Optim Eng (2017) 18:815–848
DOI 10.1007/s11081-017-9366-1

Best practices for comparing optimization algorithms

Vahid Beiranvand[1] · Warren Hare[2] · Yves Lucet[1]

Received: 3 June 2016 / Revised: 6 March 2017 / Accepted: 23 August 2017 /
Published online: 19 September 2017

Abstract Comparing, or benchmarking, of optimization algorithms is a complicated task that involves many subtle considerations to yield a fair and unbiased evaluation. In this paper, we systematically review the benchmarking process of optimization algorithms, and discuss the challenges of fair comparison. We provide suggestions for each step of the comparison process and highlight the pitfalls to avoid when evaluating the performance of optimization algorithms. We also discuss various methods of reporting the benchmarking results. Finally, some suggestions for future research are presented to improve the current benchmarking process.

Keywords Benchmarking · Algorithm comparison · Guidelines · Performance · Software · Testing · Metric · Timing · Optimization algorithms

1 Introduction

As the number of optimization methods, and implementations of those methods, has increased, researchers have pursued comparative studies to evaluate their performance. When done well, such studies can be of great value in helping end-users choose the most suitable optimization method for their problems. Such studies are generally referred to as optimization benchmarking.

In the most general sense, benchmarking is the comparison of one or more products to an industrial standard product over a series of performance metrics. In the case of benchmarking optimization algorithms, the products are the specific

✉ Yves Lucet
yves.lucet@ubc.ca

[1] Department of Computer Science, University of British Columbia, Kelowna, BC, Canada

[2] Department of Mathematics, University of British Columbia, Kelowna, BC, Canada

implementations of given algorithms, and the performance metrics are generated by running the implementations on a series of test problems. This framework presents a certain clarity in benchmarking optimization algorithms, as there is at least some agreement on what constitutes "better". If one algorithm runs faster, uses less memory, and returns a better final function value, on all possible problems, then it can be considered better than the alternative. Of course, in practice such a clear conclusion seldom arises. Thus, interpreting the conclusions of algorithmic comparisons can be tricky.

Nonetheless, when done well, benchmarking optimization algorithms can have great practical value. It can reveal both strengths and weaknesses of an algorithm, which allows for better research focus. It can aid in determining if a new version of optimization software is performing up to expectations. And, it can help guide end-users in selecting a good choice of algorithm for a particular real-world problem.

However, when done poorly, benchmarking optimization algorithms can also be misleading. It can hide algorithm's weaknesses (or strengths), report improvements that do not exist, or suggest the incorrect algorithmic choice for a given situation.

In optimization benchmarking many subjective choices are made, such as the test set to solve, the computing environment to use, the performance criteria to measure, etc. Our primary objective in this paper is to help researchers to design a proper benchmarking approach that is more comprehensive, less biased, and less subject to variations within a particular software or hardware environment. Our secondary objective is to provide a comprehensive review of the benchmarking literature for optimization algorithms.

In pursuing these objectives, we focus on single-objective optimization algorithms that run in serial (i.e., that do not use parallel processing). Comparing algorithms for multi-objective optimization, or optimization algorithms that use parallel processing, introduces new levels of complexity to the benchmarking process. While we provide some comments on the challenges for benchmarking some algorithms in the conclusions, we consider these issues outside of the scope of this paper.

We also note that much of the presentation within this paper discusses algorithms as if the underlying optimization problem is a continuous unconstrained problem. This is for ease of presentation, and in most cases translating the ideas to other styles of optimization problems is clear. As such, we limit ourselves to discussing specific styles of optimization problems only when the translation is not straightforward.

1.1 Historical overview of benchmarking in optimization

We begin with a brief historical overview of optimization benchmarking.

One of the very first studies in benchmarking of algorithms was given by Hoffman et al. (1953), in which three different methods for linear programming were compared. Although this computational experiment was performed early in the development of computers, when there existed almost no compiler and programming environment, the reported techniques have been used for a long time and can be considered as the foundation of the current comparison techniques. They include such ideas as using test sets to compare algorithms, employing performance

measures (accuracy, CPU time, number of iterations, and convergence rate), and paying attention to the impact of coding on the performance of algorithms.

Another early contribution to the field of benchmarking is Box's work from 1966 (Box 1966). In this work, Box evaluates the performances of eight algorithms for unconstrained optimization using a collection of 5 test problems with up to 20 variables. He considers the number of function evaluations, the importance of model size and the generality of the optimization algorithms.

In the late 1960s, optimization benchmarking research began to expand rapidly. Comparative studies have been performed throughout the optimization literature, for example in unconstrained optimization (Tabak 1969; Huang and Levy 1970; Moré et al. 1981), constrained optimization (Beltrami 1969; Schittkowski and Stoer 1978; Famularo et al. 2002) nonlinear least squares (Bard 1970; Ramsin and Wedin 1977; Vanderbei and Shanno 1999; McGeoch 2002), linear programming (Mulvey 1982; Baz et al. 2007; Berthold 2013), nonlinear programming (Colville 1968; Bard 1970; Asaadi 1973; Schittkowski 1980; Eason and Fenton 1974; Sandgren and Ragsdell 1980a, b; Eason 1982; Bongartz et al. 1997; Vanderbei and Shanno 1999; Benson et al. 2003, 2004; Hough et al. 2001; Yeniay 2005; Tedford and Martins 2010), geometric programming (Dembo 1978; Rijckaert and Martens 1978), global optimization (Hansen et al. 1992; Pintér 2002; Bussieck et al. 2003; Khompatraporn et al. 2005; Neumaier et al. 2005; Regis and Shoemaker 2007; Strongin and Sergeyev 2000; Zhigljavsky and Žilinskas 2008; Kvasov and Mukhametzhanov 2016), derivative-free optimization (Hare and Wang 2010; Rios and Sahinidis 2013; Zhang 2014; Sergeyev and Kvasov 2006; Paulavičius et al. 2014), and other areas of optimization (Tabak 1969; Huang and Levy 1970; Miele et al. 1972; Houstis et al. 1988; Mittelmann 2003; Opara and Arabas 2011; Parejo et al. 2012)—amongst many more.

In addition, a few researchers have focused on improving the (optimization) benchmarking process. Crowder et al. (1979) presented the first study that attempted to provide standards and guidelines on how to benchmark mathematical algorithms. It includes a detailed discussion of experimental design and notes the necessity of a priori experimental design. The authors pay attention to reproducibility of the results and provide a method for reporting the results. Similar research conducted by Jackson et al. (1990) delivered an updated set of guidelines. Dolan and Moré (2002) introduced performance profiles, which have rapidly become a gold standard in benchmarking of optimization algorithms with more recent work pointing out its limitations (Gould and Scott 2016). In this paper, we attempt to provide a modern picture of best-practice in the optimization benchmarking process.

1.2 Paper framework

We now provide a general framework for benchmarking optimization algorithms, which we also use to structure discussion in the paper.

1. *Clarify the reason for benchmarking* In Sect. 2, we discuss some of the common reasons to compare optimization algorithms, and some of the pitfalls that arise when the purpose of benchmarking is unclear.
2. *Select the test set* In Sect. 3, a review of test sets for various problem categories is presented, the challenges related to test sets are discussed, and some guidelines are provided for assembling an appropriate test set.
3. *Perform the experiments* In Sect. 4, we review and discuss various considerations related to the critical task of designing experiments, including performance measures, tuning parameters, repeatability of the experiments, and ensuring comparable computational environments.
4. *Analyze and report the results* Section 5 contains a review of different reporting methods for optimization algorithms, including tabular methods, trajectory plots, and ratio-based plots (such as performance and data profiles).

In addition to the aforementioned sections, Sect. 6 contains a review of recent advances in the field of automated benchmarking and Sect. 7 presents some concluding thoughts.

2 Reason for benchmarking

Having a clear understanding of the purpose of a numerical comparison is a crucial step that guides the rest of the benchmarking process. While seemingly self-evident, it is surprisingly easy to neglect this step. Optimization benchmarking has been motivated by a variety of objectives. For example:

1. To help select the best algorithm for working with a real-world problem.
2. To show the value of a novel algorithm, when compared to a more classical method.
3. To compare a new version of optimization software with earlier releases.
4. To evaluate the performance of an optimization algorithm when different option settings are used.

In a practical sense, all of these work towards gathering information in order to rank optimization algorithms within a certain context. However, the context can, and should, play a major role in guiding the rest of the benchmarking process.

For example, if the goal is to select the best algorithm for a particular real-world application, then the test problems (Sect. 3) should come from examples of that application.

Alternatively, if the goal is to show the value of a new optimization algorithm, then it is valuable to think about exactly where the algorithm differs from previous methods. Many new algorithms are actually improvements on how a classical method deals with some aspect of an optimization problem. For example, in Hare and Sagastizábal (2010) the authors develop a new method to deal with nonconvexity when applying a *proximal-bundle method* to a *nonsmooth optimization problem*. As such, to see the value of the method, the authors compare it against

other proximal-bundle methods on a collection of nonconvex nonsmooth optimization problems. If they had compared their method against a quasi-Newton method on smooth convex optimization problems, then very little insight would have been gained.

Regardless of the reason, another question researchers must consider is what aspect of the algorithm is most important. Is a fast algorithm that returns infeasible solutions acceptable? Is it more important that an algorithm solves every problem, or that its average performance is very good? Is the goal to find a global minimizer, or a highly accurate local minimizer? The answers to these questions should guide the choice of performance metrics that need to be collected (Sect. 4) and how they should be analyzed (Sect. 5). Answering these types of questions before running the experiments is time well spent.

3 Test sets

A test problem contains a test function together with some further criteria such as the constraint set, feasible domain, starting points. A test set is a collection of test problems. Obviously, benchmarking yields meaningful results only when competing algorithms are evaluated on the same test set with the same performance measures.

The selection of the appropriate test sets to benchmark the performance of optimization algorithms is a widely noticed issue among researchers (Pintér 2007; Dolan and Moré 2004; Jackson et al. 1990; Sergeyev et al. 2013; Zhigljavsky and Žilinskas 2008). Generally, there are three sources for collecting test problems: real-world problems, pre-generated problems, and randomly-generated problems. Real-world problems can be found through instances of specific applications, and pre-

Table 1 Some test sets reported in the literature

Collection type	Resources
Unconstrained optimization problems	Ali et al. (2005), Moré et al. (1981), Andrei (2008) and Jamil and Yang (2013)
Global optimization	GAMS (Pintér 2007), COCONUT (Shcherbina et al. 2003), and other collections (Floudas and Pardalos 1990; Floudas et al. 1999; Schoen 1993; Pintér and Kampas 2013; Addis and Locatelli 2007; Törn et al. 1999; Famularo et al. 2002)
Linear programming	Netlib
Local optimization	Floudas et al. (1999) and Floudas and Pardalos (1990)
Nonlinear optimization problems	CUTEr (Buckley 1992; Bongartz et al. 1995), CUTEst (Gould et al. 2015), COPS (Bondarenko et al. 1999; Dolan and Moré 2004), and collections (Hock and Schittkowski 1981; Schittkowski 2008; Averick et al. 1991; Dembo 1976; Dolan and Moré 2000)
Mixed integer linear programming	MIPLIB (Romesis et al. 2003; Koch et al. 2011)

generated problems exist in common test set libraries; see Table 1. Conversely, randomly-generated test problems are often more ad hoc in nature, with researchers creating methods to randomly generate test problems that are used in only a single paper (see Nash and Nocedal 1991; Hare and Sagastizábal 2010; Hare and Planiden 2014, among many others). However, some researchers have gone to the effort to study methods to randomly generate test problems for a given area; some examples appear in Table 2.

While the real-world test sets provide specialized information about the performance of the optimization algorithms within a specific application, the results may be difficult to generalize. The difficulties lie in the facts that real-world test sets are often small and the problems are often application-specific. Nonetheless, if the goal is to determine the best algorithm to use for a particular real-world application, then a real-world test set focused on that application is usually the best option.

On the other hand, the artificial and randomly-generated test sets can provide useful information about the algorithmic characteristics of optimization algorithms. Artificial and randomly-generated test sets can be extremely large, thereby providing an enormous amount of comparative data. However, it can be difficult to rationalize their connection to the real-world performance of optimization algorithms. If the goal is to compare a collection of algorithms across a very wide spectrum, then artificial and randomly-generated test sets are usually the better option.

When selecting a test set, it is always important to keep the particular goal of the comparison in mind. Regardless of the goal, an appropriate test set should generally seek to avoid the following deficiencies.

1. *Too few problems* Having more problems in the test set makes the experiment more reliable and helps the results reveal more information about the strengths or weaknesses of the evaluated algorithms.
2. *Too little variety in problem difficulty* A test set containing only simple problems is not enough to identify the strengths and weaknesses of algorithms. In contrast, a test set which has only problems that are so difficult that no algorithm can solve them, clearly, does not provide useful information on the relative performance of algorithms.

Table 2 Some test problem generators reported in the literature

Test problem generators	
Network programming	Elam and Klingman (1982)
Nonlinear optimization problems	Schittkowski (1980) and Liu and Zhang (2000)
Combinatorial problems	Grundel and Jeffcoat (2009) and Elam and Klingman (1982)
Quadratic programming	Lenard and Minkoff (1984)
Global optimization	Schoen (1993), Gaviano et al. (2003), Addis and Locatelli (2007) and Ng and Li (2014)

3. *Problems with unknown solutions* When possible, it is better to use test problems where the solution is known. Depending on the analysis performed (see Sect. 5), the "solution" could be interpreted as the minimum function value, or the set of global minimizers. Having access to the solution greatly improves the ability to evaluate the quality of the algorithmic output. However, when the test set is comprised of real-world test problems, then a lack of known solutions may need to be accepted as inevitable.
4. *Biased starting points* Allowing different algorithms to use different starting-points will obviously bias the result. However, more subtle problems can also exist. For example, if a starting point lies on the boundary of a constraint set, then an interior point method will be severely disadvantaged. Another example comes from considering the Beale test function, which has a global minimizer at (3, 0.5) (Moré et al. 1981). If a *compass search* (see, e.g., Kolda et al. 2003) with an initial step length of 1 is started at (0.5, 0.5), then it will converge to the exact minimizer in just four iterations. However, if a starting point of (0.51, 0.51) is used, then the exact same algorithm will use 63 iterations to reach a point within 10^{-2} of the global minimizer.[1]
5. *Hidden structures* Many test sets have some structure that is not realistic in real-world problems. For example, about 50% of the problems in the test set (Moré et al. 1981) have solution points that occur at integer-valued coordinates. An algorithm that employs some form of rounding may perform better than usual on these problems.

Thus, when choosing test sets for the benchmarking task the following considerations should be taken into account as much as possible.

1. If the test set contains only a few problems, then the experiment should be referred to as a *case study* or a *proof of concept*, but not benchmarking. While there is no fixed number that determines how many problems is enough to be considered benchmarking, we recommend that in order to achieve a reliable conclusion about the performance, an experiment should contain at least 20 test problems (preferably more). In the specific case of comparing a new version of an optimization algorithm with a previous version, the number of test problems should be significantly greater—in the order of 100 or more. In all cases, the more problems tested, the more reliable the conclusions.
2. When possible, a test set should include at least two groups of problems: an *easy group* which consists of the problems that are easy to solve within a reasonable time on a regular contemporary computer using all the optimization algorithms tested, and a *hard group* that contains the problems which are solvable but computationally expensive and may require a specific optimization algorithm.
3. Whenever possible, ensure that at least a portion of the test set includes problems with known solutions.

[1] Note that this example is artificially constructed to emphasize the results; the recommended starting point for the Beale test problem is (1, 1).

4. For test sets that include starting points, new starting points can be generated by introducing small (random) perturbations to the given starting points. For other test sets, randomly-generated starting points can be created from scratch. In either case, all starting points should be created for each problem, and then every algorithm should be provided the same starting point for testing. This approach can be further used to increase result reliability, by repeating tests on the same function with a variety of starting points.
5. Examine the test set with a critical eye and try to determine any hidden structure. Some structures can be removed through methods similar to the random perturbation of starting points in (4). One quick test is to set an algorithm to minimize $f(x)$ starting at x^0 and then set the algorithm to minimize $\hat{f}(x) = f(x - p)$ starting from $\hat{x}^0 = x^0 - p$ (where p is any random point). Constraints can then be shifted in a similar manner, effectively shifting the entire problem horizontally by the vector p. While it relocates the origin, and moves any constraints away from special integer values, it has no theoretical effect on the geometry of the problem. As such, the results of both tests should be very close (theoretically they should be identical, but numerical errors may cause some deviation). If the results differ, then perhaps some hidden structure is being exploited by the algorithm, or perhaps hidden constraints are causing issues. Regardless of the reason, the researcher should recognize the issue and consider a wider test set.

Using suitable standard test sets is usually a good option when benchmarking optimization algorithms. In particular, it is usually easier to compare results across research groups when standard tests are employed, although even within a specific research field there is generally no consensus on the appropriate test set to draw specific conclusions. Many interesting and diverse test sets have been reported in the literature; see Tables 1 and 2.

Producing random test sets using test problem generators has its own drawbacks. Are the generated problems representative or difficult? Is there any hidden structure in the problems? Some papers that use random test problem generators are listed in Table 2.

Figure 1 shows a decision tree that summarizes the fundamental decisions required for assembling an appropriate test set for benchmarking of optimization algorithms.

4 Performing the experiments

The performance of algorithms is influenced by two general types of factors: *environmental factors* and *algorithmic factors*.

Environmental factors refer to factors that are out of the algorithm scope and usually beyond the control of the researcher. A common example is the computer environment used to test the algorithms, which includes processor speed, operating system, computer memory, etc. Environmental factors may also include the

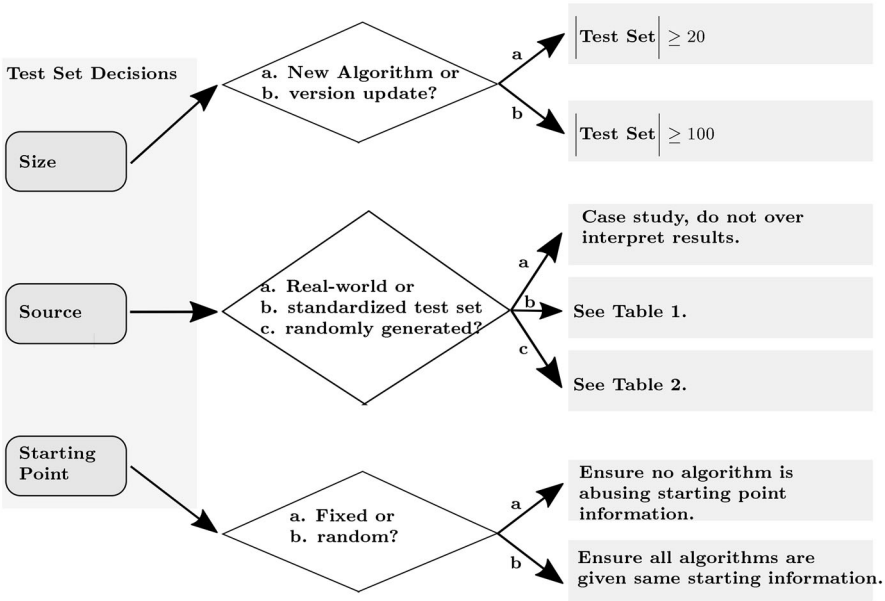

Fig. 1 Test set decision tree

programmer's skill and the programming language/compiler used. This is particularly evident when multiple pieces of software by a variety of programmers are being compared. In essence, if the benchmarking process is repeated by another researcher elsewhere, then environmental factors are unlikely to remain constant, and so the benchmarking results are expected to change.

Algorithmic factors are related to the algorithm itself. These are factors that are considered global across a variety of computing platforms. If the software is programmed by the researcher, then it is assumed these factors are independent of the implementation aspects of the algorithm.

Optimization benchmarking seeks to measure the algorithmic factors, and proceeds under the key *assumption* that, while environmental factors are expected to change the results, the algorithmic factors are sufficiently strong that the general ranking of algorithms should remain constant under the specific ranges of parameters under consideration.

To compare algorithms, it is necessary to collect data that measures the overall performance of each algorithm. This is done by running each algorithm on the test set (discussed in Sect. 3), and collecting data on the results. The data collection and the selection of performance measures is based on the research questions motivating the experimental study. In general, performance measures fall into three categories: efficiency, reliability, and quality of algorithmic output. We discuss these performance categories in Sects. 4.1, 4.2, and 4.3. Table 3 provides a classification of the comparative measures for optimization algorithms based partly on the guidelines provided by Hoffman and Jackson (1982).

Table 3 Comparative measures

Performance category	Example criteria
Efficiency	1. Number of fundamental evaluations
	2. Running time
	3. Memory usage
Reliability	1. Success rate
	2. Number of constraint violations
	3. Percentage of global solutions found
Quality of solution	1. Fixed-cost solution result
	2. Fixed-target solve time
	3. Computational accuracy

In order to allow for maximal data analysis (and thereby the best understanding of overall performance), it is recommended to collect at least some data from every performance category.

4.1 Efficiency

The efficiency of an optimization algorithm refers to the computational effort required to obtain a solution. In mathematical programming, there are two primary measures of efficiency: the number of fundamental evaluations and the running time. A third, less common, measure is memory usage.

4.1.1 Number of fundamental evaluations

The term fundamental evaluation is used to refer to any subroutine that is called by the algorithm in order to gain fundamental information about the optimization problem. The most obvious example is an objective function evaluation but the evaluation may involve complex simulation algorithms. Other fundamental evaluations could include gradient evaluations, Hessian evaluations, or constraint function evaluations. The number of fundamental evaluations can be used as a standard unit of time, and is often assumed to be platform independent. In many situations, the number of fundamental evaluations is a particularly important measure, as for real-world problems these evaluations often dominate the internal workings of the algorithm (Hock and Schittkowski 1983; Crowder et al. 1979; Dixon and Szegö 1978; Conn et al. 1996; Barton 1987; Nash and Nocedal 1991; Vanden Berghen and Bersini 2005; Huang and Levy 1970; Miele et al. 1972; Asaadi 1973; Shcherbina et al. 2003; Neumaier et al. 2005; Eason 1982; Audet et al. 2014b; Evtushenko 1985; Paulavičius et al. 2014; Kvasov and Sergeyev 2015). Note however that this measure is unreasonable when fundamental evaluations do not dominate the internal workings of the algorithm (Ali et al. 2005).

4.1.2 Running time

Running time, as a measure for optimization benchmarking, is usually measured by either CPU time or wall clock time.[2] Wall clock time contains CPU time, and has been argued to be more useful in real-world settings (McGeoch 2002). However, wall clock time is not reproducible or verifiable since it is tied to a specific hardware platform and software combination. CPU time is considerably more stable, as it is independent of background operations of the computer. Moreover, CPU time is more-or-less consistent for the same version of an operating system running on the same computer architecture.

It should be noted that, due to the wide variety and complexity of modern computing architectures, the number of situations in which time is dominated by memory access costs is increasing, hence the precision of CPU timers has been reduced. To improve the precision of CPU timers, tools such as cache and memory access tracers can help obtaining a more accurate CPU time performance. For a more detailed discussion of these techniques we refer to Knuth (1994) and LaMarca and Ladner (1996).

Another issue with CPU time is the increasing prevalence of multi-core machines. Thorough reporting would require indicating the number of cores and the CPU time for each core, but also how efficiently the different levels of memory were used and cache hits/misses. Since such measurements are not straightforward to obtain for multi-core machines, the wall-clock time along with the hardware specifications are usually reported. (Unless the new algorithm contribution focuses specifically on optimizing computation for a multi-core architecture, in which case more precise measures are warranted.) Eventually, the onus is on the researchers to explain how simplified measurements support the conclusions drawn; this is especially true for multi-core machines.

Regardless of whether wall clock time or CPU time is used, in order to maximize the reliability of the data, it is important to ensure that any background operations of the computer are kept to a minimum. Furthermore, any manuscript regarding the benchmarking should clearly state which form of running time was collected.

4.1.3 Other measures

In addition to the categorization presented above, in some specific cases, there is another issue that influences the choice of an appropriate measure for running time: *the type of algorithm*. For example, to evaluate the running time for branch-and-bound based algorithms, the number of branch-and-bound nodes is a common criterion, while for simplex and interior point based algorithms, the number of iterations is often used. Therefore, when deciding on the choice of a suitable efficiency measure, the type of algorithm to be evaluated should also be taken into account.

[2] *Wall clock time* refers to the amount of time the human tester has to wait to get an answer from the computer. Conversely, *CPU time* is the amount of time the CPU spends on the algorithm, excluding operating system tasks and other processes.

4.2 Reliability

The reliability and robustness of an optimization algorithm is defined as the ability of the algorithm to "perform well" over a wide range of optimization problems (Moré et al. 1981; Barr et al. 1995). The most common performance measure to evaluate the reliability is success rate (Törn and Žilinskas 1989; Rijckaert and Martens 1978; Eason 1982; Strongin and Sergeyev 2000). Success rate is gauged by counting the number of test problems that are successfully solved within a pre-selected tolerance. This can be done using objective function value, or distance of the solution point from a minimizer. In convex optimization these two approaches are largely, but not perfectly, interchangeable. However, if the objective function has multiple local minimizers, then the researcher must decide whether good local solutions are acceptable outcomes, or if the algorithm must converge to a global minimizer (Schittkowski 1980; Ramsin and Wedin 1977). In addition to the success rate, the average objective function values and the average constraint violation values have also been reported to measure reliability (Schittkowski 1980).

When studying reliability, the researcher should consider whether the algorithms are deterministic or non-deterministic, and repeat tests multiple times if the algorithm is non-deterministic. Reliability can be based on a fixed starting point (if one is given with the test set), but it is often better to use multiple starting points.

In deterministic optimization algorithms, reliability can be interpreted as the number of problems in the given test set that are solved by the optimization algorithm. When dealing with non-deterministic algorithms, it is important to repeat each test multiple times, to make sure that reliability is measured in aggregate, and not skewed by a single lucky (or unlucky) algorithmic run.

Using multiple repeats of each test raises the issue of how to aggregate the results. One option is to consider each algorithmic run as a separate test problem and then compare solvers across this expanded test set. This allows comparisons based on worst-case or best-case scenarios. Another option is to use averaged data, for example, average runtime, average solution accuracy, average reliability, etc. If averaging is used, then it is important to also include standard deviations of the data. In either case, data collection is best performed by considering each algorithmic run as a separate test problem, as average values can easily be extracted from this data, while reconstructing the full test data from averaged values is not possible.

It should be noted that in some cases multiple repeats of a non-deterministic method is impractical due to the time it takes to solve a single problem.

4.2.1 Multiple starting points

As mentioned in Sect. 3, many academic test problems come with suggested starting points. While algorithms should always be tested using these starting points, it is often enlightening to test the algorithm using other starting points as well. Most deterministic algorithms should show little change in performance if a starting point is perturbed by a small random vector—provided the new starting point retains whatever feasibility properties the algorithm requires in a starting point.

Hillstrom (1977) was one of the first to consider testing optimization algorithms at nonstandard starting points. He proposed using random starting points chosen from a box surrounding the standard starting point. In another approach to this problem, in Moré et al. (1981) the authors present a large collection of test functions along with some procedures and starting points to assess the reliability and robustness of unconstrained optimization algorithms. In some cases, prior knowledge is available about the solution of a test problem. Some methods use such information to construct a starting point close to the optimal solution (Nocedal and Wright 2006).

Regardless of how starting points are selected, fair benchmarking requires all the algorithms to use the same starting point for each test. Therefore, starting points should be generated and stored outside of the testing process.

4.3 Quality of algorithmic output

The quality of the algorithmic output is obviously important when comparing optimization algorithms. Measuring quality falls into two easily separated categories: a known solution is available, and no known solutions are available.

4.3.1 Known solution available

When the expected solution for a problem is available, two methods can be employed to measure the quality of an algorithmic output: fixed-target and fixed-cost (Fowler et al. 2008; Rardin and Uzsoy 2001; Rios and Sahinidis 2013).

In the *fixed-target* method, the required time (function calls, iterations, etc) to find a solution at an accuracy target ε_{target} is evaluated. The main problem with fixed-target methods is that some algorithms may fail to solve a test problem. Therefore, the termination criterion cannot rely only on accuracy, but should also include some safety breaks such as the maximum computational budget. If the algorithm successfully reaches the desired accuracy, then the time to achieve the accuracy can be used to measure the quality of the algorithm on that test problem. If the algorithm terminates before reaching the desired accuracy, then it should be considered unsuccessful on that test problem.

Let x^0 be the initial point from a test run, $\bar{x} \in \mathbb{R}^n$ be the termination point obtained from the test run, and $x^* \in \mathbb{R}^n$ be the known solution for the problem. In the *fixed-cost* approach, the final optimization error $f(\bar{x}) - f(x^*)$ is checked after running the algorithm for a certain period of time, number of function calls, number of iterations, or some other fixed measurement of cost. Then, the smaller the final optimization error is, the better the quality of the algorithmic output.

The fixed-target versus fixed-cost decision can be seen as a multiobjective problem. It is analogous in engineering to minimizing cost, subject to constraints on performance versus maximizing performance, subject to a constraint on cost.

If a fixed-cost approach is used, then there are many ways to quantify the accuracy of the algorithmic output. We need to determine whether or not $\bar{x}$ approximates x^*. For example, this can be done using the function value or the distance from the solution:

$$f_{\mathtt{acc}} = f(\bar{x}) - f(x^*), \quad \text{and} \quad x_{\mathtt{acc}} = \|\bar{x} - x^*\| \text{ respectively.}$$

It is often valuable to "normalize" these quantities by dividing by the starting accuracy:

$$f^n_{\mathtt{acc}} = \frac{f(\bar{x}) - f(x^*)}{f(x^0) - f(x^*)}, \quad \text{and} \quad x^n_{\mathtt{acc}} = \frac{\|\bar{x} - x^*\|}{\|x^0 - x^*\|}.$$

Finally, to improve readability, and reduce floating point errors, many researchers take a base-10 logarithm:

$$f^l_{\mathtt{acc}} = \log_{10}(f(\bar{x}) - f(x^*)) - \log_{10}(f(x^0) - f(x^*)), \quad \text{and}$$
$$x^l_{\mathtt{acc}} = \log_{10}(\|\bar{x} - x^*\|) - \log_{10}(\|x^0 - x^*\|).$$

The values $f^l_{\mathtt{acc}}$ and $x^l_{\mathtt{acc}}$ can be loosely interpreted as the negative of the number of new digits of accuracy obtained (measured on a continuous scale), thus making these values very useful for discussion. Finally, to avoid exact solutions making an algorithm look better than it is, one can select a "maximal improvement value" M (typically about 16) and set

$$\gamma = \begin{cases} -f^l_{\mathtt{acc}}, & \text{if } -f^l_{\mathtt{acc}} \le M \\ M, & -f^l_{\mathtt{acc}} > M \text{ or } f(\bar{x}) - f(x^*) = 0, \end{cases} \tag{1}$$

or the analogous equation using $x^l_{\mathtt{acc}}$. Note that we have multiplied $f^n_{\mathtt{acc}}$ by -1, so γ can be interpreted as the number of new digits of accuracy obtained up to a maximal improvement value of M.

Similar measures can be used to quantify the amount of constraint violation for a test run. Considering $\min\{f(x) : g_i(x) \le 0, i = 1, 2, \ldots, m\}$,

$\sum_{i=1}^{m} \max\{0, g_i(\bar{x})\}$ gives the sum of violated constraints,

$\sum_{i=1}^{m} (\max\{0, g_i(\bar{x})\})^2$ gives the squared sum of violated constraints,

$\frac{1}{m} \sum_{i=1}^{m} \max\{0, g_i(\bar{x})\}$ gives the mean constraint violation, and

$\prod_{i:g_i(\bar{x}) > 0} g_i(\bar{x})$ amounts to the geometric mean of the violated constraints.

The selection of the appropriate strategy among the variety of approaches depends on the objectives of the experimental research, the problem structure, and the type of optimization algorithm used. The researcher should also carefully select the success criteria, e.g., how to fairly compare a solution that barely satisfies the constraints versus a solution that barely violates the constraints but returns a much lower objective function value.

4.3.2 No known solution available

In many situations, the test set used will not have known solutions to all problems. This is particularly true if the test set includes instances of real-world applications. To evaluate the quality of an algorithmic output in this situation, some new considerations are required (McGeoch 1996; Johnson et al. 1996).

First, it should be immediately obvious that, if no known solution is available, then fixed-target approaches cannot be applied. Fixed-cost approaches are still applicable, but since $f(x^*)$ is not known, measuring the accuracy of the final algorithmic output's function value, $f(\bar{x})$, becomes difficult. Measuring the accuracy of the final algorithmic output's point, $\bar{x}$, becomes essentially impossible.

To measure the quality of the final algorithmic output's function value $f(\bar{x})$, the simplest approach is to replace $f(x^*)$ with the best known value for the problem. For any given test run, this guarantees that at least one algorithm gets the exact answer, so it is important to select a reasonable maximal improvement value. Another approach is to estimate the optimal solution using statistical techniques. For example, in combinatorial optimization problems, some researchers (Dannenbring 1977; Derigs 1985) use a sample of algorithmic outputs to predict the location of the solution. In Golden and Stewart (1985), such an approach is explained in an evaluation of non-deterministic algorithms. Another strategy is to calculate a lower bound on the cost of an optimal solution, and to compare the algorithmic output cost with that lower bound. As an example, the total sum of the weight list in packing problems can be considered as a lower bound on the total number of bins used in a packing. Finally, one may abandon comparing the algorithmic output quality with the optimal solution, and assess only the quality of the algorithmic output with similar results published in the literature or other algorithms being tested.

4.4 Parameter tuning and stopping conditions

Additional parameters, such as stopping tolerances, population size, step sizes, or initial penalty parameters, are required for most optimization algorithms.

Among such parameters, stopping conditions play a highly notable role, as different stopping conditions can drastically change the output of an algorithm (Strongin and Sergeyev 2000; Sergeyev et al. 2013; Zhigljavsky and Žilinskas 2008). Moreover, if stopping tests are internalized within a method, it may not be possible to ensure all algorithms use the same stopping conditions (Strongin and Sergeyev 2000; Sergeyev and Kvasov 2006). However, if a fixed-cost or fixed-target approach (see Sect. 4.3) is employed, then other stopping conditions can be turned off, thereby ensuring all algorithms use the same stopping conditions. If it is not possible to ensure all algorithms use the same stopping conditions, then researchers should recognize this potential source of error when drawing conclusions from the results.

Other parameters, such as initial step length, can also have an impact on the performance of an optimization algorithm. Such parameters often require tuning in order to obtain better performance. If different choices of input parameters are allowed in an algorithm, researchers should mention the parameter settings used and

how they were selected. Different strategies used for tuning parameters affect the benchmarking process. Choosing appropriate parameter settings for an optimization algorithm is usually based on experiments and statistical analysis.

The tuning strategy should be chosen in conjunction with a specific algorithm and in a replicable manner (Johnson et al. 1996). Any improvements obtained from hand-tuning can of course be reported, but separately from more systematic comparative experiments. In some studies, algorithmic methods are presented to automate the tuning procedure of parameters (Audet and Orban 2006; Audet et al. 2014a; Baz et al. 2007; Hutter et al. 2009; Ridge 2007; Ridge and Kudenko 2010). The major disadvantage of these tuning methods is that they require a considerable computational investment because they usually try many possible settings to find an appropriate one. Nonetheless, in recent years some studies have specifically focused on the automatic tuning of parameters in optimization solvers. Examples of these efforts include the machine learning based method proposed in Baz et al. (2007), CPLEX automatic tuning tool (2014), use of derivative-free optimization (Audet and Orban 2006), ParamILS (Hutter et al. 2009), and the procedure proposed in Hutter et al. (2010) for mixed integer programming solvers. Similarly, some of the tuning techniques for non-deterministic methods include sequential parameter optimization (SPO) (Bartz-Beielstein et al. 2005; Bartz-Beielstein and Preuss 2014), relevance and calibration approach (Nannen and Eiben 2006), and F-Race (Birattari 2009).

In view of the considerable research on the automatic tuning of optimization solvers, a more accurate approach for benchmarking of optimization solvers requires a pre-processing step in which an automatic tuning method is employed to find suitable configuration settings for all the solvers (Hare and Wang 2010). As this is not always practical, it is important to emphasize that tuning parameters can have a major impact on the performance of an algorithm, therefore it is not appropriate to tune the parameters of some methods while leaving other methods at their default settings.

5 Analyzing and reporting the results

Many studies use basic statistics (e.g., average solving time) to report the experimental results. Basic statistics are a reasonable starting point, but provide little information about the overall performance of optimization methods. Reporting methods can be loosely broken down into three categories: numerical tables, graphics, and performance ratio methods (e.g., performance and data profiles).

5.1 Tables

Numerical tables provide the most complete method of reporting benchmarking results, so for the sake of completeness, we recommend making full tables of results readily available. However, such tables are cumbersome, so are often better included in an appendix or in additional online material linked to an article.

As full tables of results can easily overwhelm a reader, researchers have developed various techniques that provide easy-to-understand and compact methods

for reporting the experimental results. Summary tables can be employed as a first step (Sergeyev and Kvasov 2006). For example, in Billups et al. (1997) optimization methods were rated by the percentage of problems for which a method's time is termed *competitive* or *very competitive*. The solving time of an algorithm was called competitive if $T_s \leq 2T_{min}$ in which T_s is the solving time obtained by an algorithm on a particular problem and T_{min} is the minimum solving time obtained among all the algorithms on that specific problem. Similarly, if $T_s \leq \frac{4}{3}T_{min}$, then they call that method very competitive. Tables such as these provide good talking points for discussing benchmarking data, but fail to give a complete picture of the results. One criticism of this particular approach is it does not explore how much the table would change if, for example, the cut-off for very competitive was changed from $\frac{4}{3}T_{min}$ to $\frac{5}{4}T_{min}$.

Many other forms of summary tables are present throughout the optimization benchmarking literature, however all suffer from the same fundamental problem—to be readable, a summary table must distill the results down to a highly condensed format, thereby eliminating much of the benchmarking information.

5.2 Graphics

Well-conceived graphics can provide more information than some other data presentations. Simple graphical methods, such as histograms, box-plots, and trajectory plots, provide a next step in the analysis, while more complete methods include performance profiles, data profiles, and accuracy profiles. Depending on the objectives of an experimental research, one or more of these techniques might be useful to report the results. In Tukey (1977) and Tufte and Graves-Morris (1983), different types of plots are introduced, which are useful for data representation in general.

A more specialized plot for optimization algorithms is the trajectory plot (Fowler et al. 2008; Regis and Shoemaker 2007; Sandgren and Ragsdell 1980b; Eason 1982; Eason and Fenton 1974; Kortelainen et al. 2010; Strongin and Sergeyev 2000). In a trajectory plot, the performance of an optimization algorithm on a given test problem is visualized by plotting a path that connects the points generated by each iteration of the algorithm. An example appears in Fig. 2, where the trajectories of two algorithms attempting to minimize the Rosenbrock function are plotted. Both algorithms begin at the point (3, 3), and the first iteration moves both algorithms to the point (0.2, 3.5). Algorithm 1 (represented by the solid line) proceeds to (0.7, −0.2) and continues in a zig-zag path towards the minimizer. Algorithm 2 (represented by the dashed line) proceeds to (1.1, 1.3) and then follows a fairly straight path towards the minimizer, albeit with very small step sizes. While trajectory plots are useful to build a better understanding of how each algorithm behaves, they are not particularly good for benchmarking as they can only present the results for one test problem at a time. They are also limited to plots of functions of 2 or 3 variables, or to plotting projections onto subspaces for more than 3 variables.

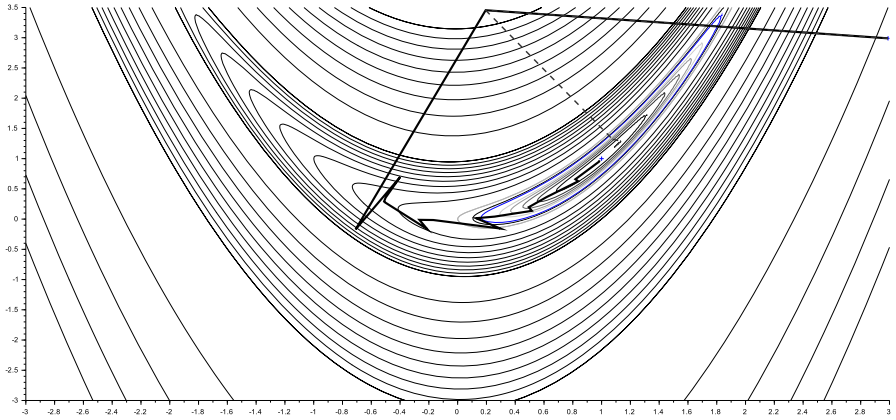

Fig. 2 A sample trajectory plot

Another specialized plot for optimization benchmarking is the *convergence plot*. In a convergence plot the performance of different optimization methods is visualized by plotting the best function value found against some measure of fundamental evaluation (Sect. 4.1). An example convergence plot is given in Fig. 3.

In Fig. 3 the results of four optimization methods are plotted for a given test problem. In this example, method M1 starts well, but stalls after about 300 function evaluations, while method M2 shows a steady decrease for about 800 function evaluations before stalling. Method M3 initially decreases the fastest, but stalls after about 350 function evaluations. Finally, method M4 starts very slowly, but ultimately finds the lowest value. Like trajectory plots, convergence plots are useful for discussing specific behavior of the algorithm, but are poor for benchmarking as they can only be used to analyze one test problem at a time.

While trajectory and convergence plots are useful to visualize a method on one problem, their main drawback is that they represent the results for a single problem

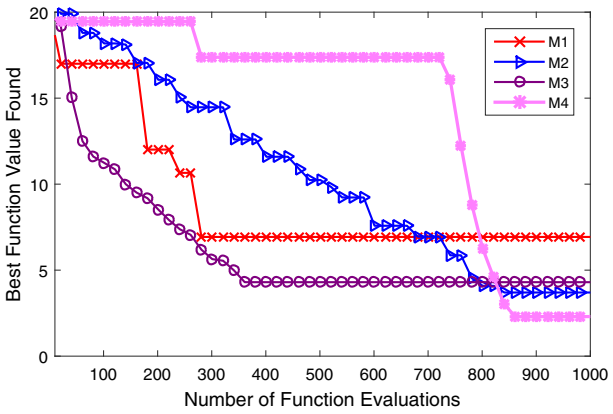

Fig. 3 A sample convergence plot

per plot. So if the test set contains a large number of problems then it will be difficult to evaluate the overall performance of these methods. Other types of plots can be found in the literature, but generally have the same limitations as trajectory and convergence plots (Benson et al. 2004; Regis and Shoemaker 2007).

For many optimization algorithms, researchers are interested in how the problem scales with the size of the input (e.g., dimension of the problem). For such research it can be valuable to produce a *runtime plot*. Runtime plots visualize the data by plotting the time to solve across a series of problem instances with different sizes. Runtime plots can suffer from similar issues to trajectory and convergence plots, namely, they represent the results for a single series of problem instances. However, this problem can be somewhat mitigated by aggregating data from a collection of problems to create an "average runtime" plot.

5.3 Performance profiles

According to Sergeyev et al. (2016), the idea of creating graphical comparisons of optimization methods dates back to at least the paper by Grishagin (1978).[3] In 2000, Strongin and Sergeyev presented the idea of *operational characteristics* for an algorithm: a graphical method to visualize the probability that an algorithm solves a problem within a set time-frame (Strongin and Sergeyev 2000). However, it was not until the paper by Dolan and Moré (2002) that the idea of graphically presenting benchmarking results became mainstream. Dolan and Moré (apparently unaware of the work of Grishagin or Strongin and Sergeyev) called their proposed graphs *performance profiles*.

Performance profiles provide interesting information such as efficiency, robustness, and probability of success in a graphically compact form (Dolan and Moré 2002). Their use has grown rapidly in optimization benchmarking, and should certainly be considered for any benchmarking optimization research.

Let $\mathcal{P}$ be a set of problems, $\mathcal{S}$ a set of optimization solvers, and $\mathcal{T}$ a convergence test. Assume proper data has been collected. The performance profiles are now defined in terms of a performance measure $t_{p,s} > 0$, obtained for each pair of $(p, s) \in P \times S$. This measure can be the computational time, the number of function evaluations, etc. A larger value of $t_{p,s}$ indicates worse performance. For each problem p and solver s, the performance ratio is defined as

$$r_{p,s} = \begin{cases} \dfrac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}} & \text{if convergence test passed,} \\ \infty & \text{if convergence test failed.} \end{cases} \tag{2}$$

for a specific problem p and test s (the best solver has $r_{p,s} = 1$). The *performance profile* of a solver s is defined as follows

[3] We thank "Mathematics Referee #1" for pointing out that reference.

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|} \text{ size } \{p \in \mathcal{P} : r_{p,s} \leq \tau\}, \tag{3}$$

where $|\mathcal{P}|$ represents the cardinality of the test set $\mathcal{P}$. Then, $\rho_s(\tau)$ is the portion of the time that the performance ratio $r_{p,s}$ for solver $s \in \mathcal{S}$ is within a factor $\tau \in \mathbb{R}$ of the best possible performance ratio.

Note that $\rho_s(1)$ represents the percentage of problems for which solver $s \in \mathcal{S}$ has the best performance among all the other solvers. And for τ sufficiently large, $\rho_s(\tau)$ is the percentage of the test set that can be solved by $s \in \mathcal{S}$. Solvers with consistently high values for $\rho_s(\tau)$ are of interest.

Figure 4 shows a sample performance profile plot [created using data from Beiranvand et al. (2015)] for logarithmic values of τ. The logarithmic values are employed to deal with smaller values for τ. This will result in a more accurate plot which shows the long-term behavior of the methods. To demonstrate the difference, Fig. 5 shows the same performance profile using non-logarithmic values of τ. Depending on the data collected, logarithmic or non-logarithmic values of τ may be more appropriate. Researchers should create both profiles, but it may only be necessary to provide one in the final manuscript.

The performance profile in Fig. 4 compares four different optimization methods on a test set of 60 problems. The method M1 has the best performance (in terms of CPU time) for almost 93% of the problems; meaning that M1 is able to solve 93% of the problems as fast or faster than the other two approaches. M3 solves roughly 11% of the problems as fast or faster than the other approaches. On the other hand, given enough time M1 solves about 92% of the problems, while M3 solves about 94% of the problems. The graphs of M1 and M3 cross at about $\log_2(\tau) \approx 3$ (i.e., $\tau \approx 8$); the two methods solve the same number of problems if time to solve is relaxed to be within a factor of 8.

Since performance profiles compare different methods versus the best method, the interpretation of the results should be limited to comparison to the best method

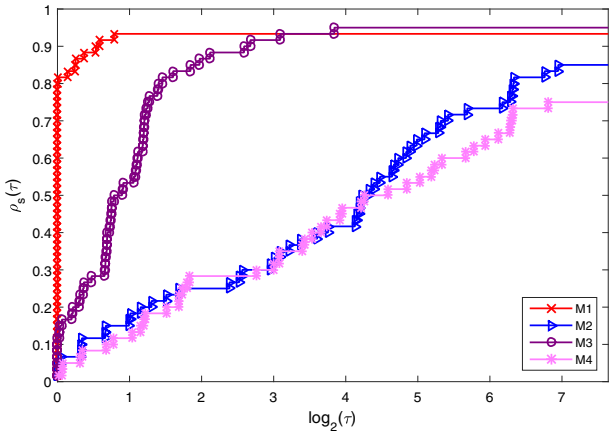

Fig. 4 An example performance profile using logarithmic values of τ

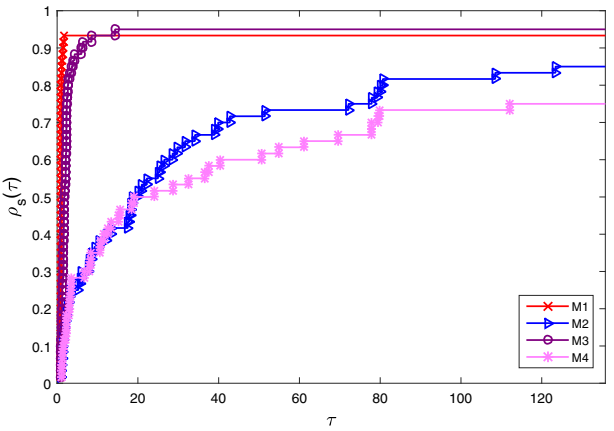

Fig. 5 The performance profile from Fig. 4 using non-logarithmic values of τ

and no interpretation should be made between, e.g., the second best and third best method since a switching phenomenon may occur (Gould and Scott 2016).[4] To compare the second and third best methods, a new performance profile should be drawn without the first method; see the explicit examples provided in Gould and Scott (2016).

Performance profile plots can be customized by substituting the standard performance measure *time*. For example, in Ali et al. (2005), Audet et al. (2010), and Vaz and Vicente (2007), the objective function value is used as the performance measure to compare the profiles. In particular, $t_{p,s}$ is replaced with

$$m_{p,s} = \frac{\hat{f}_{p,s}\ (\text{after } k \text{ function evaluations}) - f^*}{(f_w - f^*)}, \tag{4}$$

for problem p and solver s, where f_w is the largest (worst) function value obtained among all the algorithms, and $\hat{f}_{p,s}$ is the estimated function value after k function evaluations. In another example, Sergeyev and Kvasov (2015) create a performance measure based on proximity to optimal points.

The primary advantage of performance profiles is that they implicitly include both speed and success rate in the analysis. The value of $\rho_s(\alpha)$ gives a sense of how promising the algorithmic outputs are relative to the best solution found by all the optimization algorithms that are compared together.

One criticism of performance profiles is that the researcher must select a definition for the convergence test passing and failing. Changing this definition can substantially change the performance profile (Hare et al. 2011). Also note that if a fixed-cost approach is used to performing the benchmarking experiments, then performance profiles become inappropriate, as all the algorithms will use the same "time". Another criticism is that the profile is only showing performance with respect to the best method and does not allow one to compare other methods with

[4] We thank "Engineering Referee #3" for pointing out that reference.

each other due to the appearance of a switching phenomenon (Gould and Scott 2016).

Nonetheless, performance profiles have become a gold-standard in modern optimization benchmarking, and should be included in optimization benchmarking analysis whenever possible with an appropriate interpretation.

5.4 Accuracy profiles

Similar to performance profiles, *accuracy profiles* provide a visualization of an entire optimization benchmarking test set. However, accuracy profiles are designed for fixed-cost data sets. They begin by defining, for each problem $p \in \mathcal{P}$ and solver $s \in \mathcal{S}$, an accuracy measure (similar to Eq. (1)):

$$\gamma_{p,s} = \begin{cases} -f_{\text{acc}}^{p,s}, & \text{if } -f_{\text{acc}}^{p,s} \leq M \\ M, & -f_{\text{acc}}^{p,s} > M \text{ or } f_{\text{acc}}^{p,s} \text{ is undefined}, \end{cases}$$

where $f_{\text{acc}}^{p,s} = \log_{10}(f(\bar{x}_{p,s}) - f(x_p^*)) - \log_{10}(f(x_p^0) - f(x_p^*))$, $\bar{x}_{p,s}$ is the candidate solution point obtained by solver s on problem p, x_p^* is the optimal point for problem p, and x_p^0 is the initial point for problem p. The performance of the solver $s \in \mathcal{S}$ on the test set $\mathcal{P}$ is measured using the following function

$$R_s(\tau) = \frac{1}{|\mathcal{P}|} \text{ size } \{\gamma_{p,s} | \gamma_{p,s} \geq \tau, p \in \mathcal{P}\}.$$

The accuracy profile $R_s(\tau)$ shows the proportion of problems such that the solver $s \in \mathcal{S}$ is able to obtain a solution within an accuracy of τ of the best solution. An example accuracy profile (using data from Hare and Sagastizábal 2006) appears in Fig. 6.

In Fig. 6, we see four methods (M1, M2, M3, and M4) plotted against each other in an accuracy profile format. Examining the profile, notice that method M1 achieves 5 digits of accuracy on almost all test problems, and 6 digits of accuracy on about 75% of test problems. All other methods achieve this level of accuracy on

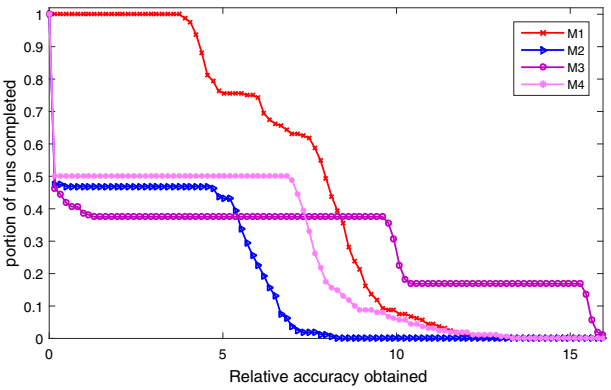

Fig. 6 An example accuracy profile

50% or less of test problems. Thus, if 5 or 6 digits is the desired level of accuracy, then M1 is a clear winner. However, if the particular application requires much higher accuracy, then M3 becomes a contender. Indeed, only M3 was able to achieve 12 digits of accuracy on any reasonable portion of the test problems. (In this particular test, accuracy was capped at 16 digits, but no method managed to achieve this on a significant portion of the test problems.)

Accuracy profiles do not provide as much information as performance profiles, but are suitable when fixed-cost data sets are collected. This is appropriate in cases where the cost of obtaining the exact solution exceeds the budget, so the optimization target is to find as good a solution as possible within a limited time.

5.5 Data profiles

Moré and Wild (2009) proposed data profiles as an adjustment to performance profiles for derivative-free optimization algorithms. Data profiles try to answer the question: what percentage of problems (for a given tolerance τ) can be solved within the budget of k function evaluations? They assume the required number of function evaluations to satisfy the convergence test is likely to grow as the number of variables increases. The data profile of an optimization algorithm s is defined using (Moré and Wild 2009)

$$d_s(k) = \frac{1}{|\mathcal{P}|} \text{ size } \left\{ p \in \mathcal{P} : \frac{t_{p,s}}{n_p + 1} \leq k \right\}, \tag{5}$$

in which $t_{p,s}$ shows the number of function evaluations required to satisfy the convergence test, n_p is the number of variables in the problem $p \in \mathcal{P}$, and $d_s(k)$ is the percentage of problems that can be solved with $k(n_p + 1)$ function evaluations. The value $k(n_p + 1)$ is used since $n_p + 1$ is the number of function evaluations required to compute a "simplex gradient" (a one-sided finite-difference estimate of the gradient).

It is worth noting that data profiles could easily be defined replacing $\frac{t_{p,s}}{n_p+1}$ by any other measure of fundamental evaluations used. Moreover, if $\frac{t_{p,s}}{n_p+1}$ is replaced by iterations, then data profiles become a slight variation of the *operational characteristics* defined in Strongin and Sergeyev (2000).

Figure 7 shows a typical data profile. Suppose the user has a budget limit of 100 simplex gradients; according to Fig. 7, with this budget method M4 has the best performance, solving roughly 22% of the problems; while M3 has the worst performance among all the solvers since with this budget it does not solve any problems.

Like performance profiles, data profiles are cumulative distribution functions, and thus, monotone increasing step functions with a range in [0, 1]. Data profiles do not provide the number of function evaluations required to solve a specific problem, but instead provide a visualization of the aggregate data. Also note that the data profile for a given solver $s \in S$ is independent of other solvers; this is not the case for performance profiles.

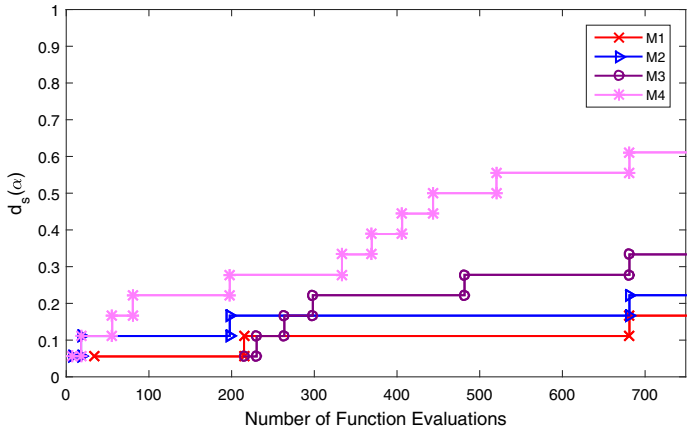

Fig. 7 An example data profile

Although the data profiles are useful for benchmarking, they have not received the same extensive attention as the performance profiles. This is partly because they are newer, but perhaps also because they are primarily used with derivative-free optimization algorithms. However, data profiles could be easily adjusted to a broader class of algorithms by replacing $t_{p,s}$ with any measure of time, and $n_p + 1$ by any dimensional normalization factor. For example, for a subgradient based method, $d_s(\alpha)$ could be redefined as

$$d_s(\alpha) = \frac{1}{|\mathcal{P}|} \text{ size } \{p \in \mathcal{P} : g_{p,s} \leq \alpha\},$$

where $g_{p,s}$ is the number of subgradient evaluations. This might make them an appropriate tool for benchmarking bundle methods (Hiriart-Urruty and Lemaréchal 1993, §XIV–XV).

Table 4 summarizes the reporting methods discussed in this section.

6 Automated benchmarking

As we have seen, the benchmarking process of optimization algorithms is a complicated task that requires much effort, from data preparation and transformation to the analysis and visualization of benchmarking data. Accordingly, some researchers have begun the development of software tools to facilitate and automate developing test sets, solving the problems using a variety of optimization algorithms, and carrying out performance analysis and visualization of benchmarking data.

The PAVER server (Mittelmann and Pruessner 2006; Bussieck et al. 2014) is an online server that provides some tools for the automated performance analysis, visualization, and processing of benchmarking data. An optimization engine, either a modeling environment such as AMPL (Fourer et al. 2002) or GAMS (Rosenthal

Table 4 Reporting methods summarization

Reporting method	Evaluates	Advantage	Drawback	Recommendation
Full data tables	–	Comprehensive	Overwhelming	Provide in appendix or online data set
Summary tables simple graphs	Varies	Brief	Incomplete	Provide as talking point, but include other forms of analysis
Trajectory plots convergence plots	Speed and accuracy efficiency	Clear precise	Examines one problem at a time	Good for case-studies, but should include other forms of analysis for larger data sets
Performance profiles–accuracy profiles–data profiles	Speed and robustness accuracy speed and robustness	Strong graphical representation that incorporates the entire dataset	Cannot be used for fixed-cost data sets	Include at least one of these three profiles whenever possible

2014), or a stand-alone solver, is required to obtain solution information such as the objective function value, resource time, number of iterations, and the solver status. Then, the benchmark data obtained by running several solvers over a set of problems can be automatically analyzed via online submission to the PAVER server. PAVER returns a performance analysis report through e-mail in HTML format. The tools available in PAVER allow either direct comparisons between two solvers or comparisons of more than two solvers simultaneously in terms of efficiency, robustness, algorithmic output quality, or performance profiles.

The High-performance Algorithm Laboratory (Nell et al. 2011) (HAL) is a computational environment designed to facilitate the empirical analysis and design of algorithms. It supports conducting large computational experiments and uses a database to handle data related to algorithms, test sets, and experimental results. It also supports distributed computation on a cluster of computers. Its major advantage over other tools is its aim to develop a general-purpose tool that can handle different categories of problems, although the initial deployment of problems and algorithms is tricky.

The Optimization Test Environment (Domes et al. 2014) is another tool that can be used for benchmarking the performance of different optimization algorithms. It provides some facilities to organize and solve large test sets, extract a specific subset of test sets using predefined measures, and perform statistical analysis on the benchmarking data. The results obtained by each optimization algorithm are verified in terms of feasibility and correctness. A variety of information is reported such as the number of global numerical solutions found (i.e., the best solution found among all optimization algorithms), number of local solutions found, number of wrong claims. For problem representation, it uses Directed Acyclic Graphs (DAGs) from the Coconut Environment (Schichl and Markót 2012). This user-friendly

environment analyzes results and automatically summarizes them before reporting them in an easy-to-use format such as LaTeX, JPEG, and PDF.

Other software tools for automating benchmarking process include EDACC (Balint et al. 2010), LIBOPT (Gilbert and Jonsson 2009), CUTEr (Gould et al. 2003) and a testing environment reported in Billups et al. (1997).

Using automated performance analysis tools has the potential to facilitate the benchmarking process. Moreover, the automation of the process may reduce the risk of biased comparison, by taking some of the comparison decisions away from the algorithm designer. However, automated benchmarking tools are not yet accepted by the research community due to their shortcomings. The major drawback of these tools is that the flexibility of a researcher to design experiments based on their research objectives is restricted to the tools' limitations and the way they view the benchmarking process. Moreover, so far all of these tools operate in expert mode, meaning that the usability aspect needs to be improved in terms of application and design of experiments. In most cases preparation of an experiment beyond the scope of default facilities of the benchmarking tools is nontrivial and involves some customization, e.g., scripting. Further research in this direction will create valuable tools for the optimization community, but the current status is not ready for widespread use.

7 Conclusion

This article reviews the issue of benchmarking optimization algorithms. For the sake of having a careful, less-biased, explicitly-stated, and comprehensive evaluation of the optimization algorithms an a priori benchmarking design is required. To develop an appropriate experimental design, the first task is to clarify the questions that are to be answered by the experiment. This includes selecting a suitable test set and suitable performance measures based on the objectives of the research. The data must be analyzed and processed in a transparent, fair, and complete manner. Within this paper we discuss each of these topics, and present a review of the state-of-the-art for each of these steps. We include several tables and figures that summarize the process, and provide key advice designed to lead to a fair benchmarking process.

A final important point must be raised in regards to optimization benchmarking:

> as in all scientific research, benchmarking optimization algorithms should be reported in a manner that allows for reproducibility of the experiments.

When reporting results, be sure to describe algorithms, parameters, test problems, the computational environment, and the statistical techniques employed with an acceptable level of detail. It should be clarified that it is usually difficult to provide enough information in a published paper to enable the reader to rerun the stated experiments and replicate completely the reported results. Moreover, the pace of computational development is so high that it is virtually impossible to entirely reproduce a computational experiment, due to development and modifications in operating systems, computer architecture, programming languages, etc. However,

the minimum standard for replication of the experiments is that at least the authors themselves should be able to replicate the experiments (Crowder et al. 1979). Therefore, it is important that the researcher keep all the programs and data necessary to redo all the computations and recreate all graphs. Such programs should be made available whenever possible.

7.1 Some final insights and remarks from the referees

This paper provides a high-level view of the benchmarking of optimization algorithms. While it does not aim to be all encompassing, it hopefully provides a baseline for best practice when benchmarking optimization algorithms. Many nuances exist when benchmarking specific genres of algorithms. We end with some final discussion of some of these nuanced areas. Many of these final remarks were provided through the insights of five excellent referees.

The state-of-the-art in optimization benchmarking currently has (at least) two major voids that require further research: how to properly benchmark optimization algorithms that make use of parallel processing, and how to properly benchmark multi-objective optimization algorithms.

Evaluating the performance of parallel optimization algorithms is different from traditional optimization methods in various aspects: performance measures such as time, the appropriate test sets, the new measures of merit involved in parallel processing such as the concept of speedup, efficiency. All of these concerns together with the fast pace of technological advances in parallel computing motivate research into the benchmarking of parallel optimization algorithms. A good start in this regard is the research paper by Barr and Hickman (1993).

Benchmarking multi-objective optimization algorithms is similarly in its infancy. Appropriate test sets and performance measures have yet to surface. Multi-objective optimization is a rapidly advancing field, and research into proper benchmarking in this discipline would be highly valuable.

A benchmarking challenge that we have not addressed is how to compare optimization algorithms that are different in nature.[5] For example, consider the comparison of a deterministic and a non-deterministic method (Gillard and Kvasov 2017; Kvasov and Mukhametzhanov 2017). If the multiple repeats of the non-deterministic method are considered, is it fair to compare the average quality to the single run of the deterministic method. Some ideas on this, including a proposed method for comparing deterministic and non-deterministic methods, can be found in Sergeyev et al. (2016).

Another benchmarking challenge that has not been fully addressed is how to compare algorithms that approach the same problem from fundamentally different viewpoints.[6] For example, when working with constrained optimization problems, some researchers have explored *infeasible point methods* while others have focused on *interior point methods*. Infeasible point methods typically take a two-phase approach, where one phase aims for a decrease in the function value and the second

[5] We thank "Mathematics Referee #1" for pointing out this challenge.

[6] We thank "Engineering Referee #3" and "Mathematics Referee #2" for pointing out this challenge.

phase aims to improve feasibility. Interior point methods assume a strictly feasible starting point and use some form of penalty function to maintain the feasibility of all trial points. Comparing these two styles of algorithms is very challenging, and possibly meaningless, as one assumes an infeasible starting point and the other assumes a feasible starting point. Other algorithms adopt a hybrid approach by approximating the feasible set with some tolerance (Regis and Wild 2017); in that case, the tolerance parameter could greatly influence the result of the comparison.

A source of debate in benchmarking global optimization algorithms is how to deal with *rescaling of the domain*.[7] Many global optimization algorithms are designed with the baseline assumption that the optimization problem's constrained region is the unit hypercube $[0, 1]^n$. Of course, in practical applications this is not always true. Some algorithms deal with this at the solver level, using the constraint set's diameter to select parameters like initial step lengths; while other algorithms deal with this at the problem level, assuming that the end-user will rescale the constraint set to be the unit hypercube (which is not always easy to do). Comparisons of algorithms that place fundamentally different assumptions on the problem structure may impact the selection of an appropriate test set and may limit the conclusions one can draw from the numerical results.

Another potential limitation on what conclusions can be drawn from a numerical study is the sensitivity analysis of the parameters.[8] A robust study should investigate a range of parameters and report on their impact on the validity of the conclusions. We leave the complexity of how best to report such information to future research.

References

Addis B, Locatelli M (2007) A new class of test functions for global optimization. J Glob Optim 38(3):479–501

Ali MM, Khompatraporn C, Zabinsky ZB (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J Glob Optim 31(4):635–672

Andrei N (2008) An unconstrained optimization test functions collection. Adv Model Optim 10(1):147–161

Asaadi J (1973) A computational comparison of some non-linear programs. Math Program 4(1):144–154

Audet C, Orban D (2006) Finding optimal algorithmic parameters using derivative-free optimization. SIAM J Optim 17(3):642–664

Audet C, Dang CK, Orban D (2010) Algorithmic parameter optimization of the DFO method with the OPAL framework. In: Ken N, Keita T, John C, Reiji S (eds) Software automatic tuning. Springer, New York, pp 255–274

Audet C, Dang K-C, Orban D (2014a) Optimization of algorithms with OPAL. Math Program Comput 6(3):233–254

Audet C, Le Digabel S, Peyrega M (2014b) Linear equalities in blackbox optimization. Technical report, Les Cahiers du GERAD

Averick BM, Carter RG, Moré JJ (1991) The MINPACK-2 test problem collection. Technical report, Argonne National Laboratory, Argonne

Balint A, Gall D, Kapler G, Retz R (2010) Experiment design and administration for computer clusters for SAT-solvers (EDACC), system description. J Satisf Boolean Model Comput 7:77–82

[7] We thank "Mathematics Referee #2" for pointing out this challenge.

[8] We thank "Mathematics Referee #1" for pointing out this challenge.

Bard Y (1970) Comparison of gradient methods for the solution of nonlinear parameter estimation problems. SIAM J Numer Anal 7(1):157–186
Barr RS, Hickman BL (1993) Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. INFORMS J Comput 5(1):2–18
Barr RS, Golden BL, Kelly JP, Resende MGC, Stewart WR Jr (1995) Designing and reporting on computational experiments with heuristic methods. J Heuristics 1(1):9–32
Barton RR (1987) Testing strategies for simulation optimization. In Proceedings of the 19th conference on winter simulation, WSC'87, New York, NY, USA. ACM, pp 391–401
Bartz-Beielstein T, Preuss M (2014) Experimental analysis of optimization algorithms: tuning and beyond. In: Borenstein Y, Moraglio A (eds) Theory and principled methods for the design of metaheuristics. Natural computing series. Springer, Berlin, pp 205–245
Bartz-Beielstein T, Lasarczyk CWG, Preuss M (2005) Sequential parameter optimization. In: The 2005 IEEE congress on evolutionary computation, vol 1, pp 773–780
Baz M, Hunsaker B, Brooks P, Gosavi A (2007) Automated tuning of optimization software parameters. Technical report, University of Pittsburgh, Department of Industrial Engineering
Beiranvand V, Hare W, Lucet Y, Hossain S (2015) Multi-haul quasi network flow model for vertical alignment optimization. Technical report, Computer Science, University of British Columbia, Kelowna, BC, Canada
Beltrami EJ (1969) A comparison of some recent iterative methods for the numerical solution of nonlinear programs. In: Beckmann M, Künzi HP (eds) Computing methods in optimization problems. Lecture notes in operations research and mathematical economics, vol 14. Springer, Berlin, pp 20–29
Benson HY, Shanno DF, Vanderbei RJ (2003) A comparative study of large-scale nonlinear optimization algorithms. In: Di Pillo G, Murli A (eds) High performance algorithms and software for nonlinear optimization. Applied optimization, vol 82. Springer, New York, pp 95–127
Benson HY, Shanno DF, Vanderbei RJ (2004) Interior-point methods for nonconvex nonlinear programming: jamming and numerical testing. Math Progr 99:35–48
Berthold T (2013) Measuring the impact of primal heuristics. Oper Res Lett 41(6):611–614
Billups SC, Dirkse SP, Ferris MC (1997) A comparison of large scale mixed complementarity problem solvers. Comput Optim Appl 7(1):3–25
Birattari M (2009) Tuning metaheuristics: a machine learning perspective. Springer, Berlin **(1st ed. 2005. 2nd printing edition)**
Bondarenko AS, Bortz DM, Moré JJ (1999) COPS: large-scale nonlinearly constrained optimization problems. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory. Technical report ANL/MCS-TM-237
Bongartz I, Conn AR, Gould N, Toint PL (1995) CUTE: constrained and unconstrained testing environment. ACM Trans Math Softw 21(1):123–160
Bongartz I, Conn AR, Gould NIM, Saunders MA, Toint PL (1997) A numerical comparison between the LANCELOT and MINOS packages for large scale constrained optimization. Technical report, SCAN-9711063
Box MJ (1966) A comparison of several current optimization methods, and the use of transformations in constrained problems. Comput J 9(1):67–77
Buckley AG (1992) Algorithm 709: testing algorithm implementations. ACM Trans Math Softw 18(4):375–391
Bussieck MR, Drud AS, Meeraus A, Pruessner A (2003) Quality assurance and global optimization. In: Bliek C, Jermann C, Neumaier A (eds) Global optimization and constraint satisfaction. Lecture notes in computer science, vol 2861. Springer, Berlin, pp 223–238
Bussieck MR, Dirkse SP, Vigerske S (2014) PAVER 2.0: an open source environment for automated performance analysis of benchmarking data. J Glob Optim 59(2–3):259–275
CPLEX's automatic tuning tool. Technical report, IBM Corporation, 2014
Colville AR (1968) A comparative study of nonlinear programming codes. Technical report 320-2949, IBM Scientific Center, New York
Conn AR, Gould N, Toint PL (1996) Numerical experiments with the LANCELOT package (release A) for large-scale nonlinear optimization. Math Program 73(1):73–110
Crowder H, Dembo RS, Mulvey JM (1979) On reporting computational experiments with mathematical software. ACM Trans Math Softw 5(2):193–203
Dannenbring DG (1977) Procedures for estimating optimal solution values for large combinatorial problems. Manag Sci 23(12):1273–1283

Dembo RS (1976) A set of geometric programming test problems and their solutions. Math Program 10(1):192–213
Dembo RS (1978) Current state of the art of algorithms and computer software for geometric programming. J Optim Theory Appl 26(2):149–183
Derigs U (1985) Using confidence limits for the global optimum in combinatorial optimization. Oper Res 33(5):1024–1049
Dixon LCW, Szegö GP (1978) Towards global optimisation 2. North-Holland, Amsterdam
Dolan ED, Moré JJ (2000) Benchmarking optimization software with COPS. Technical report, Argonne National Laboratory research report
Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91:201–213
Dolan ED, Moré JJ (2004) Benchmarking optimization software with COPS 3.0. Argonne National Laboratory research report
Domes F, Fuchs M, Schichl H, Neumaier A (2014) The optimization test environment. Optim Eng 15(2):443–468
Eason ED (1982) Evidence of fundamental difficulties in nonlinear optimization code comparisons. In: Mulvey JM (ed) Evaluating mathematical programming techniques. Lecture notes in economics and mathematical systems, vol 199. Springer, Berlin, pp 60–71
Eason ED, Fenton RG (1974) A comparison of numerical optimization methods for engineering design. J Manuf Sci Eng 96(1):196–200
Elam JJ, Klingman D (1982) NETGEN-II: a system for generating structured network-based mathematical programming test problems. In: Mulvey JM (ed) Evaluating mathematical programming techniques. Lecture notes in economics and mathematical systems, vol 199. Springer, Berlin, pp 16–23
Evtushenko YG (1985) Numerical optimization techniques. Translation series in mathematics and engineering. Optimization Software, Inc., Publications Division, New York (distributed by Springer, New York, Translated from the Russian, Translation edited and with a foreword by J. Stoer)
Famularo D, Pugliese P, Sergeyev YD (2002) Test problems for Lipschitz univariate global optimization with multiextremal constraints. In: Dzemyda G, Šaltenis V, Žilinskas A (eds) Stochastic and global optimization. Nonconvex optimization and its applications, vol 59. Kluwer Academic Publishers, Dordrecht, pp 93–109
Floudas CA, Pardalos PM (1990) A collection of test problems for constrained global optimization algorithms, vol 455. Springer, Berlin
Floudas CA, Pardalos PM, Adjiman CS, Esposito WR, Gëmës ZH, Harding ST, Klepeis JL, Meyer CA, Schweiger CA (1999) Handbook of test problems in local and global optimization. Springer, New York
Fourer R, Gay DM, Kernighan BW (2002) AMPL: a modeling language for mathematical programming, 2nd edn. Duxbury Press, Belmont
Fowler KR, Reese JP, Kees CE, Dennis JE Jr, Kelley CT, Miller CT, Audet C, Booker AJ, Couture G, Darwin RW, Farthing MW, Finkel DE, Gablonsky JM, Gray G, Kolda TG (2008) Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. Adv Water Resour 31(5):743–757
Gaviano M, Kvasov DE, Lera D, Sergeyev YD (2003) Algorithm 829: software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans Math Softw 29(4):469–480
Gilbert JC, Jonsson X (2009) LIBOPT an environment for testing solvers on heterogeneous collections of problems the manual, version 2.1. Technical report, INRIA, Le Chesnay
Gillard JW, Kvasov DE (2017) Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Stat Interface 10:59–70
Golden BL, Stewart WR (1985) Empirical analysis of heuristics. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, New York, pp 207–249
Gould N, Scott J (2016) A note on performance profiles for benchmarking software. ACM Trans Math Softw 43(2):15:1–15:5
Gould N, Orban D, Toint PL (2003) CUTEr and SifDec: a constrained and unconstrained testing environment, revisited. ACM Trans Math Softw 29(4):373–394
Gould NM, Orban D, Toint PL (2015) CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput Optim Appl 60(3):545–557

Grishagin VA (1978) Operating characteristics of some global search algorithms. In: Problems of stochastic search, vol 7. Zinatne, Riga, pp 198–206 **(in Russian)**
Grundel D, Jeffcoat D (2009) Combinatorial test problems and problem generators. In: Floudas CA, Pardalos PM (eds) Encyclopedia of optimization. Springer, New York, pp 391–394
Hansen P, Jaumard B, Lu SH (1992) Global optimization of univariate Lipschitz functions: II. New algorithms and computational comparison. Math Program 55(1–3):273–292
Hare W, Planiden C (2014) The NC-proximal average for multiple functions. Optim Lett 8(3):849–860
Hare W, Sagastizábal C (2006) Benchmark of some nonsmooth optimization solvers for computing nonconvex proximal points. Pac J Optim 2(3):545–573
Hare W, Sagastizábal C (2010) A redistributed proximal bundle method for nonconvex optimization. SIAM J Optim 20(5):2442–2473
Hare WL, Wang Y (2010) Fairer benchmarking of optimization algorithms via derivative free optimization. Technical report, optimization-online
Hare WL, Koch VR, Lucet Y (2011) Models and algorithms to improve earthwork operations in road design using mixed integer linear programming. Eur J Oper Res 215(2):470–480
Hillstrom KE (1977) A simulation test approach to the evaluation of nonlinear optimization algorithms. ACM Trans Math Softw 3(4):305–315
Hiriart-Urruty J-B, Lemaréchal C (1993) Convex analysis and minimization algorithms. II. Grundlehren der Mathematischen Wissenschaften [Fundamental principles of mathematical sciences]. Advanced theory and bundle methods, vol 306. Springer, Berlin
Hock W, Schittkowski K (1981) Test examples for nonlinear programming codes. Lecture notes in economics and mathematical systems. Springer, Berlin
Hock W, Schittkowski K (1983) A comparative performance evaluation of 27 nonlinear programming codes. Computing 30(4):335–358
Hoffman KL, Jackson RHF (1982) In pursuit of a methodology for testing mathematical programming software. In: Mulvey JM (ed) Evaluating mathematical programming techniques. Lecture notes in economics and mathematical systems, vol 199. Springer, Berlin, pp 177–199
Hoffman A, Mannos M, Sokolowsky D, Wiegmann N (1953) Computational experience in solving linear programs. J Soc Ind Appl Math 1(1):17–33
Hough P, Kolda T, Torczon V (2001) Asynchronous parallel pattern search for nonlinear optimization. SIAM J Sci Comput 23(1):134–156
Houstis EN, Rice JR, Christara CC, Vavalis EA (1988) Performance of scientific software. In: Rice JR (ed) Mathematical aspects of scientific software. The IMA volumes in mathematics and its applications, vol 14. Springer, New York, pp 123–155
Huang HY, Levy AV (1970) Numerical experiments on quadratically convergent algorithms for function minimization. J Optim Theory Appl 6(3):269–282
Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. J Artif Intell Res 36(1):267–306
Hutter F, Hoos HH, Leyton-Brown K (2010) Automated configuration of mixed integer programming solvers. In: Lodi A, Milano M, Toth P (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 6140. Springer, Berlin, pp 186–202
Jackson RHF, Boggs PT, Nash SG, Powell S (1990) Guidelines for reporting results of computational experiments. Report of the ad hoc committee. Math Program 49(1–3):413–425
Jamil M, Yang XS (2013) A literature survey of benchmark functions for global optimisation problems. Int J Math Model Numer Optim 4(2):150–194
Johnson DS, McGeoch LA, Rothberg EE (1996) Asymptotic experimental analysis for the Held–Karp traveling salesman bound. In: Proceedings of the seventh annual ACM-SIAM symposium on discrete algorithms, SODA'96, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp 341–350
Khompatraporn C, Pinter JD, Zabinsky ZB (2005) Comparative assessment of algorithms and software for global optimization. J Glob Optim 31(4):613–633
Knuth DE (1994) The Stanford GraphBase: a platform for combinatorial computing, vol 37. Addison-Wesley Publishing Company, Boston
Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011) MIPLIB 2010. Math Program Comput 3(2):103–163

Kolda TG, Lewis RM, Torczon V (2003) Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev 45(3):385–482
Kortelainen M, Lesinski T, Moré J, Nazarewicz W, Sarich J, Schunck N, Stoitsov MV, Wild S (2010) Nuclear energy density optimization. Phys Rev C 82(2):024313
Kvasov DE, Mukhametzhanov MS (2016) One-dimensional global search: nature-inspired vs. lipschitz methods. AIP Conf Proc 1738(1):400012
Kvasov DE, Mukhametzhanov MS (2017) Metaheuristic vs. deterministic global optimization algorithms: the univariate case. Appl Math Comput
Kvasov DE, Sergeyev YD (2015) Deterministic approaches for solving practical black-box global optimization problems. Adv Eng Softw 80:58–66
LaMarca A, Ladner R (1996) The influence of caches on the performance of heaps. J Exp Algorithmics 1:4
Lenard ML, Minkoff M (1984) Randomly generated test problems for positive definite quadratic programming. ACM Trans Math Softw 10(1):86–96
Liu D, Zhang XS (2000) Test problem generator by neural network for algorithms that try solving nonlinear programming problems globally. J Glob Optim 16(3):229–243
McGeoch CC (1996) Toward an experimental method for algorithm simulation. INFORMS J Comput 8(1):1–15
McGeoch CC (2002) Experimental analysis of algorithms. In: Pardalos PM, Romeijn HE (eds) Handbook of global optimization. Nonconvex optimization and its applications, vol 62. Springer, New York, pp 489–513
Miele A, Tietze JL, Levy AV (1972) Comparison of several gradient algorithms for mathematical programming problems. Aero-astronautics report no. 94, Rice University, Houston
Mittelmann HD (2003) An independent benchmarking of SDP and SOCP solvers. Math Program 95(2):407–430
Mittelmann HD, Pruessner A (2006) A server for automated performance analysis of benchmarking data. Optim Methods Softw 21(1):105–120
Moré JJ, Wild S (2009) Benchmarking derivative-free optimization algorithms. SIAM J Optim 20(1):172–191
Moré JJ, Garbow BS, Hillstrom KE (1981) Testing unconstrained optimization software. ACM Trans Math Softw 7(1):17–41
Mulvey JM (ed) (1982) Evaluating mathematical programming techniques, vol 199. Springer, Berlin
Nannen V, Eiben AE (2006) A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Proceedings of the 8th annual conference on genetic and evolutionary computation, GECCO'06, New York, NY, USA. ACM, pp 183–190
Nash S, Nocedal J (1991) A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. SIAM J Optim 1(3):358–372
Nell C, Fawcett C, Hoos HH, Leyton-Brown K (2011) HAL: a framework for the automated analysis and design of high-performance algorithms. In: Coello CAC (ed) Learning and intelligent optimization. Lecture notes in computer science, vol 6683. Springer, Berlin, pp 600–615
Netlib: Netlib linear programming library. http://netlib.org/
Neumaier A, Shcherbina O, Huyer W, Vinkó T (2005) A comparison of complete global optimization solvers. Math Program 103(2):335–356
Ng C-K, Li D (2014) Test problem generator for unconstrained global optimization. Comput Oper Res 51:338–349
Nocedal J, Wright S (2006) Numerical optimization. Springer series in operations research and financial engineering. Springer, New York
Opara K, Arabas J (2011) Benchmarking procedures for continuous optimization algorithms. J Telecommun Inf Technol 4:73–80
Parejo JA, Ruiz-Cortés A, Lozano S, Fernandez P (2012) Metaheuristic optimization frameworks: a survey and benchmarking. Soft Comput 16(3):527–561
Paulavičius R, Sergeyev YD, Kvasov DE, Žilinskas J (2014) Globally-biased Disimpl algorithm for expensive global optimization. J Glob Optim 59(2–3):545–567
Pintér JD (2002) Global optimization: software, test problems, and applications. In: Pardalos PM, Romeijn HE (eds) Handbook of global optimization. Nonconvex optimization and its applications, vol 62. Springer, New York, pp 515–569
Pintér JD (2007) Nonlinear optimization with GAMS /LGO. J Glob Optim 38(1):79–101

Pintér JD, Kampas FJ (2013) Benchmarking nonlinear optimization software in technical computing environments. Top 21(1):133–162
Ramsin H, Wedin P (1977) A comparison of some algorithms for the nonlinear least squares problem. BIT Numer Math 17(1):72–90
Rardin RL, Uzsoy R (2001) Experimental evaluation of heuristic optimization algorithms: a tutorial. J Heuristics 7(3):261–304
Regis RG, Shoemaker CA (2007) A stochastic radial basis function method for the global optimization of expensive functions. INFORMS J Comput 19(4):497–509
Regis RG, Wild SM (2017) Conorbit: constrained optimization by radial basis function interpolation in trust regions. Optim Methods Softw 32(3):1–29
Ridge E (2007) Design of experiments for the tuning of optimisation algorithms. Department of Computer Science, University of York, Heslington
Ridge E, Kudenko D (2010) Tuning an algorithm using design of experiments. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Experimental methods for the analysis of optimization algorithms. Springer, Berlin, pp 265–286
Rijckaert MJ, Martens XM (1978) Comparison of generalized geometric programming algorithms. J Optim Theory Appl 26(2):205–242
Rios L, Sahinidis NV (2013) Derivative-free optimization: a review of algorithms and comparison of software implementations. J Glob Optim 56(3):1247–1293
Romesis M, Xie M, Minkovich K, Cong J (2003) Optimality study project. Technical report, UCLA Computer Science Department. http://cadlab.cs.ucla.edu/~pubbench/
Rosenthal RE (2014) GAMS—a user's guide. Technical report, GAMS Development Corporation
Sandgren E, Ragsdell KM (1980a) The utility of nonlinear programming algorithms: a comparative study, part 1. J Mech Des 102(3):540–546
Sandgren E, Ragsdell KM (1980b) The utility of nonlinear programming algorithms: a comparative study, part 2. J Mech Des 102(3):547–551
Schichl H, Markót MC (2012) Algorithmic differentiation techniques for global optimization in the COCONUT environment. Optim Methods Softw 27(2):359–372
Schittkowski K (1980) Nonlinear programming codes: information, tests, performance. Lecture notes in economics and mathematical systems. Springer, Berlin
Schittkowski K (2008) An updated set of 306 test problems for nonlinear programming with validated optimal solutions—user's guide. Technical report, Department of Computer Science, University of Bayreuth
Schittkowski K, Stoer J (1978) A factorization method for the solution of constrained linear least squares problems allowing subsequent data changes. Numer Math 31(4):431–463
Schoen F (1993) A wide class of test functions for global optimization. J Glob Optim 3(2):133–137
Sergeyev YD, Kvasov DE (2006) Global search based on efficient diagonal partitions and a set of Lipschitz constants. SIAM J Optim 16(3):910–937
Sergeyev YD, Kvasov DE (2015) A deterministic global optimization using smooth diagonal auxiliary functions. Commun Nonlinear Sci Numer Simul 21(1–3):99–111
Sergeyev YD, Strongin RG, Lera D (2013) Introduction to global optimization exploiting space-filling curves. Springer briefs in optimization. Springer, New York
Sergeyev YD, Kvasov DE, Mukhametzhanov MS (2016) Operational zones for comparing metaheuristic and deterministic one-dimensional global optimization algorithms. Math Comput Simul 141:96–109
Shcherbina O, Neumaier A, Sam-Haroud D, Vu XH, Nguyen TV (2003) Benchmarking global optimization and constraint satisfaction codes. In: Bliek C, Jermann C, Neumaier A (eds) Global optimization and constraint satisfaction. Lecture notes in computer science, vol 6683. Springer, Berlin, pp 211–222
Strongin RG, Sergeyev YD (2000) Global optimization with non-convex constraints: sequential and parallel algorithms. Springer, New York
Tabak D (1969) Comparative study of various minimization techniques used in mathematical programming. IEEE Trans Autom Control 14(5):572–572
Tedford NP, Martins JRRA (2010) Benchmarking multidisciplinary design optimization algorithms. Optim Eng 11(1):159–183
Törn A, Žilinskas A (1989) Global optimization. Lecture notes in computer science, vol 350. Springer, Berlin
Törn A, Ali MM, Viitanen S (1999) Stochastic global optimization: problem classes and solution techniques. J Glob Optim 14(4):437–447

Tufte ER, Graves-Morris PR (1983) The visual display of quantitative information, vol 2. Graphics Press, Cheshire

Tukey JW (1977) Exploratory data analysis. Pearson, Reading

Vanden Berghen F, Bersini H (2005) CONDOR, a new parallel, constrained extension of powell's UOBYQA algorithm: experimental results and comparison with the DFO algorithm. J Comput Appl Math 181(1):157–175

Vanderbei RJ, Shanno DF (1999) An interior-point algorithm for nonconvex nonlinear programming. Comput Optim Appl 13(1–3):231–252

Vaz AIF, Vicente LN (2007) A particle swarm pattern search method for bound constrained global optimization. J Glob Optim 39(2):197–219

Yeniay O (2005) A comparative study on optimization methods for the constrained nonlinear programming problems. Math Probl Eng 2005(2):165–173

Zhang Z (2014) Sobolev seminorm of quadratic functions with applications to derivative-free optimization. Math Program 146(1–2):77–96

Zhigljavsky A, Žilinskas A (2008) Stochastic global optimization. Springer optimization and its applications, vol 9. Springer, New York