Anytime Performance Assessment in Blackbox Optimization Benchmarking

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Tea Tusar

Anytime Performance Assessment in Blackbox Optimization Benchmarking

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tušar

***Abstract*—We present concepts and recipes for the anytime performance assessment when benchmarking optimization algorithms in a blackbox scenario. We consider runtime—oftentimes measured in number of blackbox evaluations needed to reach a target quality—to be a universally measurable cost for solving a problem. Starting from the graph that depicts the solution quality versus runtime, we argue that runtime is the *only* performance measure with a generic, meaningful, and quantitative interpretation. Hence, our assessment is solely based on runtime measurements. We discuss proper choices for solution quality indicators in single- and multiobjective optimization, as well as in the presence of noise and constraints. We also discuss the choice of the target values, budget-based targets, and the aggregation of runtimes by using simulated restarts, averages, and empirical cumulative distributions which generalize convergence graphs of single runs. The presented performance assessment is to a large extent implemented in the *comparing continuous optimizers* (COCO) platform freely available at https://github.com/numbbo/coco.**

***Index Terms*—anytime optimization, performance assessment, benchmarking, blackbox optimization, quality indicator**

I. Introduction

We present practical concepts and ideas for the performance assessment of optimization algorithms when benchmarked in a blackbox and anytime scenario. Going beyond a simple ranking of algorithms, we aim to provide a *quantitative* and *meaningful* performance assessment, which allows for conclusions like *algorithm A is seven times faster than algorithm B* in solving a given problem or in solving problems with certain characteristics. To achieve this end in a comparative and timeless manner, we argue that we should measure the *number of blackbox evaluations* to reach a predefined *quality indicator* value (a target). More generally, we argue to measure a cost that is defined on a ratio scale and is comparable across publications. We call this measure the *runtime* of the algorithm to reach a given target. Yet, our assessment methodology does not depend on any specific cost measure, as long as the costs are quantitative and comparable.[1]

In this paper, we formalize the optimization goal by a so-called quality indicator. Its definition may heavily depend on the optimization scenario, e.g. the number of objectives or constraints. Broadly speaking, a quality indicator is based on the sequence of all so-far visited solutions. In the simplest case, it is the objective function value of the last visited solution.

Runtimes represent the cost of optimization. Compared to the quality indicator, the definition of costs depends to a lesser extent on the specific optimization scenario. To sustain reproducibility and comparability across publications, we recommend against CPU or wall-clock time as cost measure[2] (see also Hooker [22] for a further discussion on the unwanted consequences of benchmarking based on CPU time).

Benchmarking is usually computationally expensive and benchmarking for a *single* budget seems vastly inefficient by i) addressing only one of many possible budget scenarios (scenarios heavily depend on the software and hardware environment) and ii) throwing away most of the data generated during the experiment. An anytime approach to benchmarking prevents these drawbacks. To allow for a budget-free performance assessment even for nonanytime algorithms that have a maximum or timeout budget as decisive or mandatory *input* parameter (decided by the user), we collect data with an any-budget *experimental procedure* that runs repeated experiments with increasing input budget [31].[3] Nonanytime algorithms that do not take a maximum budget as input parameter can be accurately assessed only by the time of their final solution proposal.

In this article, we advocate to routinely use (anytime) *empirical runtime distributions* to assess the performance of optimization algorithms. We demonstrate how to directly compare the runtime distributions of algorithms that have

Nikolaus Hansen, Anne Auger and Dimo Brockhoff are with Inria and Ecole Polytechnique, Institut Polytechnique de Paris, France.

Tea Tušar is with the Jožef Stefan Institute, Ljubljana, Slovenia.

Manuscript received 18 September 2021; revised 16 February 2022 and 16 June 2022; accepted 14 September 2022. This work was supported in part by the French National Research Agency (NumBBO) under Grant ANR-12-MONU-0009, and in part by the Slovenian Research Agency under the Grants P2-0209 and N2-0254.

[1] We are grateful to the anonymous reviewer pointing this out to us.

[2] An exploratory CPU timing experiment to get an estimate of the internal time complexity of the algorithm is still advisable, like it is prescribed in the COCO platform [18].

[3] We can stop the procedure when the last budget was not fully exhausted. Increasing the budget each time by a factor of $r > 1$ adds to the overall computational costs for the experimentation less than $\sum_{i=0}^{\infty} 1/r^i = r/(r-1)$ times the last consumed budget. For the performance assessment, always the data from the smallest eligible budget is used. The performance assessment will be too optimistic by tacitly assuming that the budget can be set properly without additional costs. On the other hand, runtimes may be overestimated (by less than a factor of r). A code example is provided in `example_experiment_non_anytime.py`.

vastly different success rates via *simulated restarts* and bootstrapping.

The performance assessment aspects discussed in this paper are rather generic and, for example, independent of the search domain. Most of them are implemented in the benchmarking platform COCO [20] for continuous and mixed-integer optimization and have stood the test of time in our benchmarking practice. Most elements presented here were touched upon already in [16] where a general benchmarking framework is presented. The purpose of this paper is to lay out specifics and caveats of the performance assessment aspect. We also introduce a new quality indicator for noisy optimization and graphically present the direct link between the lower envelope of convergence graphs and empirical runtime distributions.

The paper is structured as follows: Section II introduces necessary terminology and basic definitions. Section III discusses performance measures in general and specific aspects like quality indicators, fixed-budget versus fixed-target measurements, missing values and setting up targets. Section IV formalizes the runtime measurement and touches upon alternative costs measures and different instances. Section V discusses aggregating measures like expected runtimes and simulated restarts via bootstrapping. Section VI introduces empirical cumulative distribution functions of recorded or simulated runtimes and shows examples of these *empirical runtime distributions* (RTD). Section VII discusses relation to previous works and Section VIII summarizes main aspects and concludes the paper.

II. Terminology and Definitions

We consider a set of parametrized benchmark functions $f_\theta : S \to \mathbb{R}^m$ to be minimized, where $S = S_1 \times \cdots \times S_n$ and each S_i is a set of numbers, for example $\{0, 1\}$ or $\mathbb{Z}$ or $\mathbb{R}$, and $\theta \in \Theta$ parametrizes a function instance. This formulation can also accommodate constraints as components of the vector-valued f_θ which are strictly positive for infeasible solutions, nonpositive for feasible solutions, and only to be minimized in the infeasible domain (down to zero).

We define a specific *problem* in the form of an instance quadruple, $p^{(4)} = (n, m, f_\theta, \theta)$, by the search space dimension n, the number of objective (and constraint) functions m, the objective (and constraint) function f, and its instance parameters θ [16].[4]

The instance notion is introduced to allow for simple repetitions when benchmarking deterministic algorithms. Instances also prevent that any lucky or unlucky *coincidental* match between function and algorithm can dominate the outcome. Instance generation may remove artificial function properties like the location of the optimum in all-zero or separability. Different instances, θ, have different locations of the optimal solution, may have different rotations applied to continuous variables, have different optimal f-values, etc. [17]. The separation of dimension and instance parameters in the notation is convenient because we never aggregate over dimension but usually aggregate over all θ-values for which experimental results are available, see also Section V.

In the performance assessment setting, we associate to a problem instance $p^{(4)}$ a quality indicator mapping (see Section III-A) and a target value τ which depends on a target precision ε (see Section III-F). Then, a problem is expressed as a quintuple $p^{(5)} = (n, m, f_\theta, \theta, \tau(\varepsilon))$. The quality indicator usually remains the same for all problems of a problem suite (and is thus omitted from the quintuple), while we have subsets of problems which only differ in their target value.

III. On Performance Measures

Assessing performance is necessarily based on performance *measures*, the definition of which plays a crucial role for the evaluation. Here, we introduce a list of desired properties for a performance measure in general, as well as in the context of blackbox optimization specifically. Ideally, a performance measure should be

1) quantitative, as opposed to a simple *ranking* of entrants (e.g., algorithms); ideally, the measure should be defined on a ratio scale[5] (as opposed to an interval or ordinal scale) [29], which allows to state that "entrant A is a factor α *times* better than entrant B" for some $\alpha > 0$,
2) assuming a wide variation of values in that typical values do not only range between, say, 0.98 and 1.0,[6]
3) well interpretable, in particular by having meaning and semantics for the measured numbers,
4) relevant and meaningful with respect to the "real world",[7]
5) available for most targeted algorithms and problems,[8]
6) readily reproducible independently of specific experimental settings, for example the machine on which experiments have been run,
7) comparable across different problems,
8) as simple and comprehensible as possible.

As we will see in the following subsections, measurements derived from the *runtime* to reach a target value, measured in number of evaluations of candidate solutions, appear to be the only obvious candidate measurements with all the desired properties in the context of blackbox optimization. Runtime is well interpretable and meaningful with respect to the real world, representing the time needed to solve a problem. Measuring the number of evaluations avoids the shortcomings of CPU or wall-clock time measurements which

[4] An optimization algorithm has access to the value of n, the number of objectives and constraints, the definition of S, regions of interest in S and in the image of f_θ, a feasible initial solution in S, and it can acquire return values of f_θ for any candidate solution in S, for example by calling a coded function. Because the algorithm is not allowed to use any other information, in particular on the character or inner workings of f_θ, the scenario is called blackbox optimization.

[5] A variable that is defined on a ratio scale has a meaningful zero, allows for division and hence for the statement "A is α *times* better" based on $\mathrm{B}/\mathrm{A} = \alpha$, and can be taken to the logarithm in a meaningful way [29].

[6] A transformation like $x \mapsto \ln(1-x)$ or, similarly, the logit transformation $x \mapsto \ln(x/(1-x))$ could alleviate the problem in this case, given that zooming in on values close to 1 is relevant and meaningful.

[7] As a counterexample, minimizing the third decimal digit of the quality indicator would be a well defined and achievable goal but neither relevant nor meaningful in practice, unless also the preceding digits are minimal.

[8] For example, minimizing the norm of the gradient is meaningful and relevant on unimodal differentiable functions, but the gradient is often not available in the blackbox optimization setting.

depend on the programming language, coding style, machine used to run the experiment, etc.—parameters that are difficult, impractical or even impossible to control.[2]

Additionally, even if algorithm-internal computations dominate the wall-clock time in a practical application, comparative runtime results *in number of evaluations* can usually still be converted *a posteriori* to reflect the practical scenario. This also holds true for a speed-up from parallelization.

A. Quality Indicators

Let us assume that an algorithm optimizes a problem instance, $p^{(4)}$, in a blackbox setting. After each evaluation of the problem, that is, after each time step $t = 1, 2, \ldots$ the algorithm obtains new information on the problem. Using this information, the algorithm can generate a new proposal solution, x_t, to solve the problem at time step t. For example, the proposal can be the current best estimate of the optimum, or an element of an estimated Pareto set or the most recently evaluated solution (by default). In the remainder, we assume this sequence of proposal solutions is given. We assess the solution quality of the sequence with a quality indicator and define various indicators for different optimization scenarios.

A quality indicator I maps at each time step t the sequence $(x_1, \ldots, x_t)$ of proposal solutions to a real value, the quality indicator value. We provide a few examples.

For *single-objective unconstrained noiseless optimization*, the quality indicator usually outputs the function value of the last solution in the sequence. Taking the best solution is equivalent in the following, because both indicator value sequences trigger the same *first* hitting time.

For *single-objective noisy optimization*, a *deterministic* quality indicator must operate on a noiseless value, for example the 50%-ile of the true f-value distribution for any given solution.[9] Based on this deterministic f-value, we propose as quality indicator at time step t the 1%-ile worst of the last $\lceil \ln^2(t+3)/2 \rceil$ values in the sequence.[10] Taking the worst from an increasing number of the most recent solutions makes it virtually impossible to obtain a good indicator value just by chance.[11] The formula is crafted to give 100%, 40%, 11%, and 2% from $1, 10, 100$, and 1000 values, respectively, and 1061 values of 10^{20}. This appears to be both, large enough to make exploitation very difficult and small enough to represent a reasonably recent algorithm state and be easily manageable. The 1%-ile worst is taken (instead of the worst) to yield a measure that is independent of data size.

For *constrained optimization*, the quality indicator can be the value of the last *feasible* solution. This is however not a meaningful indicator for algorithms that converge from the infeasible domain towards the feasible optimum. Hence, we propose a somewhat arbitrary quality indicator mapping of the sequence of solutions $(x_1, \ldots, x_t)$ to the value $|f(x_t) - f_{\text{opt}}|^+ + \sum_i \lambda_i |g_i(x_t)|^+$ for constraints $g_i(x) : S \to \mathbb{R}$ which are nonpositive in the feasible domain, where f_{opt} is the f-value of the feasible optimum, $|.|^+$ clips the argument to positive values and λ_i are positive weights for the constraints.[12]

For *multiobjective optimization*, we propose a quality indicator based on the hypervolume indicator [9], given in detail in the Appendix.

B. Alternative Cost Measures

Generally, we proposed to count (blackbox) function evaluations to measure cost or runtime. Without losing desired properties of the performance assessment, we can also count *constraints* evaluations or any combination of function and constraints evaluations (as implemented in COCO). Even further, instead of incrementing the count by one, the increment can be the actually measured and recorded cost of each evaluation such that the cumulative cost value becomes a positive real number. This can account for varying costs, for example, due to partial evaluations or due to correlations between costs and location of the evaluated solution or between costs and number of already evaluated solutions. Replacing the count with any cumulative cost value does not change the assessment methodology.[1] The caveat is to ensure that the computed costs remain generically comparable between different functions (to allow for aggregation) and different algorithms ideally also across publications.

C. Fixed-Budget versus Fixed-Target Approach

For a given quality indicator, the basic but comprehensive performance display plots the indicator value against the number of evaluations in a single graph, that is, "quality" against costs, see Figure 1. There are essentially only two ways to measure performance now (and aggregates thereof, representing certain areas under the graph).

We can measure the reached quality indicator values after exhausting a **fixed budget** of evaluations. The fixed budget can be illustrated as a *vertical* line in Figure 1. Alternatively, we can measure the number of evaluations, the *runtime*, to reach a **fixed target** quality indicator value. The fixed target can be illustrated as a *horizontal* line in Figure 1. In both cases, we

[9]The ordering between indicator values is independent of the chosen percentile if and only if for every pair of f-value distributions either one stochastically dominates the other or they are the same, as it is implemented in the `bbob-noisy` test suite of COCO.

[10]As discussed at https://github.com/numbbo/coco/issues/168 but not yet implemented in COCO.

[11]In the noisy scenario, a trustworthy algorithm cannot rely on a single f-measurement to pick its return value. The probability to obtain a good solution by pure chance also quickly decreases with increasing dimension and increasing target difficulty roughly like $1/\sqrt{\varepsilon^n}$ such that hitting a target as given in Eq. (1) indicates that the algorithm must be able to locate solutions of according quality *by design*.

[12]The indicator remains applicable even if the value of f_{opt} is unknown and approximated only after obtaining the experimental data. Generally, it seems preferable to put the weights λ_i in the definition of g_i such that they are also exposed to the algorithm. Setting the weights a posteriori seems to allow for a somewhat arbitrary and undesirable tuning of the performance assessment. Apart from setting $\lambda_i = 1$, that is, using the constraints as presented to the algorithm, we see arguments for two further weight settings to be meaningful. 1) Setting $\lambda_i = \infty$, that is, disallowing infeasible solutions as valuable all together. Consequently, any value between 1 and ∞ could be justified as putting an increasing weight on feasibility in comparison to f. 2) If the g_i are differentiable, we may set λ_i proportional to the Lagrange multipliers at the global optimum of the constrained problem (as suggested to us by Paul Dufossé in personal communications). These multipliers are unique when the gradients of the active constraints in the optimum are linearly independent [26, p. 321] and can be approximated by an optimization algorithm [1], [2]. In COCO, we use $\lambda_i = 1$ for all constraints i.

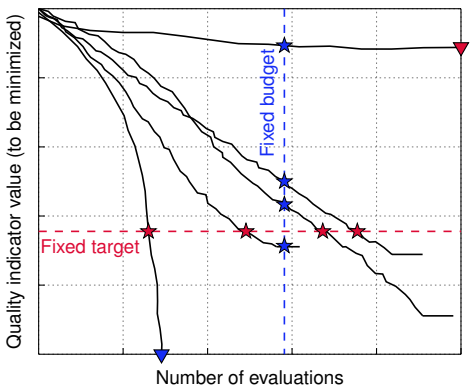

Fig. 1. Illustration of the fixed-budget (dashed vertical line) and the fixed-target (dashed horizontal line) approach to measure performance. Black lines depict the best quality indicator value plotted versus number of blackbox evaluations. Stars depict the measurements used for the performance assessment. Triangles depict an imprecise or bounding measurement.

can also collect *several* measurements for different budgets or different targets in a single run. If the discretization is fine enough, we collect in both cases a loss free representation of the convergence graph, see below.

1) Discussion of the fixed-budget approach: The major advantage of a fixed-budget approach is the straightforward experimental setup and execution. A meaningful budget is easy to choose and the overall time needed to run the experiment is straightforward to predict.

However, there is a caveat with larger budgets: algorithms may naturally terminate before the budget is exhausted. In this case, independent randomized restarts could be applied.[13] However, terminated algorithms may return a sufficiently good solution long before the budget is exhausted, as for example in the leftmost graph in Figure 1 (blue triangle). In a fixed-budget setup, exceptionally good algorithms cannot be directly distinguished from algorithms that reach the global optimum just within the given budget.

The measurement outcome in the fixed-budget approach is a set of quality indicator values. Without specific knowledge of the function and the quality indicator mapping, only the order of these values can be interpreted in a meaningful way. The observation that, say, an Algorithm A reaches a *two times smaller* quality indicator value than Algorithm B can in general not be interpreted such that Algorithm A is *two times better* than Algorithm B. There is no *a priori* way to quantify *how much* better or more difficult it is to reach a two times smaller indicator value. Even a *mean* indicator value may not be (semantically) admissible as it requires data with interval scale [29]. A fixed-budget measure exhibits in general only an ordinal (rank) scale. Whether any higher level quantification is meaningful depends on the function, the definition of the quality indicator and may even depend on the specific indicator values that are compared. Finally, aggregating or mixing results from different functions arguably introduces the issue of mixing different units of measurement.

[13]Restarts can sometimes be favorably combined with a parameter sweep, for example for the population size in an evolutionary algorithm [21], [4].

2) Discussion of the fixed-target approach: The experimental setup in the fixed-target approach requires to find appropriate target values for the quality indicator. Although any well-posed definition of an optimization problem should entail a definition of the target,[14] setting up targets can be difficult with new or scarcely studied functions and may require preliminary experiments. The choice of the target value(s) is however instrumental: it determines the difficulty and often the characteristic of the problem to be solved.[15]

The measurement outcome in the fixed-target approach is a set of runtimes. The major advantage of this setup is that this measurement allows conclusions of the type *Algorithm A is two (or ten, or a hundred) times faster than Algorithm B in solving this problem*. That is, the assessed measure is quantitative on a ratio scale. Runtimes from different target values or different functions are measured in the very same unit and can in general be meaningfully aggregated. Additionally, runtimes can be understood and interpreted without problem-specific knowledge.

3) Comparing both approaches: In the fixed-budget approach, we use a simple-to-interpret measure for the setup and leave the more difficult to interpret measure to be assessed by the examiner. In the fixed-target approach, we must invest in a more involved setup for getting a simple-to-interpret measure in the assessment. The main obstacle has in both approaches the same origin: to determine or interpret quality indicator values. This difficulty has to be resolved in the fixed-target setup *before* data can be displayed, ideally by exploiting all available domain expertise and engaging domain experts. In the fixed-budget setup, the obstacle appears only when results are displayed and remains in essence unresolved. An inexperienced examiner is often unaware of the low measurement scale of such data which may lead to decisive misinterpretations.

In conclusion, the fixed-budget approach is preferable for a quick and simple experimental setup which can be useful for explorative experimentation. Yet, plotting the ensemble of indicator graphs over time is in this case usually the method of choice. The fixed-target approach is superior for assessing the measurement outcome because it gives *quantitatively and meaningfully interpretable* results.

D. Imprecise Values

In both cases, the fixed-budget and fixed-target approach, we can encounter an "imprecise value situation": in Figure 1, not all graphs intersect with either the vertical or the horizontal line (red and blue triangles). As already mentioned, an algorithm might solve the function long before the budget is exhausted. This algorithm performs "optimally" and better than the fixed-budget measurement is able to reflect precisely, which can lead to significant misjudgments of what the best

[14]"Optimizing a function" without defining the target quality of the solution is not a well-posed problem, unless we tacitly assume that the global optimum is desired. In continuous search spaces, the best we can generally hope for is a close approximation of the global optimum which likewise requires a specification of the meaning of "close".

[15]For example, the well-known Rastrigin function is highly multimodal in the vicinity of the global optimum but it has no local optima and is easy to optimize outside of the domain $[-10\pi, 10\pi]^n$, that is, for target f-values greater than $10^3 n$ where n is the search space dimension.

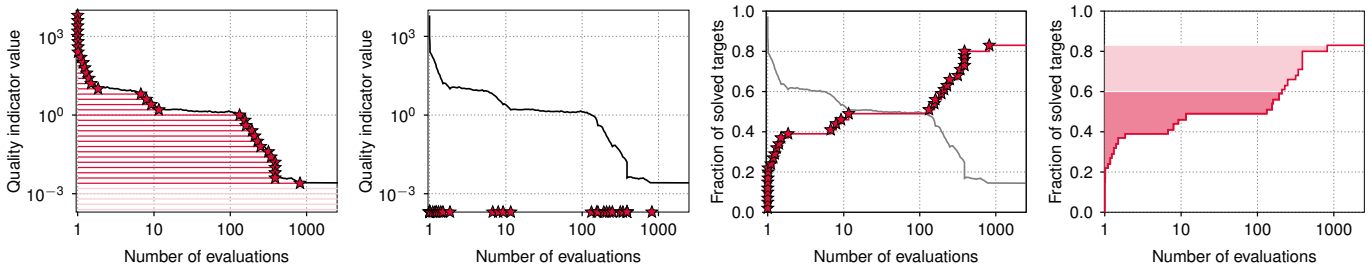

Fig. 2. Construction of the (anytime) empirical runtime distribution (RTD) from the quality indicator convergence graph and equally spaced target values. The lengths of the horizontal lines in (a) represent the runtime measurements for different target values taken from the quality indicator convergence graph, as do the stars in (a)–(c). When the target values are evenly spaced on the given y-axis, the RTD reconstructs, up to discretization, the lower envelope of the convergence graph flipped upside down, (c). The areas above the RTD in (d) represent average runtimes, in that the logarithm of the geometric average of a fraction of the smallest runtimes equals the area size divided by the trim fraction. The dark area corresponds to the shortest 60%, dark plus light area correspond to the fraction of all *solved* problems.

performing algorithm is. A remedy is to define a *final* target value and measure the runtime when the final target is hit.[16]

On the other hand, an algorithm may never hit a (difficult) target under the given experimental conditions, giving raise to a missing runtime value in the fixed-target approach (red triangle in Figure 1). The observed runlength is a lower bound for this "missing" runtime measurement. To reduce the number of imprecise or missing runtime values, we can i) run the algorithm longer, depending on the observed termination conditions with or without restarts; ii) use the final quality indicator value as additional (inferior) measurement; iii) apply simulated restarts based on the existing or added repetitions (see Section V-A).

E. First Hitting Time

The runtime connected to a given target value is the intersection between a quality indicator graph and the horizontal target line, as depicted in Figure 1. The intersection is however not necessarily unique and we adopt the *first* intersection (first hitting time) to measure the runtime. This has the advantage that prolonging a run (increasing the budget) cannot lead to a worse outcome which is consistent with the expectation that algorithms should not achieve, in general or on average, worse results when run with a larger timeout budget. A nonmonotone measure may disincentivize the use of larger experimental budgets and is not likely to reflect reality well.

Using the first hitting time implies that the approach becomes budget-free: *increasing the budget can fill missing values, but it cannot change measurements that are already obtained with a smaller budget*. Based on this insight, in practice we often repeat an entire experiment with successively increasing budgets by a factor between two and five. Such a scheme has comparatively little time overhead, but allows to investigate results early on and fix potential bugs earlier. A similar scheme can also be applied to benchmark budget-dependent algorithms [31] as mentioned above.

Figure 2 illustrates collecting several runtime measurements from a single run by setting several target values. We also see the missing values for targets below $10^{-2.7}$ (Figure 2a). As the timeout budget is the lower bound for these missing runtime values, terminating an algorithm when it stagnates decreases this lower bound. The figure also visualizes the construction of the runtime distribution from equally spaced targets in three steps, see the caption for details.

Collecting runtimes from several targets in *the same* runtime distribution graph is a main distinction to data profiles [25].

F. Target Value Settings

As noticed in Section III-C, the requirement to define target values for the quality indicator is the main disadvantage of the fixed-target approach. Here, we discuss different methodologies to establish these values.[17] Generally, targets should neither be so easy to allow trivial algorithms to hit them within a few iterations nor be so difficult that they can not be achieved at all. Exploratory experiments can at least easily confirm the first condition. We describe the target values $\tau(\varepsilon)$ by a reference value that depends on the problem instance p and one or several offsets or target precision values ε,

$$\tau(\varepsilon) = l^p_{\mathrm{ref}} + \varepsilon \ , \tag{1}$$

where in some cases also the precision ε may depend on p. The reference value, l^p_{ref}, is either the true optimal (here minimal) indicator value, or an estimate of the optimal value, or the best value observed from running a set of algorithms up to a given budget. Different ε are usually placed equidistant on the log-scale. Often, the same ε-values are used across all functions. We also exploit comparability across instances of the same function: for a given ε, we assume that the problems $p^{(5)} = (n, m, f_\theta, \theta, \tau(\varepsilon))$ are similar for varying θ. We give a few specific examples for setting target values for different problem classes.

[16]This is also advisable from a practical perspective because an algorithm which approximates the optimum with a tight error of, say, 10^{-50} is usually *not* preferable to an algorithm which approximates the optimum with the error of, say, 10^{-20}: achieving the tighter error mainly indicates a lack of practical termination conditions, hence the algorithm is arguably even worse. In continuous domain, the solution set that conclusively solves a problem (numerically or practically) contains almost always a small neighbourhood of the global optimum or a compact set containing the optimum.

[17]We only discuss the methodological aspect. A technical aspect is how to keep the amount of data that needs to be logged and processed manageable. Storing data at each time step is often impractical or even prohibitive.

1) Moré and Wild, introducing data profiles [25], benchmark a set of algorithms and set l^p_{ref} to the smallest value achieved by any of the algorithms (within the given budget). They set ε depending on the initial point x_0 as $\varepsilon = 10^{-k}(f(x_0) - l^p_{\mathrm{ref}})$ and show results for $k \in \{1, 3, 5, 7\}$ each in a different graph. The targets represent the reduction from the first value to the best *observed* value up to $\varepsilon \times 100$ percent. All targets are reached by at least one algorithm which can be considered as an advantage of this approach to set l^p_{ref}.
2) In the single-objective benchmark suites of COCO [16], l^p_{ref} is set to the f-value of the true optimum and $\varepsilon = 10^k$ for $k \in \{k_{\min}, k_{\min}+0.2, \ldots, 1.8, 2\}$ where $k_{\min}$ is either -8 or -6. The wide range of target values echoes the anytime aspect of this approach. All measured runtimes are presented in a single graph. To present results for single targets, k is chosen similarly to $-k$ in Item 1. To apply this approach on a suite of functions, we scale each function such that the *same* target precision values cover the meaningful range of each function topography [30, Section 3.1.3].
3) In the bi-objective benchmark suite, `bbob-biobj`, of COCO, l^p_{ref} is the quality indicator of an approximation of the Pareto front [9], based on running many algorithms with a large budget. Here, $\varepsilon \in \{-10^k \,|\, k \in \{-4, -4.2, \ldots, -5\}\} + \{0\} + \{10^k \,|\, k \in \{-5, -4.9, \ldots, 1\}\}$. Targets with negative ε are achieved and useful when algorithms find a better than the currently known approximation. They make it less likely that a change of targets becomes necessary, thereby breaking comparability across runtime data sets and publications.
4) Budget-based targets (also called runlength-based targets) are based on algorithm performance data. Loosely speaking, for any given budget, it is *the easiest (largest) target that no reference algorithm can reliably reach within this budget*.[18][19] They generalize the targets used in data profiles (Item 1): budget-based targets separate reference algorithms from the investigated algorithms and allow for any number of different budgets. Setting budget-based targets does not require any knowledge about the function (apart from empirical data obtained on the function) or any particular expertise of a human designer, because only the reference *budgets* have to be chosen a priori. Reference budgets, as runtimes, are intuitively meaningful quantities which are fairly easy to set. Budget-based targets however depend on the choice of a reference data set and hence on a set of reference algorithms.

We highlight two main distinctions between these four specific approaches. First, only Item 2 does *not use data* to determine the target values and hence provides a faithful indication whether the global optimum was found. Second, results are *inherently comparable across publications* except with data profiles (Item 1).

IV. Runtime Computation

From a *single run* of an algorithm on the problem instance $p^{(4)} = (n, m, f_\theta, \theta)$, as shown in Figure 2a, we attempt to measure the runtime for each target precision ε, that is, for each problem $p^{(5)} = (n, m, f_\theta, \theta, \tau(\varepsilon))$. Choosing many targets reflects the anytime aspect of the performance evaluation. When the quality indicator value drops below the target *for the first time*, the problem $p^{(5)}$ is considered as solved and the runtime on $p^{(5)}$ is measured as the current evaluation count. If the problem is not solved the runtime remains undefined and "missing". In every single run, the measured runtime is nondecreasing with increasing target difficulty (decreasing ε).

A. Runs on Different Instances

Different instantiations of the parametrized functions f_θ are a natural way to represent randomized independent repetitions. Similarly, randomized restarts can be conducted from different initial points. For translation invariant algorithms, translating the instance or the initial point are equivalent and hence mutually exchangeable.

For the aggregating measures in the next section, we interpret runs performed on different instances $\theta_1, \ldots, \theta_K$ as repetitions of the same problem. Thereby, we assume that instances of the same parametrized function f_θ are similar to each other and have similar landscape features, or, more concisely, could reasonably well model randomized restarts on a single problem in the real world. For example, the benchmark suites of the COCO framework have for each parametrized problem a set of $K \approx 15$ instances. When independent restarts are practically meaningful, runs on these instances are used to compute artificial runtimes of a conceptual restart algorithm. To see at least seven successes for algorithms with a success rate smaller than $1/2$, we recommend, with decreasing preference, to either conduct several trials on each instance (e.g., run the benchmark several times on the concerned functions) or use more instances (both is possible in COCO) or simply apply within-trial restarts.

B. Runtime as a Random Variable

Formally, the runtime $\mathrm{RT}^{\mathrm{s}}(p^{(5)})$, simply denoted as $\mathrm{RT}^{\mathrm{s}}(p)$ in the following, is a random variable that represents the number of evaluations needed to reach the quality indicator target value for the first time. A run or trial that reached the target

[18] More concisely, we determine the budget-based target by taking the set of all target values and remove those targets for which the expected runtime (ERT, see Section V-B) of any reference algorithm stays within the budget. The easiest target (the maximum, the supremum) in the remaining set is the sought-after budget-based target, or, if the set is empty, it is the most difficult of all targets. When we want to prevent duplicates, for example for "single-target presentations", we start with the smallest budget and never reset the considered set of target values in the iterative process.

[19] Budget-based targets are available for the single-objective `bbob` benchmark suite [17] of COCO in the so-called expensive optimization scenario. They are based on data of the reference algorithms from [19] and, due to the available data, on the target "discretization" described in Item 2. The 31 budgets $10^k \times n/2$ with $k \in \{0, 1/15, \ldots, 29/15, 2\}$ were used to create, once and for all, sets of 31 target precision values that depend on n and f but not on θ.

value is called *successful*.[20] For *unsuccessful trials*, the runtime is not defined, but the overall number of evaluations in the given trial is the associated random variable and denoted by $\mathrm{EV}^{\mathrm{us}}(p)$. In a single run, the value of $\mathrm{EV}^{\mathrm{us}}$ is the same for all failed targets.

We consider the conceptual *restart algorithm*. Given an algorithm has a strictly positive probability p_{s} to solve a problem, independent restarts of the algorithm solve the problem with probability one and exhibit the runtime

$$\mathrm{RT}(p) = \sum_{j=1}^{J} \mathrm{EV}_j^{\mathrm{us}}(p) + \mathrm{RT}^{\mathrm{s}}(p) \ , \tag{2}$$

where $J \sim \mathrm{NB}(1, 1-p_{\mathrm{s}})$ is a random variable with negative binomial distribution that models the number of unsuccessful runs until one success is observed and $\mathrm{EV}_j^{\mathrm{us}}$ are independent random variables corresponding to the evaluations in unsuccessful runs [3].

Generally, the above equation for $\mathrm{RT}(p)$ expresses the runtime from repeated (randomized) independent runs on the same problem instance (while the instance θ is not given explicitly). By drawing instances θ of $p^{(5)}$ uniformly at random, we can apply the equation to runs on different instances θ. In order to model independent restarts, as desired in Sections V-A and V-B, instances should always come from the same function with the same dimension and the same target precision value ε in Eq. (1) which hence must be comparable between these instances.

V. Aggregating Measures

When benchmarking, we often collect tens of thousands of single performance data [16] which need to be assembled to become amenable to the user. The semantic idea behind aggregation is to combine performance results from problems for which i) we presume a uniform distribution (they are similar likely to appear in practice) and ii) there is no cheap way to distinguish them, like by dimension or other properties or attributes or cheap probing features. Empirical runtime distributions (RTD) are a generic way to aggregate data into single graphs without literally combining them. They are discussed in Section VI.

In this section, we first present two aggregating measures that directly combine successful and unsuccessful runs. Assuming uniform instances, as described in Section IV-A, this aggregation rigorously solves the problem to compare algorithms whose runtimes *and* success rates are different. Both aggregating measures have a natural caveat: if only a few successes are observed, we will see large variations without statistical significance.

Section V-C recalls the geometric average and its ratios as a prominent tool to aggregate results from different functions into a single number.

[20]The success notion is directly linked to a target value. A run can be successful with respect to some target values (some problems) and unsuccessful with respect to others. Success sometimes refers to the final, most difficult (smallest) target value, which implies success for all other targets in this run.

A. Simulated Restarts for Simulated Runtimes

Based on the conceptual restart algorithm as given in (2), simulated restarts generate randomized empirical runtimes from measured data that contain both, successful *and* unsuccessful runs. These runtimes must be interpreted with great caution when *manually set budgets* were exhausted (in unsuccessful runs) or when only few successful runs were recorded.

We use K different runs on the same function and dimension with a fixed target precision ε, that is, a set of problems $\{p^{(5)} = (n, m, f_{\theta_i}, \theta_i, \tau(\varepsilon)) \,|\, i = 1, \ldots, K\}$ to simulate virtual restarts. We need to have at least one successful run: otherwise the runtime remains undefined because the virtual procedure would never stop. Then, we construct artificial, simulated runs from the available empirical data: we repeatedly pick, uniformly at random with replacement, one of the K trials until we encounter a successful trial. This procedure simulates a single sample of the virtually restarted algorithm from the given data. Given in Eq. (2) as $\mathrm{RT}(p)$, the acquired simulated runtime is the sum of the number of evaluations from the unsuccessful trials added to the runtime of the last and successful trial.[21]

When all K runs were successful, no restarts are simulated. When all runs were either successful or terminated by standard termination conditions in the benchmarked algorithms, simulated restarts reflect the original algorithm with independent restarts and their expected runtime equals to the ERT as defined in Section V-B [19].

1) Bootstrapping: In practice, we repeat the above procedure hundreds or thousands of times, thereby sampling N simulated runtimes from the same underlying distribution, resembling the bootstrap algorithm [11]. To reduce the variance in this procedure, when desired, the first trials for each sample are picked *without replacement*[22] and only further trials are picked uniformly at random from all data.

2) Rationales and Limitations: Simulated restarts aggregate some of the available data and thereby extend their range of interpretation.

1) Simulated restarts allow in particular to compare algorithms with vastly different success probabilities by a single performance measure. Likewise, conducting restarts is often a valuable approach when addressing difficult optimization problems in practice.
2) Simulated restarts assume that *independent* restarts, without a memory, are suitable in the considered optimization scenario (and for the algorithms). A counterexample is the maximization of hypervolume on multiobjective problems where competitive search algorithms need to take into account all so-far evaluated nondominated solutions. Hence, independent and memory-less restarts do not suit this scenario. Additionally, measuring the hypervolume in

[21]That is, we apply (2) such that RT^{s} is uniformly distributed over all measured runtimes from successful instances θ_i, $\mathrm{RT}^{\mathrm{us}}$ is uniformly distributed over all evaluations seen in unsuccessful instances θ_i, and J has a negative binomial distribution $\mathrm{NB}(1, q)$, where q is the ratio of unsuccessful instances.

[22]The variance reducing effect is best seen in the case where all runs are successful and $N = K$, in which case each data is picked exactly once. This technique is not suited to estimate the deviation of the given data set from the original underlying distribution [11].

the assessment requires to keep and merge all recorded function values from previous trials to maintain a non-dominated archive. For these reasons, simulated restarts are not applied with the bi-objective benchmark suites of COCO [7].

3) Simulated restarts rely on the assumption that the instances have similar landscape features. When this is not the case, performance measures based on simulated restarts are more difficult to interpret and should rather be avoided.
4) Simulated restarts rely on the definition of comparable target precision values, ε in Eq. (1), for all instances θ_i.
5) The runtime of simulated restarts heavily depends on *termination conditions* applied in the benchmarked algorithm, due to the evaluations spent in unsuccessful trials, compare Eq. (2). This can be considered as disadvantage, when termination is a cumbersome but trivial detail in the implementation[23]—or as an advantage, when termination should be considered a relevant algorithm component in a practical application. Omitting to activate termination settings can lead to long stagnations and hence a too pessimistic assessment through simulated restarts. Fortunately, such stagnations can be easily identified in the empirical runtime distribution graphs discussed below.
6) The maximal runtime for which simulated restarts are meaningful and representative also depends on the experimental conditions and the outcome. If a tight maximum budget was imposed for the purpose of benchmarking, simulated restarts do not reliably reflect true performance, in particular when very few or no successes are observed.

B. Expected Runtime

The average or expected runtime (ERT, [15]), introduced as ENES[27] and analyzed as success performance SP2 [3], estimates the expected runtime of the restart algorithm given in (2). If the success rate is one, ERT reduces to the (arithmetic) average runtime. Generally, as for simulated restarts, the set of trials over which the average is taken varies in θ only (and not in ε).

The expectation of the runtime of the restart algorithm given in (2) writes [3]

$$\mathbb{E}(\mathrm{RT}) = \mathbb{E}(\mathrm{RT}^{\mathrm{s}}) + \frac{1-p_{\mathrm{s}}}{p_{\mathrm{s}}}\,\mathbb{E}(\mathrm{EV}^{\mathrm{us}}), \tag{3}$$

where $p_{\mathrm{s}} > 0$ is the success probability of the algorithm.

Given a data set with $n_{\mathrm{s}} \geq 1$ successful runs with runtimes $\mathrm{RT}^{\mathrm{s}}_i$, and n_{us} unsuccessful runs with $\mathrm{EV}^{\mathrm{us}}_j$ evaluations, an estimate of $\mathbb{E}(\mathrm{RT})$, referred to as expected runtime, reads

$$\begin{aligned}\mathrm{ERT} &:= \frac{1}{n_{\mathrm{s}}}\sum_i \mathrm{RT}^{\mathrm{s}}_i + \frac{1-p_{\mathrm{s}}}{p_{\mathrm{s}}}\,\frac{1}{n_{\mathrm{us}}}\sum_j \mathrm{EV}^{\mathrm{us}}_j \\ &= \frac{\sum_i \mathrm{RT}^{\mathrm{s}}_i + \sum_j \mathrm{EV}^{\mathrm{us}}_j}{n_{\mathrm{s}}} = \frac{\#\mathrm{evals}}{n_{\mathrm{s}}},\end{aligned}$$

where p_{s} is now the fraction of successful trials, i.e. $p_{\mathrm{s}} = n_{\mathrm{s}}/(n_{\mathrm{s}}+n_{\mathrm{us}})$ estimating the success probability used in (3), $0/0$ is understood as zero and $\#\mathrm{evals}$ is the number of evaluations conducted in all trials before reaching the given target precision. The ERT is an unbiased estimator of the expected value of the simulated runtimes [19].[24] Assuming the runtime increases (slower than exponentially) with dimension, plotting the ERT divided by dimension against dimension in a log-log plot is our recommended generic way to investigate the scaling behavior.

Rationale and Limitations: The *arithmetic* average over instances is only advisable if each instance obeys a similar distribution without heavy tails. If the distribution has a heavy tail, the average will suffer from large variations. If one instance is considerably harder than the others, the average is dominated by this instance. In such cases, we advise against the *arithmetic* average in favor of the geometric average.

C. Geometric Average Runtime Ratio

We consider the *geometric average runtime*, as represented by the area above the graph in Figure 2d, to be a measure of central importance. Providing a quantitative *comparison*, the geometric average *ratio* between two data sets $(A_1, \ldots, A_k)$ and $(B_1, \ldots, B_k)$ is defined as

$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{A_i}{B_i}\right)\right) = \frac{\sqrt[k]{\prod_{i=1}^{k} A_i}}{\sqrt[k]{\prod_{i=1}^{k} B_i}}\,. \tag{4}$$

When the data are runtimes, this equation reflects the difference area between two empirical runtime distributions when the x-axis is in log-scale. The RHS of Eq. (4) reveals that this ratio is invariant to reordering of the data and to renormalization of according data in both sets (like for so-called performance profiles [10]), and that it can easily be accommodated to different numbers of data, k, in each set. If $\prod_{i=1}^{k} B_i = 1$, Eq. (4) is the geometric average of $(A_1, \ldots, A_k)$.

In contrast to the arithmetic average, the geometric average is not dominated by the variation of the largest values (and it is not hampered by values close to zero in our scenario). It can be meaningfully used with data from different functions and with widely different expected runtimes.

VI. Empirical Runtime Distributions (RTD)

Runtime, as random variable, is fully described by its distribution and we generally recommend to display the empirical cumulative distribution function (ECDF) of a set of measured runtimes or simulated runtimes—in short, empirical *runtime distributions (RTD)* [23], [24]. Empirical runtime distributions display on the y-axis the *proportion of problems* solved within the budget specified on the x-axis. As importantly in our context, RTDs give for each fraction of problems (given on the y-axis) the largest measured runtime on the x-axis.

[23] A quick fix is the so-called SP1 [3] or Q-measure [12] which considers $\mathrm{EV}^{\mathrm{us}}$ to be distributed as RT^{s}.

[24] Randomness here comes from sampling uniformly among the measured runtimes whereas $p_s = n_s/(n_s+n_{\mathrm{us}})$ is then the *true* probability to sample a successful among the *measured* runtimes.

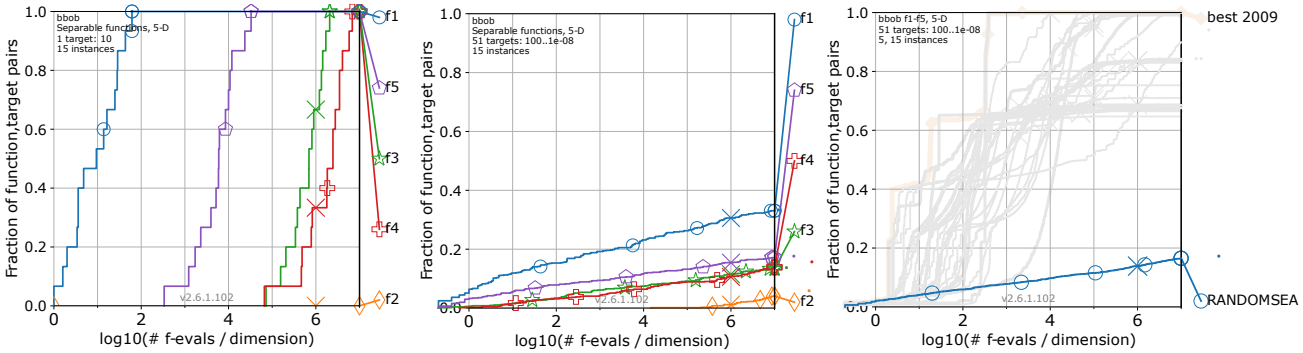

Fig. 3. Empirical runtime distributions (RTD) for random search on various 5-D problems. Left and middle: for the separable functions f_1 to f_5 from the `bbob` suite for the single target precision $\varepsilon = 10$ (left) and for 51 target precisions, equidistant on the log-scale between 100 and 10^{-8} (middle). Right: Empirical runtime distribution of the same data as in the middle plot aggregated for all five functions in one graph. Crosses indicate the median runlength of unsuccessful runs which equals the maximum budget when all runs exhausted this budget. Then, increments of the graphs beyond the cross stem from simulated restarts. Small dots on the right are the fraction of all function-target pairs for which at least one instance was successful. After this point, the runtime distribution stays flat. Without annotation in gray are RTDs of 30 further algorithms [19]. The "best 2009" depicts a portfolio algorithm from all shown RTDs (see text).

A. Rationale, Interpretation and Limitations

ECDFs are a universal technique to display *unlabeled* data in a condensed way without losing information, because each data point is "separately" displayed. They remain entirely meaningful under monotonous transformations (e.g., taking the logarithm) of the data (here runtimes).

1) As shown in Figure 2, the empirical distribution function of a set of runtimes from problems $p^{(5)} = (n, m, f_\theta, \theta, \tau(\varepsilon))$ *where only the target precision value* ε *varies* in a uniform way recovers the lower envelope of the convergence graph upside-down with the resolution defined by the targets [19]. In this scenario, we can identify the associated target values in the displayed data.

2) When runs from several instances are aggregated, the association to the single runs (and to the targets) is lost. Likewise, the association to the functions is lost when aggregating over several functions.

3) Single target RTDs are a common, legitimate display. Multitarget RTDs use benchmarking data more comprehensively and could be characterized as *anytime* runtime distributions.

The ECDF can be read in two distinct ways.

1) *x-axis as independent variable:* for any budget (x-value), we see the fraction of problems solved within the budget as y-value, where the limit value to the right is the fraction of problems solved within the maximal budget.[25] The resulting value satisfies the desired properties of a measurement listed in Section III except that it may often not assume a wide range of values, because it is by definition between 0 and 1.

2) *y-axis as independent variable:* for any fraction of shortest runtimes (y-value), we read the maximal runtime observed on these problems on the x-axis. The area below the y-value and above the graph reflects the average runtime on this subset of problems. When the x-axis is plotted on a log-scale, it is the logarithm of the *geometric* average and a horizontal shift indicates a runtime difference (ratio) by the respective factor, quantifiable, e.g., as "five times faster". The second interpretation is, to our estimate, lesser known but more informative in our context.

B. Examples

We show examples generated with the COCO framework [16]. Figure 3 shows various empirical runtime distributions (RTDs with simulated restarts and 1000 bootstraps) for uniform random search in $[-5, 5]^n$ on problems given by 15 instances of the first five functions of the `bbob` test suite [17] in dimension $n = 5$.

The left plot shows the RTD for each single function with a single target precision $\varepsilon = 10$. On the Sphere Function `f1`, the target precision is reached in all runs within $100 \times n$ evaluations. On the Separable Ellipsoidal Function `f2`, the target precision is never reached within $10^6 \times n$ evaluations.

The middle plot shows the runtimes for 51 target precisions between 10^2 and 10^{-8}, see Section III-F Item 2.[26] On the Sphere Function `f1`, almost 20% of the problems are solved within $1000 \times n$ evaluations. Runtimes to the right of the crosses at $10^6 \times n$ have at least one unsuccessful run and their further increase beyond this budget is induced by simulated restarts. This can be concluded, because pure random search exploited in all runs the same maximum budget. The small dot beyond $10^7 \times n$ evaluations depicts the fraction of target precisions that were reached at least once.

Empirical runtime distributions of single functions, in particular comparing different algorithms, are in our experience the most important data to scrutinize when investigating performance results.

The COCO framework defines subgroups of functions sharing similar properties (for instance separability, unimodality, etc.). The five functions plotted in Figure 3 belong to the separable function group. The right plot shows the same

[25] With simulated restarts, the limit value beyond the maximal budget is the average fraction of target precisions solved *at least once* in all respective instances. A high discrepancy between this limit and the (lower or equal) limit without simulated restarts indicates a high variation of the number of solved targets between instances.

[26] In contrast to Figure 2, these RTDs aggregate convergence graphs from 15 instances.

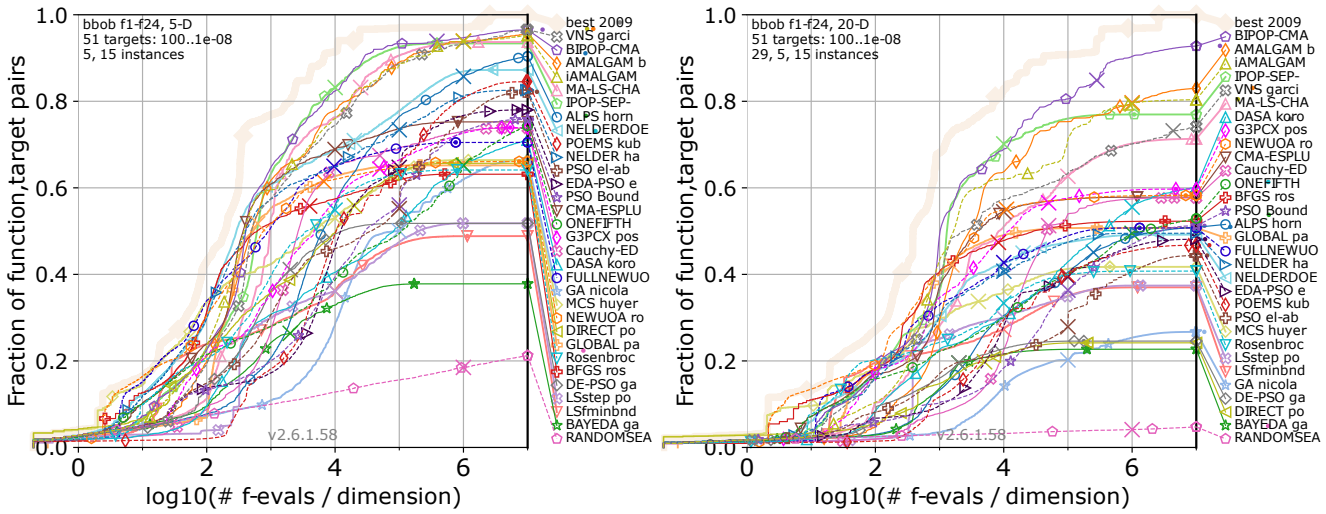

Fig. 4. Empirical runtime distribution over all functions and all targets for several algorithms benchmarked for the BBOB 2009 workshop in dimension 5 (left) and in dimension 20 (right) when aggregating over all functions of the bbob suite.

runtimes as in the middle but in a single graph (where the y-steps are respectively normalized). The runtime distribution is compared with 30 other algorithms (without annotations) and with the "best" portfolio algorithm from all displayed algorithms. The "best 2009" portfolio algorithm [19] picks for all problems with the same function, dimension and target precision the algorithm with the smallest ERT.[27] This portfolio algorithm is artificial because only with varying θ in $p^{(5)} = (n, m, f_\theta, \theta, \tau(\varepsilon))$ it always uses data from the same algorithm whereas otherwise data are chosen *a posteriori* based on performance.

Figure 4 shows runtime distributions of the 31 algorithms from [19] aggregated over all functions and 51 targets of the bbob test suite, that is, most of the available performance data from these algorithms in dimension 5 and 20. The artificial best 2009 algorithm is about two to three times faster than the left envelope of all single algorithm RTDs and solves all problems in about $10^7 \times n$ evaluations.

Although we highly recommend displaying results in graphs as above, numbers as well as statistical tests are occasionally useful to complement the visual comparison. To this end, COCO provides tables with ERT values relative to a reference algorithm, such as the "best 2009", see Table I. For effortless readability, the tables provide runtime *ratios*, with the absolute baselines given in the respective first row. The tables also show the order of magnitude of the two-sided p-values of a rank sum test.[28]

[27]The best 2009 curve is not guaranteed to be an upper left envelope of the RTDs of all algorithms from which it is constructed, that is, the RTD of an algorithm from BBOB-2009 can cross the best 2009 curve. This typically happens if an algorithm has, for some of the easier problems, a large runtime variation and its ERT is not the best but the short runtimes show up to the left of the best 2009 graph.

[28]The rank-sum test is based on data truncated to the largest budget for which all concerned unsuccessful trials (w.r.t. the respective target, without restarts) of both algorithms have a data record. The ranks (smaller are better) for unsuccessful trials are based on the exponential of the best achieved indicator value and for still successful trials on the negative of the inverse of the measured runtime. Significance is only concluded when also the ERT values identify the same algorithm as better.

TABLE I

EXPECTED RUNTIMES (ERT) TO REACH TARGETS $f_{\text{OPT}} + \varepsilon$, FOR THE ALGORITHMS BFGS AND BIPOP-CMA-ES DIVIDED BY THE ERT OF THE best 2009 ARTIFICIAL REFERENCE ALGORITHM AS SHOWN IN THE FIRST ROWS FOR THREE BBOB FUNCTIONS IN DIMENSION 5. NUMBERS IN BRACKETS ARE THE HALF DIFFERENCE BETWEEN 10TH AND 90TH-TILE OF BOOTSTRAPPED RUNTIMES. #SUCC IS THE NUMBER OF TRIALS THAT REACHED THE (FINAL) TARGET $f_{\text{OPT}} + 10^{-8}$. THE MEDIAN NUMBER OF CONDUCTED EVALUATIONS IS GIVEN IN ITALICS IF THE LAST COLUMN'S TARGET WAS NEVER REACHED. ENTRIES SUCCEEDED BY A STAR ARE STATISTICALLY SIGNIFICANTLY BETTER IN A TWO-SIDED RANK-SUM TEST WITH $p = 0.05$ OR $p = 10^{-k}$ WHEN THE NUMBER k FOLLOWS THE STAR, WITH BONFERRONI CORRECTION BY THE NUMBER OF FUNCTIONS (24). BEST RESULTS ARE IN BOLD

ε	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	11	12	12	12	12	12	12	15/15
BFGS	**1.2**(0)	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	15/15
BIPOP	3.2(2)	9.1(4)	15(4)	21(5)	28(5)	41(4)	54(5)	15/15
f2	83	87	88	89	90	92	94	15/15
BFGS	**3.9**(3)$^{\star 4}$	**5.7**(3)$^{\star 4}$	**6.3**(2)$^{\star 4}$	**6.5**(2)$^{\star 4}$	**6.6**(2)$^{\star 4}$	**6.9**(1)$^{\star 4}$	**7.1**(2)$^{\star 4}$	15/15
BIPOP	13(3)	16(3)	18(3)	19(2)	20(2)	21(2)	22(2)	15/15
f3	716	1622	1637	1642	1646	1650	1654	15/15
BFGS	107(281)	∞	∞	∞	∞	∞	∞ *2e4*	0/15
BIPOP	**1.4**(2)$^{\star 2}$	**16**(20)	**139**(322)	**139**(298)	**139**(351)	**139**(43)	**140**(557)	14/15

VII. RELATION TO PREVIOUS WORK

Recent reviews on benchmarking in optimization can be found in [6], [5]. More closely related to the presented work are empirical runtime distributions [24] that have been explored already in the late 1990s in the context of combinatorial optimization [23]. They were proposed in the setting of stochastic repetitions for finding the optimal solutions or a close approximation.

In the domain of continuous optimization, empirical runtime distributions were re-invented as *data profiles* [25], with the interpretation of measuring a *deterministic* success ratio from a set of different functions (rather than stochastic repetitions), plotted over a varying budget. Data profiles are commonly used for aggregating results from different functions and different dimensions to reach a *single* target precision [28]. They have been derived from the previously introduced performance profiles [10] which show distributions of runtime *ratios*.

Performance profiles are still considered as Gold standard in deterministic continuous optimization [6].

The idea to collect runtimes from *several* targets in a single empirical distribution [19] lead to the insight that runtime distributions generalize convergence graphs. In contrast to data profiles, aggregation over dimension was omitted and targets were chosen to be comparable across publications as presented in this paper.

In multiobjective optimization, the well-known *empirical attainment functions* [13] are a direct generalization of the fixed-budget approach where the empirical success probability is displayed over the f-vectors instead of a single f-value for a given budget [8]. Relaxing the fixed-budget constraint, the display can represent *expected runtimes* (instead of probabilities) to attain each vector in objective space [8].

VIII. Summary and Conclusions

The presented paper examines anytime performance assessments for blackbox optimization emphasizing the importance of using *quantitative* performance measures on a ratio scale. We first construct a quality indicator which formalizes the optimization goal for the problems at hand. This is exemplified for single-objective unconstrained and constrained optimization, noisy optimization and multiobjective optimization. Then, the methodology is not only independent of the search domain but also applies in the same way to a wide range of different optimization scenarios.

Starting from the indicator convergence graph, we can collect comparable data by either fixing a target or a budget. We discuss advantages and disadvantages of both approaches and conclude that for drawing meaningful *quantitative* conclusions, the fixed-target setup, where runtimes are measured for a single or several target indicator values, is strongly preferable.

Our approach is *anytime*, because we collect data for many different target values of the same function. It is *budget-free* (apart from a "timeout" budget), because running an algorithm longer does not change already measured runtimes. Anytime performance assessment however does not equal anytime optimization: runtime distributions reveal not only the number of evaluations an algorithm needed to attain a sought-after target (or solve a problem), they also reveal the performance for any *fixed budget* as the percentage of solved problems (or achieved targets). The latter allows to find the best candidate algorithm for a fixed budget in practice. Furthermore, we can assess also nonanytime *algorithms* that take a budget as input parameter by accommodating the experimental setup [31].

We emphasize the usage of empirical runtime distributions which provide a visual and quantitative display of performance with minimal loss of information allowing to faithfully display a large amount of data in a comprehensive way. Empirical runtime distributions generalize indicator convergence graphs by aggregating data across functions and targets. They present a superior alternative to presenting tables of indicator values at a given budget, which is still a common standard in the domain of multiobjective optimization. Yet, we found tables that present runtime *ratios*, similar to performance profiles, together with the baseline runtime denominator consistently useful.

One should generally be cautious when aggregating data: we do not recommend to aggregate over decisive variables that can be measured cheaply and used for algorithm selection (e.g., search or objective space dimension or number of constraints). Aggregation can also lead to biases: aggregating over dimension often overemphasizes the more frequent low-dimensional (test) functions.

Finally, most of the presented methodology is implemented in the open source platform COCO that also comes with hundreds of data sets of already benchmarked algorithms [16] which can be directly used in the `cocopp` module for comparison. Importantly, the implemented methodology allows for comparison also across publications. Additionally, new benchmarking data can be provided in online repositories.[29] Now, researchers can spend more time on exploring new ideas advancing science instead of reimplementing baseline algorithms or known benchmarking methodology for comparison. Additionally, the easiest way to adopt good assessment standards is by using already existing tools with these standards.

Appenix
Multiobjective Quality Indicator $I_{\text{HV+}}$

This section details the quality indicator used in the case of multiobjective optimization[30]. We first clarify some of the terms employed in its definition. The ideal point z_{ideal} is defined as the vector in objective space that contains the optimal (minimal) function value for each objective *independently*. The nadir point z_{nadir} consists of the worst value obtained by any Pareto-optimal solution in each objective. It can be estimated by taking the maximum value in each objective of all nondominated *extreme* points (single objective optima) instead of all Pareto-optimal solutions (the estimate is exact for two objectives). The set of points in the objective space bounded by the ideal and nadir points is called the region of interest (ROI).

Additionally, we generalize the standard Pareto dominance relation to sets by saying solution set $A = \{a_1, \ldots, a_{|A|}\}$ dominates solution set $B = \{b_1, \ldots, b_{|B|}\}$ if and only if for all solutions $b_i \in B$ there is at least one solution $a_j \in A$ that dominates it.

A. Indicator Definition

At time step t, the archive A_t contains all mutually nondominated solutions of the sequence of proposal solutions. When computing the multiobjective quality indicator $I_{\text{HV+}}$, the objective space is first normalized so that the ROI $[z_{\text{ideal}}, z_{\text{nadir}}]$ is mapped to $[0, 1]^m$.

The indicator value depends on whether the archive dominates the nadir point. If the nadir point is dominated by at least one point in the archive, $I_{\text{HV+}}$ is computed as the negative normalized hypervolume of the archive using the nadir point as the hypervolume reference point. If, on the other hand, the

[29] Similar to the data provided at https://numbbo.github.io/data-archive/ and using the `cocopp.archiving` module.

[30] Not implemented in COCO for more than 2 objectives.

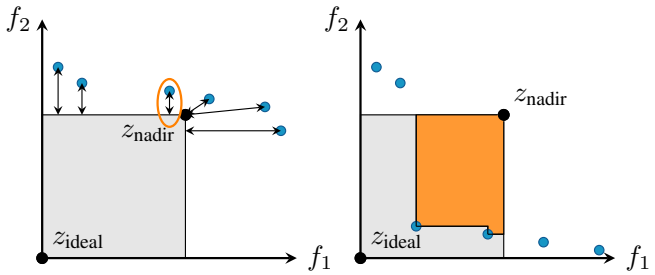

Fig. 5. Illustration of the quality indicator $I_{\text{HV+}}$ in two objectives when no (left figure) or at least one solution dominates the nadir point (right figure). Left: $I_{\text{HV+}}$ equals the shortest normalized distance between the solutions and the ROI (encircled). Right: $I_{\text{HV+}}$ is the negative normalized hypervolume of the solutions with the nadir point as the reference point, i.e., the orange area.

nadir point is not dominated by the archive, the indicator value equals the normalized distance of the archive to the ROI.

Formally, the multiobjective quality indicator $I_{\text{HV+}}$, we aim to minimize, is a function $A_t \to \mathbb{R}$ defined as

$$I_{\text{HV+}} = \begin{cases} -\text{HV}(A_t, [z_{\text{ideal}}, z_{\text{nadir}}]) & \text{if } A_t \text{ dominates } \{z_{\text{nadir}}\} \\ \text{dist}(A_t, [z_{\text{ideal}}, z_{\text{nadir}}]) & \text{otherwise} \end{cases},$$

where (with division understood to be the element-wise, Hadamard division)

$$\text{HV}(A_t, [z_{\text{ideal}}, z_{\text{nadir}}]) = \text{VOL}\left(\bigcup_{a \in A_t} \left[\frac{f(a) - z_{\text{ideal}}}{z_{\text{nadir}} - z_{\text{ideal}}}, 1\right]\right)$$

is the hypervolume of archive A_t normalized with respect to the ideal and nadir points that uses the nadir point as the reference point and

$$\text{dist}(A_t, [z_{\text{ideal}}, z_{\text{nadir}}]) = \min_{a \in A_t} \min_{z \in [z_{\text{ideal}}, z_{\text{nadir}}]} \left\| \frac{f(a) - z}{z_{\text{nadir}} - z_{\text{ideal}}} \right\|$$

is the smallest normalized Euclidean distance between a solution in the archive and the ROI $[z_{\text{ideal}}, z_{\text{nadir}}]$. See Figure 5 for an illustration of the $I_{\text{HV+}}$ indicator in the bi-objective case.

B. Properties

Solutions that are dominated by the archive do not change the value of $I_{\text{HV+}}$. Therefore, using the entire sequence of proposal solutions instead of the archive would result in the same indicator value.

The indicator value of an archive that contains the nadir point is 0. The indicator values are bounded from below by -1, which is the quality of an archive that contains the ideal point.

C. Rationales

Computing the multiobjective indicator on an archive instead of the current population has the advantage that no population size needs to be prescribed and algorithms with different or even changing population sizes can be easily compared. Furthermore, we believe that keeping an archive of nondominated solutions is relevant in real-world applications, in particular when blackbox evaluations are expensive.

Although other quality indicators could be used in place of the hypervolume, the monotonicity of the hypervolume is a strong theoretical argument for using it in the performance assessment: the hypervolume indicator value of the archive improves if and only if a new nondominated solution is generated [32].

D. Limitations

Because computing the hypervolume is time-consuming, it is best done incrementally when possible. There are now efficient ways to do so in the case of two and three objectives [14].

Not knowing the optimal quality indicator value can be an issue in the multiobjective case, even if the objectives' individual optima are known. In this case, we collect the nondominated solutions from as many preliminary experiments as possible and use their $I_{\text{HV+}}$ value as an upper bound for the optimal indicator value. Archiving the evaluated nondominated solutions of new algorithm runs could then be used to improve the approximations of the Pareto sets and the optimal indicator values from time to time.

ACKNOWLEDGMENTS

The authors would like to thank Steffen Finck, Tobias Glasmachers, Oswin Krause, Ilya Loshchilov, Olaf Mersmann, Petr Pošík, Raymond Ros, Marc Schoenauer, Thanh-Do Tran, and Dejan Tušar for their many invaluable contributions to the COCO platform.

REFERENCES

[1] D. V. Arnold and J. Porter, "Towards an augmented Lagrangian constraint handling approach for the (1+1)-ES," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 249–256.

[2] A. Atamna, A. Auger, and N. Hansen, "Linearly convergent evolution strategies via augmented Lagrangian constraint handling," in *Proceedings of the 14th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, 2017, pp. 149–161.

[3] A. Auger and N. Hansen, "Performance evaluation of an advanced local search evolutionary algorithm," in *IEEE Congress on Evolutionary Computation (CEC 2005)*, vol. 2. IEEE, 2005, pp. 1777–1784.

[4] ——, "A restart CMA evolution strategy with increasing population size," in *IEEE Congress on Evolutionary Computation (CEC 2005)*, vol. 2, 2005, pp. 1769–1776.

[5] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. López-Ibáñez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise, "Benchmarking in optimization: Best practice and open issues," *CoRR*, vol. abs/2007.03488, 2020. [Online]. Available: https://arxiv.org/abs/2007.03488

[6] V. Beiranvand, W. Hare, and Y. Lucet, "Best practices for comparing optimization algorithms," *Optimization and Engineering*, vol. 18, no. 4, pp. 815–848, 2017.

[7] D. Brockhoff, A. Auger, N. Hansen, and T. Tušar, "Using well-understood single-objective functions in multiobjective black-box optimization test suites," *Evolutionary Computation*, vol. 30, no. 2, pp. 165–193, 2022. [Online]. Available: https://hal.inria.fr/hal-01296987

[8] D. Brockhoff, A. Auger, N. Hansen, and T. Tušar, "Quantitative Performance Assessment of Multiobjective Optimizers: The Average Runtime Attainment Function," in *Evolutionary Multiobjective Optimization (EMO 2017)*. Springer, 2017, pp. 103–119.

[9] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger, "Biobjective performance assessment with the COCO platform," *CoRR*, vol. abs/1605.01746, 2016. [Online]. Available: http://arxiv.org/abs/1605.01746

[10] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[11] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

[12] V. Feoktistov, *Differential evolution*. Springer, New York, NY, USA, 2006.
[13] V. Grunert da Fonseca, C. M. Fonseca, and A. O. Hall, "Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function," in *Evolutionary Multi-Criterion Optimization (EMO 2001)*. Springer, 2001, pp. 213–225.
[14] A. P. Guerreiro and C. M. Fonseca, "Computing and updating hypervolume contributions in up to four dimensions," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 449–463, 2018.
[15] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking 2009: Experimental setup," INRIA Saclay—Ile-de-France, INRIA Research Report RR-6829, May 2009, updated February 2010.
[16] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, vol. 36, pp. 114–144, 2021.
[17] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions," INRIA, Tech. Rep. RR-6829, 2009, updated February 2019. [Online]. Available: https://coco.gforge.inria.fr/downloads/download16.00/
[18] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff, "COCO: The experimental procedure," *CoRR*, vol. abs/1603.08776, 2016. [Online]. Available: http://arxiv.org/abs/1603.08776
[19] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík, "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009," in *Genetic and evolutionary computation conference companion (GECCO 2010)*. New York, NY, USA: ACM, 2010, pp. 1689–1696.
[20] N. Hansen, D. Brockhoff, O. Mersmann, T. Tušar, D. Tušar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger, "Comparing continuous optimizers: numbbo/COCO on Github," *Zenodo*, doi: 10.5281/zenodo.2594848, Mar 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2594848
[21] G. R. Harik and F. G. Lobo, "A parameter-less genetic algorithm," in *Conference on Genetic and Evolutionary Computation (GECCO 1999)*. Morgan Kaufmann Publishers Inc., 1999, pp. 258–265.
[22] J. N. Hooker, "Testing heuristics: We have it all wrong," *Journal of heuristics*, vol. 1, no. 1, pp. 33–42, 1995.
[23] H. H. Hoos and T. Stützle, "Evaluating las vegas algorithms: pitfalls and remedies," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, 1998, pp. 238–245.
[24] ——, *Stochastic local search: Foundations and applications*. Elsevier, 2004.
[25] J. J. Moré and S. M. Wild, "Benchmarking derivative-free optimization algorithms," *SIAM J. Optimization*, vol. 20, no. 1, pp. 172–191, 2009.
[26] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.
[27] K. V. Price, "Differential evolution vs. the functions of the 2/sup nd/iceo," in *IEEE International Conference on Evolutionary Computation (ICEC 1997)*. IEEE, 1997, pp. 153–157.
[28] L. Rios and N. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.
[29] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946.
[30] T. Tušar, D. Brockhoff, and N. Hansen, "Mixed-integer benchmark problems for single-and bi-objective optimization," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2019, pp. 718–726.
[31] T. Tušar, N. Hansen, and D. Brockhoff, "Anytime benchmarking of budget-dependent algorithms with the COCO platform," in *Proceedings of the 20th International Multiconference Information Society, IS 2017*, vol. A. Jožef Stefan Institute, 2017, pp. 47–50.
[32] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance Assessment of Multiobjective Optimizers: An Analysis and Review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.