017.

Vahid Beiranvand, Warren Hare, and Yves Lucet. Best Practices for Comparing Optimization Algorithms. *Optimization and Engineering*, 18(4):815 – 848, 2017.

James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research (JMLR)*, 13:281 – 305, 2012.

James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer, 2013.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies—A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle. An Empirical Assessment of the Properties of Inverted Generational Distance on Multi- and Many-Objective Optimization. In *Proc. of the 9th International Conference on Evolutionary Multi-Criterion Optimization*, volume 10173 of *LNCS*, pages 31–45. Springer, 2017.

Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Thomas Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence (AIJ)*, 237:41 – 58, 2016.

Mark S. Boddy and Thomas L. Dean. Solving Time-Dependent Planning Problems. Technical Report CS-89-03, Brown University, Department of Computer Science, Providence, RI, USA, February 1989. URL ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-03.pdf.

Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. The Travelling Thief Problem: The First Step in the Transition from Theoretical Problems to Realistic Problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 1037 – 1044. IEEE, 2013.

Mohammad Reza Bonyadi, Zbigniew Michalewicz, Markus Wagner, and Frank Neumann. Evolutionary Computation for Multicomponent Problems: Opportunities and Future Directions. In *Optimization in Industry: Present Practices and Future Scopes*, pages 13 – 30. Springer, 2019. doi:10.1007/978-3-030-01641-8_2.

Peter A. N. Bosman and Dirk Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation (TEVC)*, 7(2):174 – 188, May 2003.

Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. Evolving Diverse TSP Instances by Means of Novel and Creative Mutation Operators. In *Proc. of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 58 – 71. ACM, 2019.

Jakob Bossek, Pascal Kerschke, Aneta Neumann, Frank Neumann, and Carola Doerr. Evolving Sampling Strategies for One-Shot Optimization Tasks. In *Proc. of Parallel Problem Solving from Nature*. Springer, 2020a. Full version available online at https://arxiv.org/abs/1912.08956.

Jakob Bossek, Pascal Kerschke, and Heike Trautmann. A Multi-Objective Perspective on Performance Assessment and Automated Selection of Single-Objective Optimization Algorithms. *Applied Soft Computing Journal (ASOC)*, 88:105901, March 2020b.

Jakob Bossek, Pascal Kerschke, and Heike Trautmann. Anytime Behavior of Inexact TSP Solvers and Perspectives for Automated Algorithm Selection. In *Proc. of the IEEE Congress on Evolutionary Computation*. IEEE, 2020c. A preprint of this manuscript can be found at https://arxiv.org/abs/2005.13289.

Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical Hyper-Parameters: No Random, No Cry. *CoRR*, abs/1706.03200, 2017. URL http://arxiv.org/abs/1706.03200.

Jürgen Branke. *Evolutionary optimization in dynamic environments*. Kluwer Academic Publishers, 2001.

Jürgen Branke, Christian Schmidt, and Hartmut Schmeck. Efficient Fitness Estimation in Noisy Environments. In *Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 243 – 250. Morgan Kaufmann, 2001.

Dimo Brockhoff, Tea Tušar, Anne Auger, and Nikolaus Hansen. Using well-understood single-objective functions in multiobjective black-box optimization test suites, 2016.

Jason Brownlee. A Note on Research Methodology and Benchmarking Optimization Algorithms. Technical report, Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID 70125, 2007.

Eduardo G Carrano, Elizabeth F Wanner, and Ricardo HC Takahashi. A multicriteria statistical based comparison methodology for evaluating evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 15(6): 848–870, 2011.

Marie-Liesse Cauwet, Camille Couprie, Julien Dehos, Pauline Luc, Jérémy Rapin, Morgane Rivière, Fabien Teytaud, and Olivier Teytaud. Fully parallel hyperparameter search: Reshaped space-filling. *CoRR*, abs/1910.08406, 2019. URL http://arxiv.org/abs/1910.08406.

Shelvin Chand and Markus Wagner. Evolutionary Many-Objective Optimization: A Quick-Start Guide. *Surveys in Operations Research and Management Science*, 20(2):35 – 42, 2015.

Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. CRISP-DM 1.0: Step-by-Step Data Mining Guide. Technical report, SPSS Inc., 2000.

Marco Chiarandini, Luis Paquete, Mike Preuss, and Enda Ridge. Experiments on Metaheuristics: Methodological Overview and Open Issues. Technical report, Institut for Matematik og Datalogi Syddansk Universitet, 2007.

Nicos Christofides. The Vehicle Routing Problem. *Revue française d'automatique, d'informatique et de recherche opérationnelle (RAIRO). Recherche opérationnelle*, 10(1):55 – 70, 1976. URL http://www.numdam.org/item?id=RO_1976__10_1_55_0.

Carlos Artemio Coello Coello and Margarita Reyes Sierra. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In *Proc. of the Third Mexican International Conference on Artificial Intelligence (MICAI)*, pages 688–697. Springer, 2004.

Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Replication and Comparison of Computational Experiments in Applied Evolutionary Computing: Common Pitfalls and Guidelines to Avoid Them. *Applied Soft Computing*, 19: 161 – 170, June 2014.

Harlan Crowder, Ron S Dembo, and John M Mulvey. On reporting computational experiments with mathematical software. *ACM Transactions on Mathematical Software (TOMS)*, 5(2):193–203, 1979.

Kenneth De Jong. Parameter Setting in EAs: a 30 Year Perspective. In *Parameter Setting in Evolutionary Algorithms*, pages 1 – 18. Springer, 2007.

Thomas Dean and Mark S. Boddy. An analysis of time-dependent planning. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI-88*, pages 49–54. AAAI Press/MIT Press, Menlo Park, CA, 1988. URL http://www.aaai.org/Conferences/AAAI/aaai88.php.

Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

Joaquín Derrac, Salvador García, Sheldon Hui, Ponnuthurai Nagaratnam Suganthan, and Francisco Herrera. Analyzing convergence performance of evolutionary algorithms: A statistical approach. *Information Sciences*, 289:41–58, 2014.

Jay L Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.

Luc Devroye. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1972.

Josef Dick and Friedrich Pillichshammer. *Digital Nets and Sequences: Discrepancy Theory and Quasi–Monte Carlo Integration*. Cambridge University Press, 2010.

Benjamin Doerr and Frank Neumann. *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization*. Springer, 2020.

Benjamin Doerr, Carola Doerr, and Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1479 – 1487. ACM, 2019.

Carola Doerr and Markus Wagner. On the Effectiveness of Simple Success-Based Parameter Selection Mechanisms for Two Classical Discrete Black-Box Optimization Benchmark Problems. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 943–950. ACM, 2018.

Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints*, art. arXiv:1810.05281, Oct 2018. Wiki page of IOHprofiler is available at https://iohprofiler.github.io/.

Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking Discrete Optimization Heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027, 2020.

Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

Tome Eftimov and Peter Korošec. The impact of statistics for benchmarking in evolutionary computation research. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1329–1336, 2018.

Tome Eftimov and Peter Korošec. Identifying practical significance through statistical comparison of meta-heuristic stochastic optimization algorithms. *Applied Soft Computing*, 85:105862, 2019.

Tome Eftimov, Peter Korošec, and Barbara Koroušić Seljak. A Novel Approach to Statistical Comparison of Meta-Heuristic Stochastic Optimization Algorithms Using Deep Statistics. *Information Sciences*, 417:186 – 215, 2017.

Tome Eftimov, Gašper Petelin, and Peter Korošec. Dsctool: A web-service-based framework for statistical comparison of stochastic optimization algorithms. *Applied Soft Computing*, 87:105977, 2020.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H. Hoos, and Kevin Leyton-Brown. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, volume 10, December 2013.

Ágoston Endre Eiben and Márk Jelasity. A Critical Note on Experimental Research Methodology in EC. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, volume 1, pages 582 – 587. IEEE, 2002.

Ágoston Endre Eiben and Selmar K Smit. Evolutionary Algorithm Parameters and Methods to Tune Them. In *Autonomous search*, pages 15 – 36. Springer, 2011.

Ágoston Endre Eiben and James E Smith. *Introduction to Evolutionary Computing*. Natural Computing. Springer, 2 edition, 2015.

Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60:25 – 64, 2010.

Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. COCO Documentation, Release 15.03, November 2015. URL http://coco.lri.fr/COCOdoc/COCO.pdf.

Andreas Fischbach and Thomas Bartz-Beielstein. Improving the reliability of test functions generators. *Applied Soft Computing*, 92:106315, 2020. ISSN 1568-4946. doi:https://doi.org/10.1016/j.asoc.2020.106315. URL http://www.sciencedirect.com/science/article/pii/S1568494620302556.

Philip J. Fleming and John J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM*, 29(3):218–221, March 1986.

Roger Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical analysis*, pages 73–89. Springer, 1976.

Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger H. Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. In *Proc. of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3397 —- 3403. AAAI Press, 2016.

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617, 2009.

Ian P. Gent and Toby Walsh. How not to do it. In *AAAI Workshop on Experimental Evaluation of Reasoning and Search Methods*, 1994.

Ian P. Gent, Stuart A. Grant, Ewen MacIntyre, Patrick Prosser, Paul Shaw, Barbara M. Smith, and Toby Walsh. How not to do it. Technical Report 97.27, School of Computer Studies, University of Leeds, May 1997.

Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

Bruce L Golden, Arjang A Assad, Edward A Wasil, and Edward Baker. Experimentation in optimization. *European Journal of Operational Research*, 27(1):1–16, 1986.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2018.

David Hadka and Patrick M. Reed. Diagnostic assessment of search controls and failure modes in many-objective evolutionary optimization. *Evolutionary Computation*, 20(3):423 – 452, 2012.

Raphael T. Haftka. Requirements for papers focusing on new or improved global optimization algorithms. *Structural and Multidisciplinary Optimization*, 54(1):1–1, 2016.

Doug Hains, L. Darrell Whitley, Adele E. Howe, and Wenxiang Chen. Hyperplane Initialized Local Search for MAXSAT. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 805 – 812. ACM, 2013.

Michael Pilegaard Hansen and Andrzej Jaszkiewicz. Evaluating the Quality of Approximations to the Non-Dominated Set. IMM Tech. Report IMM-REP-1998-7, Technical University of Denmark, Institute of Mathematical Modelling, Lyngby, Denmark, March 1998.

Nikolaus Hansen. Invariance, self-adaptation and correlated mutations in evolution strategies. In *Proc. of International Conference on Parallel Problem Solving from Nature*, pages 355–364. Springer, 2000. ISBN 978-3-540-45356-7.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-parameter black-box optimization benchmarking: Experimental setup. Technical report, Université Paris Sud, INRIA Futurs, Équipe TAO, Orsay, France, March 24, 2012. URL http://coco.lri.fr/BBOB-downloads/download11.05/bbobdocexperiment.pdf.

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. COCO: performance assessment. *CoRR*, abs/1605.03560, 2016a. URL http://arxiv.org/abs/1605.03560.

Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *arXiv preprint*, abs/1603.08785v3, August 2016b. URL http://arxiv.org/abs/1603.08785v3.

Anna Hart. Mann-whitney test is not just a test of medians: differences in spread can be important. *Bmj*, 323(7309): 391–393, 2001.

Jörg Heitkötter and David Beasley. The hitch-hiker's guide to evolutionary computation, 1994.

Michael Hellwig and Hans-Georg Beyer. Benchmarking Evolutionary Algorithms For Single-Objective Real-valued Constrained Optimization – A Critical Review. *Swarm and Evolutionary Computation*, 44:927–944, 2019.

John N Hooker. Needed: An Empirical Science of Algorithms. *Operations research*, 42(2):201 – 212, 1994.

John N Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1(1):33 – 42, 1995.

Simon Huband, Philip Hingston, Luigi Barone, and Lyndon While. A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit. *IEEE Transactions on Evolutionary Computation (TEVC)*, 10(5):477 – 506, 2006.

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507 – 523. Springer, 2011.

Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Thomas Marius Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. AClib: A Benchmark Library for Algorithm Configuration. In *Proc. of the 8th International Conference on Learning and Intelligent Optimization*, pages 36 – 40. Springer, 2014.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified Distance Calculation in Generational Distance and Inverted Generational Distance. In *Proc. of the 8th International Conference on Evolutionary Multi-Criterion Optimization*, pages 110 – 125. Springer, 2015.

W. James and Charles Stein. Estimation with quadratic loss. In *Proc. of the 4th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1: Contributions to the Theory of Statistics*, pages 361–379. University of California Press, 1961.

Alexandre D. Jesus, Arnaud Liefooghe, Bilel Derbel, and Luís Paquete. Algorithm Selection of Anytime Algorithms. In *Proc. of the 2020 Genetic and Evolutionary Computation Conference*, pages 850 – 858. ACM, 2020.

Yaochu Jin and Jürgen Branke. Evolutionary Optimization in Uncertain Environments – A Survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303 – 317, 2005.

David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. Part I, Graph Partitioning. *Operations Research*, 37(6):865 – 892, 1989.

David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378 – 406, 1991.

David Stifler Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. In *Proc. of a DIMACS Workshop on Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 of *DIMACS – Series in Discrete Mathematics and Theoretical Computer Science*, pages 215–250, 2002a.

David Stifler Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, 59:215 – 250, 2002b.

David Stifler Johnson and Lyle A. McGeoch. Experimental Analysis of Heuristics for the STSP. In *The Traveling Salesman Problem and its Variations*, volume 12 of *Combinatorial Optimization*, chapter 9, pages 369 – 443. Kluwer Academic Publishers, 2002.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

G. Karafotias, M. Hoogendoorn, and A. E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, April 2015.

Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, 1993.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.

Pascal Kerschke and Heike Trautmann. Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation (ECJ)*, 27(1): 99 – 127, 2019a.

Pascal Kerschke and Heike Trautmann. Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco. In *Applications in Statistical Computing*, pages 93 – 123. Springer, 2019b.

Pascal Kerschke, Jakob Bossek, and Heike Trautmann. Parameterization of State-of-the-Art Performance Indicators: A Robustness Study based on Inexact TSP Solvers. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1737 – 1744. ACM, 2018a.

Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H. Hoos, and Heike Trautmann. Leveraging TSP Solver Complementarity through Machine Learning. *Evolutionary Computation (ECJ)*, 26(4):597 – 620, December 2018b.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation (ECJ)*, 27:3 – 45, 2019a.

Pascal Kerschke, Hao Wang, Mike Preuss, Christian Grimme, André H. Deutz, Heike Trautmann, and Michael T. M. Emmerich. Search Dynamics on Multimodal Multi-Objective Problems. *Evolutionary Computation (ECJ)*, 27: 577 – 609, 2019b.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598): 671 – 680, 1983.

Jack P. C. Kleijnen. Analyzing Simulation Experiments with Common Random Numbers. *Management Science*, 34 (1):65 – 74, 1988.

Jack P. C. Kleijnen. Experimental Design for Sensitivity Analysis of Simulation Models. Workingpaper, Operations Research, 2001.

Jack P. C. Kleijnen. Design and Analysis of Simulation Experiments. In *International Workshop on Simulation*, pages 3–22. Springer, 2015.

Jack P. C. Kleijnen. Regression and Kriging Metamodels with their Experimental Designs in Simulation: A Review. *European Journal of Operational Research*, 256(1):1–16, 2017.

Lasse Kliemann and Peter Sanders. *Algorithm Engineering: Selected Results and Surveys*, volume 9220. Springer, 2016.

Joshua Damian Knowles and David Corne. On Metrics for Comparing Nondominated Sets. In *Proc. of the 2002 Congress on Evolutionary Computation*. IEEE, 2002.

William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

Abhishek Kumar, Guohua Wu, Mostafa Z. Ali, Rammohan Mallipeddi, Ponnuthurai Nagaratnam Suganthan, and Swagatam Das. Guidelines for real-world single-objective constrained optimisation competition. Technical report, 2020.

Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

Howard Levene. Robust tests for equality of variances. *Contributions to probability and statistics. Essays in honor of Harold Hotelling*, pages 279–292, 1961.

Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-Objective Evolutionary Algorithms: A Survey. *ACM Computing Surveys*, 48(1), September 2015.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18:185:1–185:52, 2017.

Rui Li, Michael T. M. Emmerich, Jeroen Eggermont, Ernst G. P. Bovenkamp, Thomas Bäck, Jouke Dijkstra, and Johan H. C. Reiber. Mixed-Integer NK Landscapes. In *Proc. of Parallel Problem Solving from Nature*, pages 42–51. Springer, 2006.

Tianjun Liao, Krzysztof Socha, Marco A Montes de Oca, Thomas Stützle, and Marco Dorigo. Ant Colony Optimization for Mixed-Variable Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 18(4): 503 – 518, 2013.

Arnaud Liefooghe, Manuel López-Ibáñez, Luís Paquete, and Sébastien Verel. Dominance, Epsilon, and Hypervolume Local Optimal Sets in Multi-Objective Optimization, and How to Tell the Difference. In *Proc. of the 20th Annual Conference on Genetic and Evolutionary Computation*, pages 324 – 331. ACM, 2018.

Harold R Lindman. *Analysis of variance in complex experimental designs.* WH Freeman & Co, 1974.

Hai-Lin Liu, Lei Chen, Kalyanmoy Deb, and Erik D. Goodman. Investigating the Effect of Imbalance Between Convergence and Diversity in Evolutionary Multiobjective Algorithms. *IEEE Transactions on Evolutionary Computation (TEVC)*, 21(3):408–425, September 2017.

Jialin Liu, Antoine Moreau, Mike Preuss, Baptiste Rozière, Jérémy Rapin, Fabien Teytaud, and Olivier Teytaud. Versatile black-box optimization. *CoRR*, abs/2004.14014, 2020. URL https://arxiv.org/abs/2004.14014.

Qunfeng Liu, William V. Gehrlein, Ling Wang, Yuan Yan, Yingying Cao, Wei Chen, and Yun Li. Paradoxes in Numerical Comparison of Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*, pages 1–15, 2019.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9(Nov):2579–2605, 2008.

Katherine Mary Malan and Andries Petrus Engelbrecht. A Survey of Techniques for Characterising Fitness Landscapes and Some Possible Ways Forward. *Information Sciences (JIS)*, 241:148 – 163, 2013.

Jiří Matoušek. *Geometric Discrepancy.* Springer, Berlin, 2 edition, 2009.

Deborah G Mayo and Aris Spanos. Severe testing as a basic concept in a neyman–pearson philosophy of induction. *The British Journal for the Philosophy of Science*, 57(2):323–357, 2006.

Kent McClymont and Ed Keedwell. Benchmark Multi-Objective Optimisation Test Problems with Mixed Encodings. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, pages 2131 – 2138. IEEE, 2011.

Catherine C McGeoch. *Experimental Analysis of Algorithms.* PhD thesis, Carnegie Mellon University, Pittsburgh PA, 1986.

Catherine C McGeoch. Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996.

Michael D. McKay, Richard J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. doi:10.2307/1268522.

Ivan Reinaldo Meneghini, Marcos Antonio Alves, António Gaspar-Cunha, and Frederico Gadelha Guimaraes. Scalable and customizable benchmark problems for many-objective optimization. *Applied Soft Computing*, 90:106139, 2020.

Olaf Mersmann, Mike Preuss, and Heike Trautmann. Benchmarking Evolutionary Algorithms: Towards Exploratory Landscape Analysis. In *Proc. of International Conference on Parallel Problem Solving from Nature*, pages 73 – 82. Springer, 2010.

Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory Landscape Analysis. In *Proc. of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829 – 836. ACM, 2011.

Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. A Novel Feature-Based Approach to Characterize Algorithm Performance for the Traveling Salesperson Problem. *Annals of Mathematics and Artificial Intelligence*, 69(2):151 – 182, 2013.

Laurent Meunier, Yann Chevaleyre, Jérémy Rapin, Clément Royer, and Olivier Teytaud. On averaging the best samples in evolutionary computation. In *Proc. of Parallel Problem Solving from Nature*. Springer, 2020a. Full version available online at https://arxiv.org/abs/2004.11685.

Laurent Meunier, Carola Doerr, Jérémy Rapin, and Olivier Teytaud. Variance reduction for better sampling in continuous domains. *CoRR*, abs/2004.11687, 2020b. URL https://arxiv.org/abs/2004.11687.

Douglas C Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 9 edition, 2017.

Jorge J Moré and Stefan M Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

Jorge J Moré, Burton S Garbow, and Kenneth E Hillstrom. Testing Unconstrained Optimization Software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17 – 41, 1981.

Mario Andrés Muñoz Acosta, Michael Kirley, and Saman K. Halgamuge. Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content. *IEEE Transactions on Evolutionary Computation (TEVC)*, 19(1):74 – 87, 2015a.

Mario Andrés Muñoz Acosta, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. Algorithm Selection for Black-Box Continuous Optimization Problems: A Survey on Methods and Challenges. *Information Sciences (JIS)*, 317: 224 – 245, 2015b.

Matthias Müller-Hannemann and Stefan Schirra. *Algorithm Engineering: Bridging the Gap Between Algorithm Theory and Practice*. Springer, 2010.

Mario A. Muñoz and Kate A. Smith-Miles. Performance Analysis of Continuous Black-Box Optimization Algorithms via Footprints in Instance Space. *Evolutionary Computation*, 25(4):529–554, December 2017.

Dima Nazzal, Mansooreh Mollaghasemi, H Hedlund, and A Bozorgi. Using Genetic Algorithms and an Indifference-Zone Ranking and Selection Procedure Under Common Random Numbers for Simulation Optimisation. *Journal of Simulation*, 6(1):56 – 66, 2012.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74, 2016.

Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):1–13, 2017.

Patryk Orzechowski, William La Cava, and Jason H Moore. Where are we now? a large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1183–1190, 2018.

Patryk Orzechowski, Franciszek Magiera, and Jason H Moore. Benchmarking manifold learning methods on a large collection of datasets. In *European Conference on Genetic Programming (Part of EvoStar)*, pages 135–150. Springer, 2020.

Tom Packebusch and Stephan Mertens. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49(16):165001, 2016. doi:10.1088/1751-8113/49/16/165001. URL https://doi.org/10.1088%2F1751-8113%2F49%2F16%2F165001.

Ingo Paenke, Jürgen Branke, and Yaochu Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *IEEE Trans. Evolutionary Computation*, 10(4):405–420, 2006.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto Local Optimum Sets in the Biobjective Traveling Salesman Problem: An Experimental Study. In *Metaheuristics for Multiobjective Optimisation*, pages 177 – 199. Springer, 2004.

Fortunato Pesarin. *Multivariate permutation tests: with applications in biostatistics*, volume 240. Wiley Chichester, 2001.

Robin L Plackett and J Peter Burman. The design of optimum multifactorial experiments. *Biometrika*, 33(4):305–325, 1946.

Sergey Polyakovskiy, Mohammad Reza Bonyadi, Markus Wagner, Zbigniew Michalewicz, and Frank Neumann. A Comprehensive Benchmark Set and Heuristics for the Traveling Thief Problem. In *Proc. of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 477 —- 484. ACM, 2014. ISBN 9781450326629.

Karl Raimund Popper. *The Logic of Scientific Discovery*. Hutchinson & Co, 2 edition, 1959.

Karl Raimund Popper. *Objective Knowledge: An Evolutionary Approach*. Oxford University Press, 1975.

Mike Preuss. *Experimentation in Evolutionary Computation*, pages 27–54. Springer, 2015.

Kenneth V. Price. Differential Evolution vs. The Functions of the 2nd ICEO. In *Proc. of the IEEE International Conference on Evolutionary Computation*, pages 153 – 157. IEEE, 1997.

F. Pukelsheim. *Optimal Design of Experiments*. Wiley, New York NY, 1993.

Jeremy Rapin and Olivier Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

Carlo Raquel and Xin Yao. Dynamic multi-objective optimization: A survey of the state-of-the-art. In Shengxiang Yang and Xin Yao, editors, *Evolutionary Computation for Dynamic Optimization Problems*, pages 85–106. Springer, 2013.

Ingo Rechenberg. *Evolutionsstrategie '94*. frommann-holzboog, 1994.

Nery Riquelme, Christian von Lücken, and Benjamín Barán. Performance Metrics in Multi-Objective Optimization. In *Proc. of the 2015 Latin American Computing Conference (CLEI)*, pages 1 – 11. IEEE, 2015.

Howard Harry Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175 – 184, 1960.

Jonathan Rowe and Michael Vose. Unbiased black box search algorithms. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 2035 – 2042. ACM, 2011.

Ranjit K Roy. *Design of experiments using the Taguchi approach: 16 steps to product and process improvement*. John Wiley & Sons, 2001.

Ragav Sachdeva, Frank Neumann, and Markus Wagner. The dynamic travelling thief problem: Benchmarks and performance of evolutionary algorithms, 2020.

Thomas J Santner, Brian J Williams, William I Notz, and Brain J Williams. *The design and analysis of computer experiments*, volume 1. Springer, 2003.

Thiago Santos and Sebastio Xavier. A convergence indicator for multi-objective optimization algorithms. Technical report, Federal University of Ouro Preto, Ouro Preto, MG, Brazil, October 2018. arXiv:1810.12140v1 [cs.NE] 29 Oct 2018.

Michael A. Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13:270–276, 1968.

Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, Fachbereich Verfahrenstechnik, Berlin, Germany, 1975.

David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24 (111):647–656, 1970.

Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

Yuhui Shi and Russell Eberhart. A Modified Particle Swarm Optimizer. In *Proc. of the 1998 IEEE International Conference on Evolutionary Computation, within the IEEE World Congress on Computational Intelligence*, pages 69–73. IEEE, 1998.

Ofer M. Shir, Carola Doerr, and Thomas Bäck. Compiling a Benchmarking Test-Suite for Combinatorial Black-Box Optimization: A Position Paper. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 1753 – 1760. ACM, 2018.

Urban Škvorc, Tome Eftimov, and Peter Korošec. Understanding the Problem Space in Single-Objective Numerical Optimization Using Exploratory Landscape Analysis. *Applieed Soft Computing (ASOC)*, 90:106138, 2020.

Kate Smith-Miles and Thomas T. Tan. Measuring Algorithm Footprints in Instance Space. In *2012 IEEE Congress on Evolutionary Computation*. IEEE, 2012.

Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

Charles Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proc. of the 3rd Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1: Contributions to the Theory of Statistics*, pages 197–206. University of California Press, 1956.

Jörg Stork, Ágoston Endren Eiben, and Thomas Bartz-Beielstein. A new taxonomy of continuous global optimization algorithms, 2018. URL https://arxiv.org/abs/1808.08818/. arXiv e-prints:1808.08818.

El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons Inc., July 2009. ISBN 978-0-470-27858-1.

Ryoji Tanabe and Hisao Ishibuchi. An Easy-to-Use Real-World Multi-Objective Optimization Problem Suite. *Applied Soft Computing (ASOC)*, 89:106078, 2020.

John Wilder Tukey. *Exploratory Data Analysis*, volume 2. Reading, MA, 1977.

Tea Tušar, Dimo Brockhoff, and Nikolaus Hansen. Mixed-integer benchmark problems for single- and bi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 718 —- 726. ACM, 2019.

Niki Veček, Marjan Mernik, and Matej Črepinšek. A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms. *Information Sciences*, 277:656–679, 2014.

Vanessa Volz, Boris Naujoks, Pascal Kerschke, and Tea Tušar. Single- and Multi-Objective Game-Benchmark for Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 647 – 655. ACM, 2019.

Tobias Wagner. A subjective review of the state of the art in model-based parameter tuning. In Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss, editors, *Workshop on Experimental Methods for the Assessment of Computational Systems (WEMACS 2010)*, Algorithm Engineering Report, pages 1–13. TU Dortmund, Faculty of Computer Science, Algorithm Engineering (Ls11), 2010.

Thomas Weise. jsspinstancesandresults: Results, data, and instances of the job shop scheduling problem, 2019. URL http://github.com/thomasWeise/jsspInstancesAndResults/. A meta-study of 145 algorithm setups from literature on the JSSP.

Thomas Weise and Zijun Wu. Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 1769 — 1776. ACM, 2018. ISBN 9781450357647.

Thomas Weise, Raymond Chiong, Ke Tang, Jörg Lässig, Shigeyoshi Tsutsui, Wenxiang Chen, Zbigniew Michalewicz, and Xin Yao. Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 9:40–52, August 2014.

L. Darrell Whitley, Soraya B. Rana, John Dzubera, and Keith E. Mathias. Evaluating Evolutionary Algorithms. *Artificial Intelligence (AIJ)*, 85(1-2):245 – 276, 1996.

L. Darrell Whitley, Jean-Paul Watson, Adele Howe, and Laura Barbulescu. Testing, Evaluation and Performance of Optimization and Learning Systems. In Ian C. Parmee, editor, *Adaptive Computing in Design and Manufacture V*, pages 27 – 39, London, 2002. Springer.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

Guohua Wu, Rammohan Mallipeddi, and Ponnuthurai Nagaratnam Suganthan. Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization. Technical report, National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, September 2017.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-Based Algorithm Selection for SAT. *Journal of Artificial Intelligence Resesearch (JAIR)*, 32:565–606, 2008.

Aimin Zhou, Qingfu Zhang, and Yaochu Jin. Approximating the Set of Pareto-optimal Solutions in Both the Decision and Objective Spaces by an Estimation of Distribution Algorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1167 – 1189, 2009.

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

Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Eidgenössische Technische Hochschule Zürich, Institut für Technische Informatik und Kommunikationsnetze, Zürich, Switzerland, November 1999. URL https://sop.tik.ee.ethz.ch/publicationListFiles/zitz1999a.pdf. TIK-Schriftenreihe Nr. 30.