Simulation-based Test Functions for Optimization Algorithms

Martin Zaefferer, Andreas Fischbach, Boris Naujoks, Thomas Bartz-Beielstein
[firstname].[lastname]@th-koeln.de
TH Köln, Faculty of Computer Science and Engineering Science
Steinmüllerallee 1, 51643 Gummersbach, Germany

ABSTRACT

When designing or developing optimization algorithms, test functions are crucial to evaluate performance. Often, test functions are not sufficiently difficult, diverse, flexible or relevant to real-world applications. Previously, test functions with real-world relevance were generated by training a machine learning model based on real-world data. The model estimation is used as a test function. We propose a more principled approach using simulation instead of estimation. Thus, relevant and varied test functions are created which represent the behavior of real-world fitness landscapes. Importantly, estimation can lead to excessively smooth test functions while simulation may avoid this pitfall. Moreover, the simulation can be conditioned by the data, so that the simulation reproduces the training data but features diverse behavior in unobserved regions of the search space. The proposed test function generator is illustrated with an intuitive, one-dimensional example. To demonstrate the utility of this approach it is applied to a protein sequence optimization problem. This application demonstrates the advantages as well as practical limits of simulation-based test functions.

CCS CONCEPTS

•Theory of computation → Mathematical optimization; Gaussian processes; •Computing methodologies → Modeling and simulation;

KEYWORDS

Optimization, Test function generator, Simulation, Modeling

ACM Reference format:
Martin Zaefferer, Andreas Fischbach, Boris Naujoks, Thomas Bartz-Beielstein. 2017. Simulation-based Test Functions for Optimization Algorithms. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071190

1 INTRODUCTION

A crucial issue for the development, improvement and understanding of optimization algorithms are performance tests or benchmarks. Test functions are required to evaluate the performance of algorithms. It is particularly difficult to provide test functions for expensive optimization problems, where evaluations require high computational effort or other limited resources. Often, expensive optimization problems necessitate access to complex, confidential simulation codes, or access to expensive laboratory equipment. Even if access is granted, the evaluation costs make extensive tests infeasible. Only a limited number of expensive problems is openly available to the research community.

Thus, we need a generator of test functions which satisfy certain criteria. Besides important features, which are listed in well-known publications (e.g., [6, 30]) we focus on the following criteria.

(C.1) Difficulty: Test functions should be sufficiently complex [21]. Whitley [30] states that test "problems should be resistant to hill-climbing".
(C.2) Diversity: The problem instances are varied, randomized and not known *a priori*. This criterion is a standard in machine learning, because the available set of problem instances is partitioned into a training, a validation, and a test set [13].
(C.3) Flexibility: They should not be restricted to one specific problem instance. Flexibility is used in machine learning to characterize the number of parameters that are necessary to specify a model [15]. Flexibility will be used in our framework for characterizing functions. Some authors use the term "generalizability" to characterize this feature [2].
(C.4) Relevance: They should reflect real-world problem behavior.
(C.5) Evaluation cost: They should be inexpensive to evaluate, allowing for numerous tests.

One way to provide test functions that satisfy criteria (C.1)-(C.5) is to generate data-driven regression models of the objective function and use the derived predictor to test algorithms [2, 7, 8, 25]. This approach has an inherent problem: Almost all regression models interpolate the data they are trained on and hence yield smoothed fitness landscapes. Thus, the derived instances may be less rugged and more easy to solve than the real-world problem. Therefore, data-driven test functions should in addition respect the following criterion:

(C.6) Non-smoothing, i.e., the test instances reflect the ruggedness of the original problem.

Thus, the main research question examined in this study is: *How to generate test functions that satisfy criteria (C.1)-(C.6)?* To that end, we propose a general framework for generating test functions based on real-world data using simulation rather than estimation (prediction) techniques. Decisively, a simulation has the potential to avoid the pitfall of smoothing. Furthermore, it provides a principled way to generate diverse test function instances. To illustrate these features, we train Kriging models [5] on real-world data. The key idea is to use non-conditional and conditional simulation [5] of Kriging models to generate varied problem instances that do not smoothen the potentially rugged structure of the real-world problem. The simulation-based test functions can *reflect the behavior of the real-world problem rather than just the data itself.* These test functions are especially interesting for expensive optimization problems but obviously also apply to the cheap case.

Related approaches will be explained in Sec. 2. Afterwards, Sec. 3 will provide the details of the (non-)conditional Kriging simulation-based generator. A simple example is given in Sec. 4. To present a more complex application, the method is applied to a real-world data set in Sec. 5. Here, we also investigate practical limits of the approach. Section 6 discusses the applicability of our approach. Finally, Sec. 7 presents a summary and outlook for this work.

2 RELATED WORK

The most basic test function is a simple mathematical expression, e.g., the sphere function, which reflects the behavior of many test functions in the vicinity of the optimum [21]. In many studies, sets of such expressions are used as testbeds, e.g., combining the sphere, Branin, or Rosenbrock functions. These test suites should obey certain principles, e.g., nonlinearity, non-separability, and scalability [30]. The benefit of using sets of well established functions is that they enable comparability between different studies and can be used to guarantee generalizable results. Still, certain algorithms could easily be tailored to overfit a specific testbed, because the test functions are known in advance, i.e., before the study is performed. Furthermore, the capability of representing complex real-world behavior is probably limited. The generating principle of these classical test function suites [6, 21, 27] can be described as *inductive*, because single, simple features such as symmetry or multimodality are combined to generate a complex test function.

A more comprehensive approach is taken by the Comparing Continuous Optimizer platform (COCO), also known as the Black Box Optimization Benchmark (BBOB) [11]. BBOB comprises a framework that automates the experimental procedure involved in testing of continuous optimization algorithms. BBOB takes an inductive approach, relying on artificial test functions [12]. BBOB provides an enhanced procedure for post-processing of experimental results to enable a standardized comparison and analysis.

The *Gaussian Landscape Generator* (GLG), which was proposed by Gallagher and Yuan [10], is also an inductive approach. However, it is not based on a fixed set of functions. Rather, it randomly composes Gaussian curves. The overall fitness value is the maximum of all curves at a given point. One advantage of the GLG is the ability to control the number of local optima. Thus, the complexity of the resulting test function instances can be controlled. Also, the randomized process allows for a large variety of test functions. However, the relevance of the resulting functions is debatable. Similar test function generators are described in [1].

The *Krigifier* approach creates random Kriging models, or Gaussian processes [29]. With a user-specified trend and covariance structure, the Krigifier randomly creates a process that can be used as a non-linear test function. Thus, varied and difficult functions can be generated based on an inductive approach. The relevance of the resulting functions relies on the assumption that real-world processes are also Gaussian, but it is unclear how the resulting test functions relate to a specific real-world application.

A deductive approach has been employed for a practical application by Rudolph et al. [25]. *Deductive* approaches take a complex data set and extract important features using data-driven methods. Rudolph et al. aim to improve algorithm performance on the real problem (optimization of a ship propulsion system) by tuning performance on a Kriging surrogate model.

Bartz-Beielstein [2] proposed a deductive approach for optimization benchmarks in general. Data from a real-world system are used to train a model and for generating test function instances. Model parameters can be stochastically varied to enable diversity. Statistical tools such as mixed models are also discussed [4].

Similarly, Flasch [8] and Fischbach et al. [7] used a deductive approach based on Kriging models to generate test functions. Firstly, real-world data is taken from some experiment. Secondly, a Kriging model is trained with the data. The Kriging model is varied by making controlled changes to the model parameters, e.g., the nugget parameter or parameters of the correlation function. Hence, it will be referred to as the *parameter-variation* approach. Then, the predictor of the varied Kriging model can be used as a test function. In principle, this approach can be applied to arbitrary models and it is not restricted to Kriging or Gaussian processes. An extension by Fischbach et al. [7] takes two problems into account: (i) if a model is insensitive to some parameter, the derived test function instances will be nearly identical and (ii) if a parameter has a drastic impact, a random change may create a function without any resemblance of the original function. Both problems are handled by computing various measures of (dis-)similarity between the test function instances and the unmodified model. The computed values are required to be within user-specified bounds. Thus, simple copies (too similar) and strong distortions (not similar enough) are avoided.

Based on these results, we propose a new deductive simulation approach. The goal is to generate data-driven test functions that fulfill criteria (C.1) to (C.5) and avoid the pitfall of smoothing (C.6). The corresponding methods and required foundations are introduced in the following.

3 SIMULATION-BASED TEST FUNCTION GENERATOR

3.1 Kriging Estimation

Kriging is a modeling procedure that understands observations as realizations of a Gaussian process. A detailed description is given by Forrester et al. [9]. In optimization, Kriging is a popular choice, as it additionally provides an estimate of prediction uncertainty, which can be used to balance exploration and exploitation by computing the expected improvement of candidate solutions [19]. This approach is most famously employed in the Efficient Global Optimization algorithm (EGO) [16]. Our study utilizes Kriging to simulate responses of a Gaussian process.

Kriging approximates the data by modeling the correlation between observations, e.g., using a Gaussian correlation function (kernel) $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$, where $x, x' \in \mathcal{X}$. Here, $\mathcal{X}$ is some non-empty set, called the search space. If $\mathcal{X} = \mathbb{R}^r$ then x is a r-dimensional real vector. Furthermore, $\theta \in \mathbb{R}$ is a kernel parameter and $\mathrm{d}(x, x')$ is a distance function, e.g., $\mathrm{d}(x, x') = |x - x'|$ with $x \in \mathbb{R}$. Based on this, a correlation matrix $\boldsymbol{K}$ is computed, which collects all pairwise correlations of the training data $\boldsymbol{X} = \{x_1, ..., x_n\}$. It is used in the predictor as follows:

$$\hat{y}(x) = \hat{\mu} + \boldsymbol{k}^T \boldsymbol{K}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}), \quad (1)$$

where $\mathbf{y}$ are the training observations, $\hat{y}(x)$ is the predicted function value of a new sample x, $\hat{\mu}$ represents the process mean, $\mathbf{1}$ is a vector of ones and $\boldsymbol{k}$ is the column vector of correlations between the set of training samples $\boldsymbol{X}$ and the new sample x. All parameters (e.g., θ, $\hat{\mu}$) are determined by Maximum Likelihood Estimation (MLE).

3.2 Kriging Simulation

The predictor in Eq. (1) *estimates* a function value at a new sample x. The goal of *estimation* (or prediction) is to produce values that are as close to the true values as possible. In contrast, the goal of *simulation* is to produce values whose moments are as close to the moments of the real data as possible [17]. In case of Kriging, the simulation approach creates realizations of a Gaussian process with the same mean and covariances as the modeled process.

While the predictor is also based on the process mean and covariance matrix estimated during model training, the predictor itself does *not* have these very same properties: it smooths the data and may even be non-stationary [5]. On the other hand, simulations actually have the respective mean and covariances. This is important when generating test functions: a smoothed landscape may obviously lack important features of the real, original fitness landscape, e.g., it may have a smaller number of local optima.

Different approaches to simulation of Gaussian processes exist [5]. Our approach is based on the square root of the covariance matrix [5]. This choice is made for computational reasons. Firstly, a set $\boldsymbol{X}_s$ of m samples is selected. The process will be simulated at these samples. Secondly, the correlation matrix $\boldsymbol{K}_s$ of the set $\boldsymbol{X}_s$ is computed. It is decomposed as $\sigma^2 \boldsymbol{K}_s = \boldsymbol{C}_s = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where the process variance σ^2 is determined by MLE and $\boldsymbol{C}_s$ is the covariance matrix with the eigenvector matrix $\boldsymbol{U}$ and diagonal eigenvalue matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$. The square root of $\boldsymbol{C}_s$ is $\boldsymbol{C}_s^{1/2} = \boldsymbol{U}\mathrm{diag}(\lambda_1^{1/2}, ..., \lambda_m^{1/2})\boldsymbol{U}^T$ and the simulation is given by

$$\hat{\mathbf{y}}_{nc} = \mathbf{1}\hat{\mu} + \boldsymbol{C}_s^{1/2}\boldsymbol{\epsilon}. \quad (2)$$

Here, $\boldsymbol{\epsilon}$ is a vector of m independent, normally distributed random numbers with zero mean and unit variance.

3.3 Kriging Conditional Simulation

The goal of conditional simulation is reproducing the moments of the training data. At the same time, the simulation can be made conditional on the training data. That means, the training data is reproduced exactly by the simulation, i.e., $\hat{\mathbf{y}}_s = \mathbf{y}$ if $\boldsymbol{X}_s = \boldsymbol{X}$. Thus, the conditional simulation is a better approximation of the training data, compared to the non-conditional simulation. Still, due to the different goals of estimation and simulation, the conditional simulation error is twice as large as the estimation error [5, 17].

At a first glance, this feature may seem undesirable. The usefulness of conditional simulation can be demonstrated by a simple example given by Lantuejoul [18]. The curves in Fig. 1 represent depth measurements along an undersea cable. The interpolation (thin black line) gives a good estimate of the true depth. The conditional simulation (dashed red line) on the other hand may have a large error. However, if the goal is to determine the required length

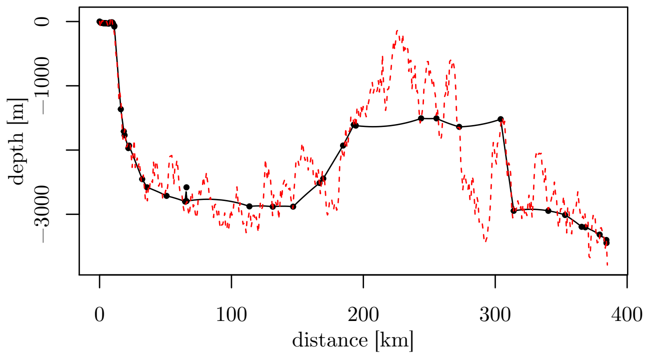

Figure 1: Undersea cable depth estimation (black line) and conditional simulation (dashed red line) and the given data (black dots). Based on the example presented in [18].

of the cable, the estimation approach may severely underestimate the true value, while the simulation may work well. Interestingly, both estimation and simulation are a result of the same trained model, but represent different features.

The conditional simulation may result in more realistic shapes than the predictor. For example, higher frequency behavior may not be visible in the predictor, but may be visible in a (conditional) simulation [5]. Similar to the non-conditional simulation, different conditional simulation approaches exist. Here, we use a straight forward approach that directly simulates the conditional Gaussian process [28]. This is not necessarily the most efficient choice. But it allows for a rather simple and transparent implementation. A more advanced approach can be substituted when needed, e.g., when growing data size renders the following approach infeasible.

As described in Sec. 3.2, we have the correlation matrix of the training data (observed data) $\boldsymbol{K}$ and the matrix of correlations between the samples to be simulated, $\boldsymbol{K}_s$. Furthermore, $\boldsymbol{K}_x$ denotes the matrix of the cross-correlations between training and simulation samples. Correlations of the combined training and simulation samples can be arranged in a block matrix as follows:

$$\boldsymbol{K}_{all} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_x^T \\ \boldsymbol{K}_x & \boldsymbol{K}_s \end{bmatrix}.$$

Following [28], the conditioned correlation matrix can be calculated as $\boldsymbol{K}_{cs} = \boldsymbol{K}_s - \boldsymbol{K}_x \boldsymbol{K}^{-1} \boldsymbol{K}_x^T$. Intuitively, identical training and simulation data results in a zero correlation matrix $\boldsymbol{K}_{cs}$, which follows from $\boldsymbol{K}_s = \boldsymbol{K}_x = \boldsymbol{K}$. To simulate the conditional process, $\boldsymbol{C}_{cs} = \sigma^2 \boldsymbol{K}_{cs}$ is used in a similar manner as in (2), with

$$\hat{\mathbf{y}}_c = \hat{\mathbf{y}} + \boldsymbol{C}_{cs}^{1/2}\boldsymbol{\epsilon}. \quad (3)$$

The estimations $\hat{\mathbf{y}}$ of the simulation samples are derived with Eq. (1).

3.4 Test Function Generator

The main proposal of this work is to use the (conditional) simulation approach to generate test functions as described in Algorithm 1. The function generator first creates or receives data of the problem (line 2-7). Then, a Gaussian process model is trained with that data (line 8) and simulation samples are created (line 9). For each desired test-function, a separate simulation is performed (line 11).

Algorithm 1 Simulation-based test function generation

```
1:  Given: number of training samples $n$, simulation samples $m$
    (usually $m \gg n$), required test functions $n_{sim}$ and (optionally)
    the expensive real-world objective function f(x).
2:  if $f(x)$ is available then
3:      Create $n$ samples $\boldsymbol{X} = \{x_1, ..., x_n\}$.
4:      Determine observations $\mathbf{y}$: $y_i = f(x_i)$.
5:  else
6:      User provides data set $\{\boldsymbol{X}, \mathbf{y}\}$.
7:  end if
8:  Train Gaussian process model $M$ based on $\{\boldsymbol{X}, \mathbf{y}\}$.
9:  Create $m$ samples $\boldsymbol{X}_s = \{x_1, ..., x_m\}$.
10: for all j $\in 1, ..., n_{sim}$ do
11:     Create (non-)conditional simulations $\hat{\mathbf{y}}_s^{(j)}$ with Eq. (2) or (3).
12:     if $\boldsymbol{X}_s = \mathcal{X}$ then
13:         Simulation $\hat{\mathbf{y}}_s^{(j)}$ is the required $j$-th test function.
14:     else
15:         Provide $j$-th test function as interpolation of simulated
            samples using Eq. (1): $\hat{y}_s^{(j)}(x) = \hat{\mu} + \boldsymbol{k}_s^T \boldsymbol{K}_s^{-1}(\hat{\mathbf{y}}_s^{(j)} - \mathbf{1}\hat{\mu})$.
16:     end if
17: end for
```

If the simulation covers the whole search space, the resulting values already represent the test function. If not, an interpolation step (line 15 in Algorithm 1) is necessary. The chosen simulation approach only simulates at the given sample location $\boldsymbol{X}_s$, and does not present an explicit formula. Hence, the interpolation is necessary when only a subset of the search space is simulated. To guarantee that the interpolation step actually reproduces the training data in the conditional simulation case, it is useful to ensure that $\boldsymbol{X} \subset \boldsymbol{X}_s$. The interpolation step has to be used with care. As this step is based on estimation, it may violate the non-smoothing criterion (C.6). However, since m simulation samples are interpolated, rather than just the n training samples, this issue is less severe than in the simple estimation case. The advantages of this test function generator are:

- It can make use of real-world data and does not require access to the actual objective function, providing access to inexpensive test functions (C.5).
- Test functions represent a problem class rather than a single problem. Diverse test functions can be produced at random (C.2).
- Test functions can reproduce the behavior of real-world problems (non-conditional simulation) *and* the underlying data (conditional simulation), thus satisfying the relevance criterion (C.4).
- Estimation-based test functions do not necessarily provide the most realistic landscape. E.g., higher frequency behavior may be ignored by the predictor. Contrarily, simulation allows to respect such behavior [5]. Thus, simulation may avoid the main pitfall of data-driven test function generation and satisfies the non-smoothing criterion (C.6). Since this avoids overly simplified test problems, this also helps to meet the difficulty (C.1) and relevance (C.4) criteria.
- Kriging models are very flexible (C.3). By adapting the kernel function (or its parameters) most problems can be approximated. Due to this feature, the simulation approach is even independent of the solution representation (data type of x) [20, 32].
- Unlike the parameter variation approach (cf. Sec. 2), the simulation approach is a more principled way of generating diverse test functions (C.2). In the parameter-variation approach, controlled changes to the parameters may have drastic effects on the resulting functions. Simulation, and especially conditional simulation, on the other hand guarantees that certain structures of the real-world data are preserved in the test function.
- Unlike the Krigifier approach, we propose to use data from real-world problems to derive the simulations (C.4). Furthermore, we outline the difficulties of excessive smoothing (C.6), which also affect the Krigifier approach. In addition, conditional simulation is used, which has not been explored by the Krigifier.

Disadvantages or possible problems are:

- The training data may introduce bias. If insufficient data is collected, the model may not learn the problem structure.
- The model selection and configuration may also introduce bias. It may be unreasonable to compare certain surrogate models or their configurations based on this procedure: The models that use the same configuration and type as the simulation model would have an unfair advantage.
- The number, value and location of local and global optima is unknown. This is in contrast to classical test functions, BBOB or the GLG. If required, such features have to be approximated.
- The number of simulation samples m is important to be set to a good value. Very large values of m may lead to computational issues due to memory and time requirements. Rather small values of m may lead to excessive smoothness, due to the final interpolation step. Importantly, this smoothness issue is less severe than in the estimation case: m is not restricted by the cost of evaluation of $f(x)$ (unlike the number of training samples n).
- Conditional simulation may produce test functions that have little variation if the trained Kriging model fits the data exceedingly well, thus violating the diversity criterion (C.2). The model estimates low variances and all realizations of the simulation will be nearly identical. To detect such a case, the estimated variances at the simulation sample locations can be compared against a threshold value. In many use-cases, sparsity of training data due to high costs of evaluation will render this issue unlikely. This issue is irrelevant for non-conditional simulation.

4 ONE-DIMENSIONAL EXAMPLE

To demonstrate the intuition behind the simulation-based test function generator, we first present a simple example. The source code can be requested from the authors. The example is based on the real valued, one-dimensional function

$$f_{1dim}(x) = \exp(-20x) + \sin(6x^2) + x, \qquad (4)$$

with $x \in [0, 1]$. Here, $n = 5$ samples are created with uniform random sampling and are evaluated with f_{1dim}. The Kriging model is trained with the data and is simulated at $m = 100$ locations. The simulation samples include the five training samples. The remaining $m - n = 95$ samples are drawn from a uniform random distribution. This process is repeated $n_{sim} = 10$ times, so that ten test functions are created. Figure 2 shows the objective function, the Kriging estimation and the (non-)conditional simulations.

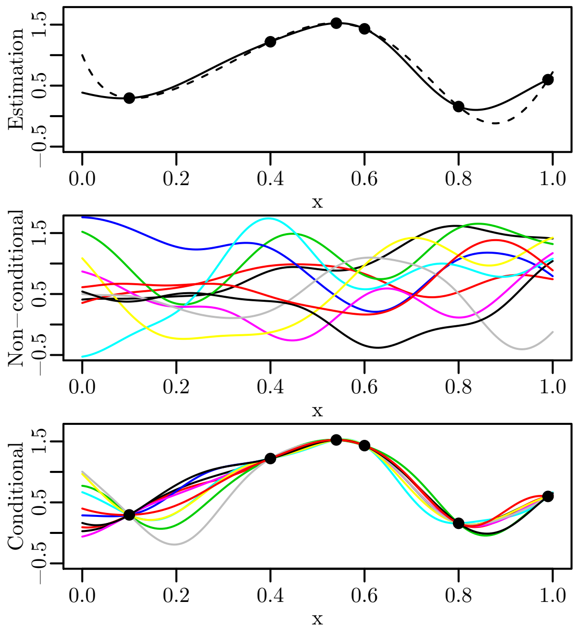

Figure 2: Top: The function $f_{1dim}(x)$ (dashed, black) and the Kriging estimation (solid, black) based on training data (dots). Middle and bottom: 10 realizations of the (conditional) simulations.

The non-conditional simulation test functions do not reproduce the training data. While the resulting functions look rather chaotic, they all share the same covariance structure and hence have similar smoothness, as well as a similar number of optima. The test functions are of similar difficulty as $f_{1dim}(x)$ (C.1), are diverse (C.2), flexible (C.3), relevant to the original problem (C.4), inexpensive (C.5), and sufficiently rugged (C.6). The motivation for using these kinds of test functions would be to test performance on functions that have similar structure as the real objective function, but are not necessarily identical to it.

Contrarily, the training data is reproduced by the conditional simulation test functions. The conditional simulation's deviation from the estimation increases with increasing distance to observed data. In general, the functions are less varied than the ones based on non-conditional simulation, but match the true function $f_{1dim}(x)$ more closely. Hence, the motivation to use non-conditional simulation would be to estimate performance on potential realizations of the same (black-box) $f_{1dim}(x)$.

5 PROTEIN LANDSCAPE APPLICATION

5.1 Data and Problem

To showcase the application of simulation-based test function generation in practice, this section presents a real-world example. To that end, an openly available data set from the field of computational biology is used [3, 23]. It contains the corresponding fitness values of all DNA sequences of length ten. Here, fitness refers to the affinity to a fluorescent target protein: allophycocyanin. The data set has previously been used for the assessment of evolutionary algorithms, using a finite state machine model [24].

Candidate solutions x are DNA sequences with $d = 10$ bases, i.e., strings with ten letters that are either `A`, `C`, `T`, or `G`. The fitness $f_{affinity}(x)$ is the result of the complex measurements described in [23], and is part of the data set. It has to be maximized.

5.2 Test Function Generation

The data set comprises all possible 10-base sequences. For our tests, $n = 100$ sequences x are selected (randomly, uniformly) and the corresponding fitness values are taken from the data set. The model M_{complete} is trained with the resulting data and is simulated at $m = 1000$ additional samples, and $n_{sim} = 10$ test functions are created. Since the complete data set is available, there is little motivation for a test function based on conditional simulation. Hence, we use the non-conditional simulation approach. The idea is to create test functions that show similar behavior as the given protein fitness landscape. Since m is smaller than the size of the complete search space, this requires to use the interpolation step during evaluation of the test function. The derived test functions are denoted with `sim(`$M_{complete}$`,interpolate)`. Here, the first argument refers to the employed model and the second argument specifies that we interpolate between the simulated samples.

To show the effect that the interpolation has on the fitness landscape, we will create two additional sets of simulation-based test functions. In both of these cases, the search space is restricted to a subspace of just 1024 sequences. To that end, the last 5 elements of each sequence are fixed to `ACGTA`. In the first case, the above described model M_{complete} is used to simulate all 1024 sequences in the subspace. This case will be denoted `sim(`$M_{complete}$, `simulate-only)`. In the second case, a new model M_{subspace} is trained with 100 sequences selected (randomly, uniformly) from the 1024 sequences of the subspace. This case will be denoted `sim(` $M_{subspace}$`,simulate-only)`.

Since the candidate solutions or samples are not real-valued, the correlation function described in Sec. 3.1 can not be used. Instead, the correlation function has to be changed [20, 32]. Here, the exponential correlation function $\mathrm{k}(x, x') = \exp(-\theta \mathrm{d}(x, x'))$ is used with the Hamming distance

$$\mathrm{d}(x, x') = \sum_{i=1}^{d} w_i \text{ with } w_i = \begin{cases} 1 & \text{if } x_i \neq x'_i, \\ 0 & \text{otherwise.} \end{cases}$$

The Hamming distance also proved to yield good results in other studies [31] and has the additional advantage of low computational cost. The Hamming distance was also used in the original study that introduced the considered data set [23].

Since the problem is discrete and of rather manageable size, we can use brute force to estimate the global optimum of each generated test function. Also, the number of local minima is determined, i.e., the number of samples whose neighbors do not have a better fitness. Here, Hamming neighborhood is employed. That means, the neighbors of a sequence are all sequences that differ in exactly one element from the original one.

5.3 Landscape Analysis

Firstly, we report some landscape characteristics of the test functions based on simulations in the complete, unrestricted search space, i.e., sim($M_{complete}$,interpolate). Rowe et al. [23] report a correlation length of roughly 4.5. They estimate this value by calculating the auto-correlation of random walks in the fitness landscape. Their result is nicely reproduced by the Kriging model, which is trained with just 100 samples: the correlation length (the reciprocal of the kernel parameter θ) determined by maximum likelihood estimation during model training is 4.48. Another good match is the reported fitness distance correlation: -0.32 (in [23]) versus -0.37 with standard deviation 0.09 (in this study).

Unfortunately, there is also a strong mismatch. Rowe et al. [23] report that the data set has 6805 local optima. The simulated test functions sim($M_{complete}$,interpolate) have 49 local optima or less. This is a major problem, as the test functions do not seem to represent the underlying problem very well. The test functions would be far to easy to solve, violating criteria C.1, C.4, and C.6.

This problem can clearly be linked to the last step of the function generation: interpolation. The interpolation is again based on estimation. It can not represent higher frequency changes in the landscape very well, as it introduces too much smoothness. Essentially, the interpolation between simulated samples will remove potential local optima which would otherwise be present in a simulation of the complete search space. Since the number of simulated samples $m = 1000$ is much smaller than the number of local optima in the real landscape, the resulting test function is necessarily much smoother than the real problem.

This problem can be revealed in a down-sized experiment. By restricting the analysis to a subspace of just 1024 DNA sequences, i.e., sim($M_{complete}$,simulate-only), interpolation can be avoided. This results into landscapes that have between 2 and 7 local optima (in this subspace). This result matches more closely to the real landscape's behavior, which should on average have $6805/4^5 \approx 6.6$ local optima in a subspace of this size.

If the model is also directly trained as well as simulated in the subspace, i.e., sim($M_{subspace}$,simulate-only), this results into 10 to 19 local optima. The real landscape has exactly 16 optima in the subspace. The earlier violated criteria (C.1, C.4, C.6) are satisfied. In theory, we could do the same pure simulation experiment on the complete search space, but that is computationally infeasible: even just storing the required 4^{10} x 4^{10} covariance matrix is prohibitive.

Clearly, this is a central issue. As shown, small discrete search spaces may allow to skip the interpolation step and the corresponding problems. Large, rugged search spaces remain a challenge. We do not resolve this numerical and computational issue in this article, rather point to some potential solution approaches. As noted in Sec. 3, we used rather simple and straightforward simulation techniques. There are more efficient simulation techniques that allow to deal with larger numbers of simulation samples. In the continuous case, one could try to adopt the spectral simulation technique that does not directly rely on a set simulated samples but is based on a sum of cosine functions [5]. Spectral simulation does not require to compute the complete covariance matrix for the simulation samples. In the discrete case, Gaussian Markov Random Field models [26] may be of interest. Here, the Markov property induces sparsity in the inverse of the covariance matrix, which may be exploited to deal with large sample sizes.

As the one dimensional example in Sec. 4 showed, interpolating a small number of simulation samples should provide satisfying results if the problem itself is rather smooth. Hence, it is desirable to estimate the required number of simulation samples m that lead to an interpolation that reflects the ruggedness of the actual problem. Clearly, the model is fully specified once all parameters are determined. It should be possible to estimate m based on the resulting covariance structure. In that sense, small parameters θ of the kernel function yield smoother landscapes that require less simulation samples to approximate. Large θ yield more rugged landscapes that require more simulation samples. One could also take an empirical approach to determine m, by increasing it in steps and observing the convergence of a suitable measure (e.g., some non-parametric measure of ruggedness of the simulation). Where possible, expert knowledge about the problem may also help to determine a suitable m, if, e.g., the number of local optima is known. Large, rugged search spaces remain a challenge, and should receive more attention in future research.

At the same time, the experimental results demonstrate that simulation-based test functions should be preferred to pure estimation. The estimation of the model M_{subspace} trained in the 5-base subspace has only 2 optima (in that same subspace). This stresses that excessive smoothing may suppress local optima. This problem extends to any kind of estimation-based test function generator.

5.4 Performance Analysis

As the landscape analysis showed, the interpolated simulation is not a good representation of the real problem behavior. Hence, we use test functions derived from a model M_{subspace} that is trained and simulated in the earlier introduced 5-base subspace, i.e., sim($M_{subspace}$,simulate-only). By restricting the performance analysis to the 5-base subspace, we avoid the interpolation problem. We use the derived simulation-based test functions to evaluate the performance of optimization algorithms. Ten different test functions are created, and each algorithm is run twenty times on each function, resulting into 200 replications.

In addition, we want to show the advantage of this approach in comparison to an estimation-based test function. Therefore, a baseline test function is derived from the estimation (prediction) of the same model. Finally, the algorithms tests were repeated on the actual objective function, i.e., directly using the real data. Since these last two cases only involve a single objective function instance, all 200 replications were spent on that single function.

The tested optimization algorithm is a variant of EGO for combinatorial optimization [16, 31]. The algorithm first generates a set of k samples (randomly, uniformly) and evaluates them with the objective function (or test function). Different values of the initial design size parameter k will be tested: k={5,10,20,50}. A Kriging surrogate model is learned with the resulting data. An optimization algorithm (here: brute force[1]) is used to optimize an infill criterion. We compare two infill criteria: expected improvement (EI) and the predicted mean. The former is, e.g., described in [16]. The latter is

[1] In larger search spaces, evolution strategies or related methods are more appropriate.

the derived from Eq. (1). The sample that optimizes the infill criterion will be evaluated by the objective function and the result is used to update the surrogate model. This procedure is iterated until a budget of 100 function evaluations is exhausted. Random search is used as a baseline comparison. Hence, we compare 9 algorithms: Random search and 8 combinations of the infill criterion and k.

Similarly to the COCO framework [11], we use a set of target values and the respective runtime required to reach these targets to measure algorithm performance. To that end, the global optimum $y_{opt} = f(x_{opt})$ is determined by brute force. Based on the determined optimum, the fitness gap is specified as follows $\text{fgap}(x) = f(x) - f(x_{opt})$. Finally, evenly spaced targets for fgap are specified on a logarithmic scale: $\text{tar}_{fgap} = 10^0, 10^{-0.2}, \ldots, 10^{-6}$. For each algorithm run, it is recorded after what runtime a certain target is reached. For each set of test functions, we aggregate the resulting data by calculating the fraction of all targets reached at a certain runtime. The resulting curves (fraction of targets reached against runtime) are called run length distributions [14], empirical cumulative distribution functions (ECDF) [11] or data profiles [22].

For our experiments, the ECDF plots are depicted in Fig. 3. The results show that using the EI infill criterion is crucial. Without EI, algorithm performance is close to the random search baseline. Results after 100 evaluations seem to be insensitive to k. Earlier in the runs, smaller design sizes yield superior results. The strong positive effect of EI can be partly attributed to the ruggedness and relatively large number of local optima. Since the estimation-based test function is overly smooth, this effect is less visible here: the runs without EI are more easily distinguished from the random search baseline in case of estimation. This stresses the importance of using simulation rather than estimation for testing. Compared to estimation, the simulation-based results are much closer to the results on the true function. The estimation-based test function severely underestimates the difficulty of the problem. This stresses that simulation approaches are more suited to satisfy the required criteria of test function generators, especially with regards to difficulty (C.1), relevance (C.4) and non-smoothness (C.6).

A slight difference between simulation-based tests and the real function performance can be observed. Note, that the goal of the simulation-based test functions was not to produce exact surrogate functions for the true, real-world problem. Rather, it was desired to create test function instances with similar behavior. Still, the selected kernel of the Kriging model may be improved. The isotropic nature of the kernel may not be a perfect choice. Rowe et al. [23] report that bases at the start of a sequence have larger impacts than the bases later in the sequence. It may be more exact to use an anisotropic kernel, which allows to learn the importance of each base. This comes at the cost of introducing additional kernel parameters.

6 DISCUSSION

One remaining question is: When should we use which test function generation approach? The answer clearly depends on the context and goals of the analysis.

We strongly recommend to use a simulation approach if data-driven, model-based test functions are desired. Simulation is a principled way of generating diversity and avoiding too much

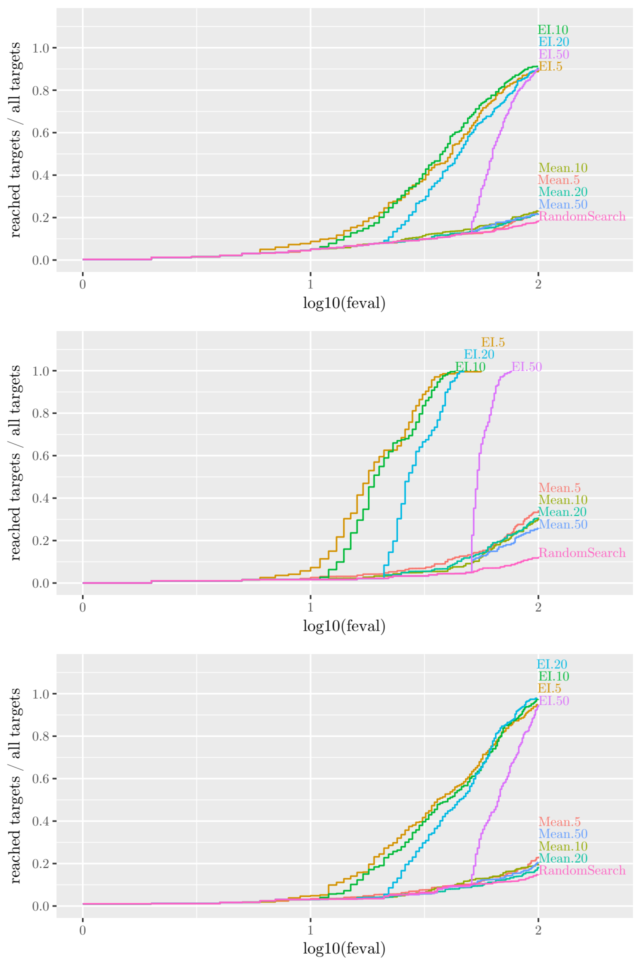

Figure 3: Logarithmic ECDF plots for three test cases: test functions based on non-conditional simulation (top), estimation (middle), and the real objective function (bottom). The labels inside the plot indicate the configuration of the employed algorithm, that is, whether EI or the predicted mean was used as an infill criterion and the size of the initial design. The x-axis depicts the logarithm of the number of fitness function evaluations (feval).

smoothness. But simulation-based test functions are not supposed to replace classical test function sets. These test function sets do have merits, e.g., their properties and behavior are well understood. If an algorithm is assessed without any specific application in mind, a mix of both would be ideal. If an algorithm is assessed with the desire to determine performance on problems with specific features (e.g., separability, unimodality), classical test functions are probably preferable. Contrarily, if an algorithm is assessed in the context of a specific real-world application (i.e., C.4 is important), a simulation-based test function generator should be preferred.

In the latter case, if performance on a class of *problems with similar behavior* as the real objective function is of interest, non-conditional simulation would be more appropriate. Conditional simulation would be more appropriate if the performance on *potential realizations of the same problem* is of interest.

7 SUMMARY AND OUTLOOK

The main research question raised in this study was: *How to generate test functions that satisfy criteria (C.1)-(C.6)?* We showed how Kriging simulation can be used to generate test functions that emulate the behavior of real-world optimization problems. The simulation-based approach allows to generate difficult (C.1), diverse (C.2), flexible (C.3), relevant (C.4), inexpensive (C.5) test problems that may avoid detrimental smoothing (C.6).

A simple example was used to illustrate the idea and a protein sequence data set was employed to demonstrate the application to a complex, real-world problem. The protein sequence experiments show that the simulation quality depends on the number of samples that are simulated. If few samples are simulated in a large, rugged search space, the resulting test functions may not respect the behavior of the real-world problem. Especially, the number of local optima may be underestimated due to an overly smooth test function. This applies even more strongly to data-driven function generators that use estimation instead of simulation. Thus, while not without hazards, simulation does present a clear advantage over estimation-based approaches.

Future research should focus on the main problem of the simulation-based function generator: dealing with large rugged search spaces. That is, simulation methods for a large number of simulation samples should be investigated. Furthermore, an estimate of the required number of simulation samples is desirable.

Finally, simulation is not limited to Gaussian processes. Other kinds of stochastic processes can also be simulated and hence used to produce test functions. It may be beneficial to include different process model types into the analysis, especially to reduce potential bias introduced by a specific choice.

ACKNOWLEDGMENTS

This work is part of a project that has received funding from the *European Union's Horizon 2020 research and innovation program* under grant agreement No. 692286.

REFERENCES

[1] Julio Barrera and Carlos A. Coello Coello. 2011. *Test Function Generators for Assessing the Performance of PSO Algorithms in Multimodal Optimization.* Springer Berlin Heidelberg, Berlin, Heidelberg, 89–117. DOI:http://dx.doi.org/10.1007/978-3-642-17390-5_4

[2] Thomas Bartz-Beielstein. 2015. How to Create Generalizable Results. In *Springer Handbook of Computational Intelligence*, Janusz Kacprzyk and Witold Pedrycz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1127–1142. DOI:http://dx.doi.org/10.1007/978-3-662-43505-2_56

[3] Bioanalytical Sciences Group - Manchester University. 2011. Microarray data: Analysis of the Complete Sequence-Fitness Landscape of a DNA Aptamer. Online, last accessed 31 jan 2017. (2011). http://dbkgroup.org/directed-evolution/

[4] Marco Chiarandini and Yuri Goegebeur. 2010. Mixed Models for the Analysis of Optimization Algorithms. In *Experimental Methods for the Analysis of Optimization Algorithms*, Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss (Eds.). Springer, Germany, 225–264.

[5] Noel A.C. Cressie. 1993. *Statistics for Spatial Data.* JOHN WILEY & SONS INC.

[6] Kenneth A. De Jong. 1975. *An analysis of the behavior of a class of genetic adaptive systems.* Ph.D. Dissertation. University of Michigan.

[7] Andreas Fischbach, Martin Zaefferer, Jörg Stork, Martina Friese, and Thomas Bartz-Beielstein. 2016. From Real World Data to Test Functions. In *Proceedings. 26. Workshop Computational Intelligence*, Frank Hoffmann, Eyke Hüllermeier, and Ralf Mikut (Eds.). KIT Scientific Publishing, Dortmund, 159–177.

[8] Oliver Flasch. 2015. *A modular genetic programming system.* Ph.D. Dissertation. TU Dortmund. DOI:http://dx.doi.org/10.17877/DE290R-7807

[9] Alexander Forrester, Andras Sobester, and Andy Keane. 2008. *Engineering Design via Surrogate Modelling.* Wiley.

[10] Marcus Gallagher and Bo Yuan. 2006. A general-purpose tunable landscape generator. *IEEE Trans. on Evolutionary Computation* 10, 5 (oct 2006), 590–603.

[11] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[12] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.* Research Report RR-6829. INRIA.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning.* Springer, Berlin, Heidelberg, New York.

[14] Holger H. Hoos and Thomas Stützle. 1998. Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 238–245.

[15] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning with Applications in R* (4th ed.). Springer.

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

[17] Andre G. Journel and Charles J. Huijbregts. 1978. *Mining Geostatistics.* Academic Press.

[18] Christian Lantuéjoul. 2002. *Geostatistical Simulation - Models and Algorithms.* Springer-Verlag Berlin Heidelberg.

[19] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. 1978. *Towards Global Optimization 2.* North-Holland, Chapter The application of Bayesian methods for seeking the extremum, 117–129.

[20] Alberto Moraglio and Ahmed Kattan. 2011. Geometric Generalisation of Surrogate Model Based Optimisation to Combinatorial Spaces. In *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'11).* Springer, Berlin, Heidelberg, Germany, 142–154.

[21] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. 1981. Testing Unconstrained Optimization Software. *ACM Trans. Math. Software* 7, 1 (1981), 17–41.

[22] Jorge J. Moré and Stefan M. Wild. 2009. Benchmarking Derivative-Free Optimization Algorithms. *SIAM Journal on Optimization* 20, 1 (jan 2009), 172–191. DOI:http://dx.doi.org/10.1137/080724083

[23] William Rowe, Mark Platt, David C. Wedge, Philip J. Day, Douglas B. Kell, and Joshua Knowles. 2009. Analysis of a complete DNA-protein affinity landscape. *Journal of The Royal Society Interface* 7, 44 (jul 2009), 397–408. DOI:http://dx.doi.org/10.1098/rsif.2009.0193

[24] William Rowe, David C. Wedge, Mark Platt, Douglas B. Kell, and Joshua Knowles. 2010. Predictive models for population performance on real biological fitness landscapes. *Bioinformatics* 26, 17 (jul 2010), 2145–2152. DOI:http://dx.doi.org/10.1093/bioinformatics/btq353

[25] Günter Rudolph, Mike Preuss, and Jan Quadflieg. 2009. *Two-layered surrogate modeling for tuning optimization metaheuristics.* Technical Report TR09-2-005. TU Dortmund, Dortmund, Germany. Algorithm Engineering Report.

[26] Håvard Rue and Leonhard Held. 2005. *Gaussian Markov Random Fields: Theory and Applications.* Monographs on Statistics and Applied Probability, Vol. 104. Chapman & Hall, London.

[27] Hans-Paul Schwefel. 1995. *Evolution and Optimum Seeking.* Wiley, New York NY.

[28] Jonathan R. Stroud, Michael L. Stein, and Shaun Lysen. 2016. Bayesian and Maximum Likelihood Estimation for Gaussian Processes on an Incomplete Lattice. *Journal of Computational and Graphical Statistics* (2016). DOI:http://dx.doi.org/10.1080/10618600.2016.1152970

[29] Michael W. Trosset. 1999. *The Krigifier: A Procedure for Generating Pseudorandom Nonlinear Objective Functions for Computational Experimentation.* Technical Report. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA.

[30] L. Darrel Whitley, Keith E. Mathias, Soraya Rana, and John Dzubera. 1995. Building Better Test Functions. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, Larry J. Eshelman (Ed.). Morgan Kaufmann, San Francisco CA, 239–246.

[31] Martin Zaefferer, Jörg Stork, and Thomas Bartz-Beielstein. 2014. Distance Measures for Permutations in Combinatorial Efficient Global Optimization. In *Parallel Problem Solving from Nature–PPSN XIII*, Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith (Eds.). Springer, Cham, Switzerland, 373–383.

[32] Martin Zaefferer, Jörg Stork, Martina Friese, Andreas Fischbach, Boris Naujoks, and Thomas Bartz-Beielstein. 2014. Efficient Global Optimization for Combinatorial Problems. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14).* ACM, New York, NY, USA, 871–878.