Mixed-Integer Benchmark Problems for Single- and Bi-Objective Optimization

Tea Tušar
Jožef Stefan Institute
Ljubljana, Slovenia
firstname.lastname@ijs.si

Dimo Brockhoff
Inria, CMAP, Ecole Polytechnique
IP Paris, France
firstname.lastname@inria.fr

Nikolaus Hansen
Inria, CMAP, Ecole Polytechnique
IP Paris, France
firstname.lastname@inria.fr

ABSTRACT

We introduce two suites of mixed-integer benchmark problems to be used for analyzing and comparing black-box optimization algorithms. They contain problems of diverse difficulties that are scalable in the number of decision variables. The `bbob-mixint` suite is designed by partially discretizing the established BBOB (Black-Box Optimization Benchmarking) problems. The bi-objective problems from the `bbob-biobj-mixint` suite are, on the other hand, constructed by using the `bbob-mixint` functions as their separate objectives. We explain the rationale behind our design decisions and show how to use the suites within the COCO (Comparing Continuous Optimizers) platform. Analyzing two chosen functions in more detail, we also provide some unexpected findings about their properties.

CCS CONCEPTS

• **Mathematics of computing → Nonconvex optimization**; • **Theory of computation → Mixed discrete-continuous optimization**;

KEYWORDS

mixed-integer optimization, benchmarking, test function suite, the COCO platform

ACM Reference Format:
Tea Tušar, Dimo Brockhoff, and Nikolaus Hansen. 2019. Mixed-Integer Benchmark Problems for Single- and Bi-Objective Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321868

1 INTRODUCTION

Real-world optimization problems are often mixed-integer, that is, include integer as well as continuous variables. Many examples of such problems come from mechanical engineering, where some variables can only assume a limited set of possible values, for example, categorical variables or variables describing components of standard sizes (predefined pipe diameters, steel plate thicknesses, etc.) or integer quantities (such as the number of heat exchanger tubes). Nevertheless, most research in Evolutionary Computation (EC) is devoted to either continuous or discrete optimization problems, while studies on mixed-integer problems are often limited to specific applications. Systematic research on the performance of mixed-integer algorithms is hindered by the lack of appropriate test suites that would support the laborious task of algorithm benchmarking.

This does not mean that there are no available mixed-integer benchmarks. MINLPLib [4], for example, is a well-known collection of over 1000 mixed-integer test problem instances. However, it is designed for benchmarking mathematical programming solvers and cannot be easily interfaced with black-box optimizers such as evolutionary algorithms. In the EC field, other selections of test problems that are similar in nature to MINLPLib (but much smaller) are being used [5, 14, 17, 20]. In addition, the AClib library [11] serves as a collection of benchmark problems arising from algorithm configuration tasks. Nevertheless, the general impression is that there are no 'standard' benchmarks for analyzing and comparing mixed-integer black-box optimizers and researchers seem compelled to propose new problems in order to benchmark their algorithms [14].

The mentioned works provide mere collections of problems, not actual benchmarking suites where careful consideration is given to selecting problems of diverse difficulties and varying dimensions in order to investigate algorithm performance in a methodical way. To the best of our knowledge, there have been only three systematic efforts in this direction. In [15], NK landscapes [13] were extended to support mixed (continuous, categorical and integer) variables. Next, [16] presented six mixed-integer problems constructed as discretized versions of the CEC benchmark functions [22]. Finally, [18] proposed a suite of mixed-integer benchmarks for multi-objective optimization constructed by separately defining the position and distance parameters, the shape of the front and correlations between objectives.

Our approach is similar to that from [16], however it builds upon the established `bbob` functions and is integrated within the COmparing Continuous Optimizers (COCO) platform making the benchmarks easily accessible to the research community. The idea behind COCO is explained in Section 2. Next, we present our main contributions: `bbob-mixint`, a single-objective suite with 24 mixed-integer functions, and `bbob-biobj-mixint`, a bi-objective suite with 92 mixed-integer functions. Both can be instantiated with diverse dimensions and instances. Details on the construction of the suites as well as the justifications for our design decisions are given in Section 3. In Section 4 we demonstrate the ease of use of running an optimization algorithm on the new suites. An investigation

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321868

of selected functions from both suites in Section 5 reveals some remarkable properties of the discretized functions. Finally, we show first results of running four optimization algorithms on the single-objective suite in Section 6 and provide conclusions in Section 7.

2 BACKGROUND

2.1 Problem definition

We formally define the instance γ of an m-objective mixed-integer *minimization* problem as

$$\begin{aligned} & f_1^\gamma(\mathbf{x}), \ldots, f_m^\gamma(\mathbf{x}) \\ \text{where} \quad & f_j^\gamma : \mathbb{Z}^k \times \mathbb{R}^{(n-k)} \to \mathbb{R},\ \mathbf{x} \mapsto f_j^\gamma(\mathbf{x}) \quad j = 1, \ldots, m \\ & x_i \in [x_i^{\min}, x_i^{\max}] \cap \mathbb{Z} \qquad\qquad\qquad\quad i = 1, \ldots, k \end{aligned}$$

The problem entails minimization of the given function(s) where the first k variables are integer and the last $n - k$ continuous. The continuous variables can be unconstrained or box-constrained. This problem formulation does not consider other types of constraints as they are beyond of the scope of the paper. The problem is single-objective when $m = 1$, and multi-objective when $m \geq 2$. We consider different instantiations $\gamma = 1, 2, 3, \ldots$ for the above problem [6], characterized by different $\mathbf{x}$- and f-space transformations.

2.2 COCO: The COmparing Continuous Optimizers Platform

The COmparing Continuous Optimizers (COCO) platform[1] [10] has been developed with the aim to simplify the benchmarking of numerical black-box optimization algorithms and to provide data to the scientific community from those benchmarking experiments. COCO is composed of two main parts: the first part aims at collecting benchmarking data by performing a numerical experiment. The user can plug in and run an optimization algorithm in either C/C++, Java, Matlab/Octave, or Python. Performance data, consisting in the number of function calls to solve a set of test functions (belonging to some predefined benchmark suite) to a set of given precisions, are then automatically collected and written to file. A lot of effort was spent to avoid the most-common pitfalls of benchmarking experiments [10] by providing a predefined benchmarking methodology on (sets of) well-chosen and -understood functions.

The second part of the platform is used for postprocessing the produced benchmarking data and is written in Python. It provides various graphical outputs and tables as HTML pages and LaTeX templates to easily compare algorithm performances. So far, experimental data for 200+ algorithms or algorithm variants have been made available through the postprocessing part of COCO. The majority of those 200+ data sets have been collected for the 24 single-objective, unconstrained functions of the `bbob` test suite, but data for the noisy unconstrained `bbob-noisy` and the bi-objective, unconstrained `bbob-biobj` test suites are available as well. The newest addition to COCO is the `bbob-largescale` test suite for large-scale optimization [24].

What has been missing so far, is a test suite with mixed-integer problems. We therefore implemented the proposed `bbob-mixint` and `bbob-biobj-mixint` test suites in COCO and will showcase their properties and how certain mixed-integer (blackbox) algorithms optimize them.

[1] https://github.com/numbbo/coco

2.3 Visualizing Algorithm Performance

In COCO, test functions are parameterized functions that depend on the dimension and an instance identifier. Transformations of "raw" functions allow to create instances of similar difficulty. The combination of a function id, instance id, and dimension finally results in a concrete function to optimize.

An actual problem, for which we can record how long an algorithm needs to solve it, is a combination of a function instance and a target precision—an absolute function value that is composed of the optimal function value and a relative precision such as 10^{-8}. For each algorithm run, COCO records for a set of given target precisions the number of function evaluations to reach a function value below each precision. Typically, 5–10 target values per order of magnitude are recorded of which the runtimes for a few dozen targets per function instance are displayed.

The main display of COCO are empirical cumulative distribution plots (ECDFs) of recorded runtimes. The ECDFs in COCO are extensions of the well-known data profiles [19] in which (a) data from multiple targets are aggregated instead for only a single target and (b) the measured runtimes are used to simulate runtimes of a restarted version of the algorithm to account for unsuccessful runs, see [9] for details.

In the multi-objective case, the quality of an algorithm at a given number of function evaluations is measured as the hypervolume of all non-dominated solutions found so far if the nadir point has been attained, or as the negative distance to the region between the ideal and nadir point if the nadir point has not been attained. Like in the single-objective case, COCO records the number of function evaluations to reach given hypervolume targets. These targets are defined relative to a given Pareto front approximation because the available multi-objective test suite in COCO does not have an analytic description of the Pareto front. For more details, we refer the interested reader to the documentation of COCO [3].

3 PROPOSED BENCHMARK SUITES

In this section we present two new suites of mixed-integer benchmark problems, the single-objective `bbob-mixint` suite and the bi-objective `bbob-biobj-mixint` suite.

3.1 The `bbob-mixint` Suite

The `bbob-mixint` suite is constructed by partially discretizing problems from the `bbob` [6] and `bbob-largescale` [24] suites. In the following, we first explain how the discretization is performed, then describe the construction of the suite and finally show how the functions are scaled to adjust their difficulty.

3.1.1 Discretization. Consider a `bbob` (or `bbob-largescale`) problem with the function $f : \mathbb{R}^n \to \mathbb{R}$ and optimal value $f^{\text{opt}} = f(\mathbf{x}^{\text{opt}})$. The resulting mixed-integer function will have the form

$$\overline{f} : \mathbb{Z}^k \times \mathbb{R}^{(n-k)} \to \mathbb{R},$$

that is, it will be defined on k integer and $n-k$ continuous variables. While all bbob functions are defined for any $\mathbf{x} \in \mathbb{R}^n$, all but the linear slope function[2] have their optimal solution within $[-4,4]^n$. The partial discretization is performed in such a way that the optimal value is preserved, that is $\overline{f}^{\text{opt}} = f^{\text{opt}}$.

Now let us assume that we wish to discretize the variable x_i, where $i \in \{1,\ldots,k\}$, into the set $\{0,\ldots,l-1\}$ of l integer values. This discretization is done as follows:

(1) First, we define a sequence of l equidistant auxiliary values $-4 < y_1 < \cdots < y_l < 4$ so that $y_{j+1} - y_j = \frac{4+4}{l+1} = y_1 - (-4) = 4 - y_l$, where $j = 1,\ldots,l-1$.
(2) We denote with y^* the value $y_j, j = 1,\ldots,l$, that is closest to x_i^{opt}. The difference between the two is represented by $d_i = y^* - x_i^{\text{opt}}$. Note that $|d_i| \leq \frac{4}{l+1}$ if $x_i^{\text{opt}} \in [-4,4]$.
(3) Then, $z_j = y_j - d_i$ for $j = 1,\ldots,l$. This aligns one of the z_j values with x_i^{opt}.
(4) Finally, the following transformation ζ is used to map any continuous value $x_i \in \mathbb{R}$ to an integer in $\{0,\ldots,l-1\}$:

$$\zeta(x_i) = \begin{cases} 0 & \text{if} \quad x_i < z_1 + \frac{4}{l+1} \\ 1 & \text{if} \quad z_1 + \frac{4}{l+1} \leq x_i < z_2 + \frac{4}{l+1} \\ \vdots & \vdots \\ l-1 & \text{if} \quad z_{l-1} + \frac{4}{l+1} \leq x_i \end{cases}$$

The values $y_j, j = 1,\ldots,l$, in Step (1) are chosen in such a way that the corresponding shifted values z_j remain within $[-4,4]$ if x_i^{opt} is also in $[-4,4]$. If not, the shift is larger, but $z_j, j = 1,\ldots,l$, never go beyond x_i^{opt}, which in practice means they remain within $[-5,5]^n$—the *region of interest* for all bbob problems.

3.1.2 Suite Construction. The bbob suite consists of problems with 24 different functions in 6 dimensions, $n = 2,3,5,10,20,40$, and 15 instances (see [6] for the function definitions). Because the discretization reduces the number of continuous variables, higher dimensions are used for the `bbob-mixint` suite to produce challenging problems. We chose $n = 5,10,20,40,80,160$ as the dimensions of the `bbob-mixint` suite.[3]

Because the numerical effort for some bbob problems scales with n^2, we use these for dimensions ≤ 40 only. For dimensions > 40, we use the corresponding problems from the `bbob-largescale` suite [24] which scale linearly with n.

As all dimensions n are multiples of 5, we define five arities for $n/5$ consecutive variables, respectively, as $l = 2,4,8,16,\infty$. We use instances 1–15 to instantiate each problem. They match the equally-numbered instances of the underlying bbob and `bbob-largescale` problems.

3.1.3 Function Scaling. Initial experiments using the algorithms Random Search, CMA-ES [8] and DE [21] (see Section 6 for more information) have shown that the new problems are of considerably different difficulties. Some are extremely hard to solve, while for others, a non-negligible percentage of targets is met already after a handful of function evaluations. Because COCO's performance assessment aggregates results over function and target pairs, we scale function values to adjust for these different difficulties.

Table 1: Factors used for scaling the `bbob-mixint` functions.

	Factor		Factor		Factor		Factor
f_1	1	f_7	1	f_{13}	0.1	f_{19}	10
f_2	10^{-3}	f_8	10^{-2}	f_{14}	1	f_{20}	0.1
f_3	0.1	f_9	10^{-2}	f_{15}	0.1	f_{21}	1
f_4	0.1	f_{10}	10^{-3}	f_{16}	1	f_{22}	1
f_5	1	f_{11}	10^{-2}	f_{17}	10	f_{23}	10
f_6	10^{-2}	f_{12}	10^{-4}	f_{18}	1	f_{24}	0.1

In order to decide on the scaling factors, we look at how many targets can be reached just by evaluating the domain middle (often the first guess of an optimization algorithm). However, because two values could be interpreted as the 'middle' value for variables of even arity, we need to sample among a large set of possible domain middle points. Figure 1 (b) shows the difference between the median f-value of 1000 domain middle samples and the f-value of the optimal solution for each problem instance in the `bbob-mixint` suite prior to scaling. In comparison, Figure 1 (a) shows the difference between the f-value of the domain middle and of the optimal solution for each problem instance for the bbob suite (note that no sampling is required here since it is clear which point is the domain middle in a continuous domain).

Keeping in mind that in COCO the easiest target defaults to 100, we see that for a number of `bbob-mixint` functions (f_2, f_6, f_{10} to f_{13} and f_{20}), the domain middle rarely (if ever) achieves this target. On the other hand, for functions such as f_{17}, f_{19} and f_{23}, evaluating the domain middle already guarantees reaching targets of 10 and less. We also see that the distances for the `bbob-mixint` suite are very similar to those for the bbob suite, albeit a bit larger in general. Based on these findings and the preliminary algorithm results, we choose to multiply the f-values of the functions with the scaling factors shown in Table 1. This setting is mindful of the connections between some functions, for example, the same scaling factors are used for the original (f_8) and rotated (f_9) Rosenbrock functions. Figure 1 (c) shows the result for all (scaled) `bbob-mixint` problems. Now the f-differences between the domain middle and the optimal solution are more uniform across the problems in the suite.

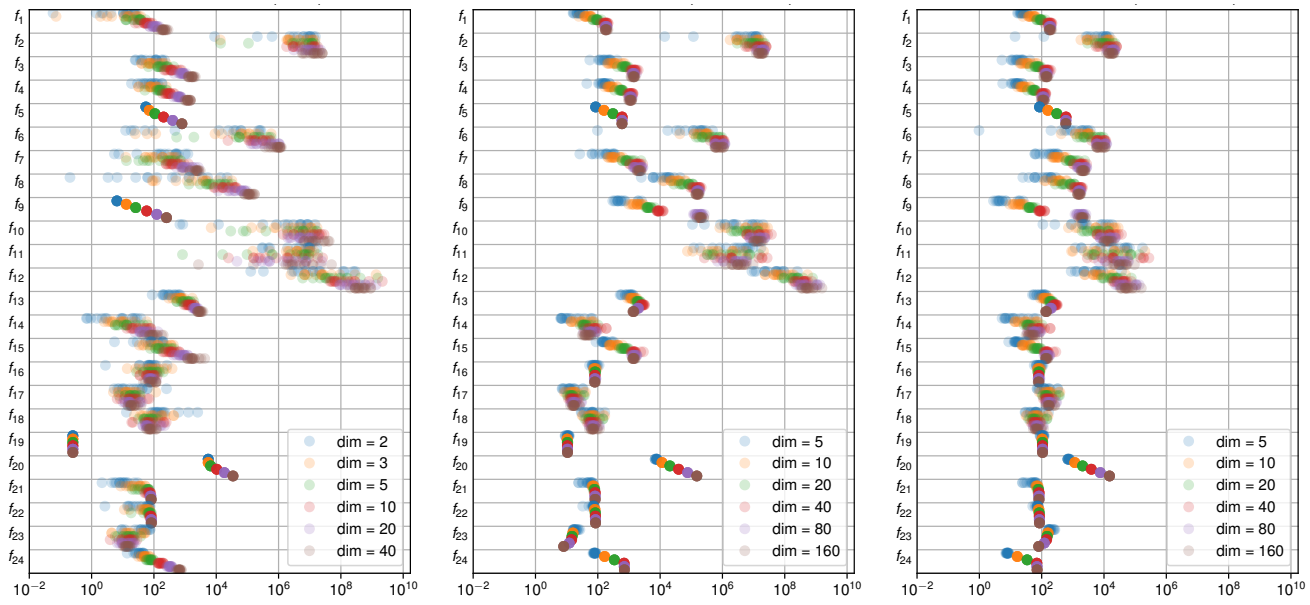
To summarize, the `bbob-mixint` suite contains mixed-integer problems constructed by discretizing the continuous problems from the bbob and `bbob-largescale` suites. Using 24 functions, 6 dimensions and 15 instances results in the total of 2160 problem instances.

3.2 The `bbob-biobj-mixint` Suite

The `bbob-biobj-mixint` suite is constructed by combining two single-objective functions from the `bbob-mixint` suite (following the idea of the `bbob-biobj` and `bbob-biobj-ext` suites [23]). Instead of making every possible bi-objective combination of two single-objective functions, which would result in a total of $\frac{24 \cdot 23}{2} = 276$ functions, we adopt the same combinations as were chosen in the `bbob-biobj-ext` suite. This gives 92 bi-objective functions (see [23] for details).

Note that, similarly as for the `bbob-biobj` and `bbob-biobj-ext` suites, the Pareto sets and fronts of these mixed-integer bi-objective

[2] The optimal solutions to the bbob linear slope function (f_5) lie at and beyond one of the corners of the subspace $[-5,5]^n$. However, this does not affect the discretization procedure.
[3] Note that the function definitions of all mentioned test suites are scalable in dimension. The six dimensions are only pre-chosen to facilitate the experimental setup.



(a) bbob suite (b) bbob-mixint suite before scaling (c) bbob-mixint suite

Figure 1: Estimation of targets reached by the domain middle (in logarithmic scale). They are computed as the distance between the f-value of the domain middle and the optimal solution for the bbob suite (a), or the median of the distances between the f-values of 1000 domain middle samples and the optimal solution for the bbob-mixint suite before (b) and after (c) scaling. Each circle depicts one problem instance for instances 1–15.

functions are generally unknown as knowing the optima to the single-objective functions only gives us information about the two extreme points of the Pareto front. The only exception is the function $F_1 = (f_1, f_1)$, the combination of two sphere functions, which is analyzed in Section 5.

In COCO, assessing the performance of algorithms on a bi-objective problem relies on knowing the hypervolume of its Pareto front. Since we cannot compute it analytically, we resort to estimate it by running several different optimization algorithms on the problem for a very large number of evaluations and computing the hypervolume of the resulting non-dominated solutions. This value is then used to set the targets for performance assessment.

```
import cocoex
import cma

suite = cocoex.Suite('bbob-mixint', '', '')
for problem in suite:
    opts = {'bounds': [problem.lower_bounds,
                       problem.upper_bounds],
            'integer_variables': list(range(
                problem.number_of_integer_variables)),
            'CMA_stds': (problem.upper_bounds
                         - problem.lower_bounds) / 5}
    cma.fmin2(problem, problem.initial_solution, 1,
              opts, restarts=9)
```

Figure 2: Python code to run the CMA-ES algorithm on all problems in the bbob-mixint suite.

4 IMPLEMENTATION IN COCO

Like other benchmark suites available in COCO, the bbob-mixint and bbob-biobj-mixint suites are implemented in C and interfaced to be used by algorithms in any of the COCO's supported languages (C/C++, Python, Java and Matlab/Octave). Figure 2 shows how easy it is to interface an optimization algorithm (in this case, the Python implementation of CMA-ES in the pycma package, [7]) with the bbob-mixint suite in Python.

After importing the two Python modules (COCO's experiments and CMA), the bbob-mixint suite is initialized with default parameters. Then, CMA-ES is run on all problems in the suite. The code snippet shows how to access the relevant information on the problem, such as the number of integer variables and the lower and upper bounds of the region of interest (the total number of variables is inferred by CMA-ES through the initial solution, but can also be obtained through problem.dimension).

The bbob-biobj-mixint suite can be employed in the same way, using a multi-objective algorithm in place of CMA-ES.

5 PROBLEM PROPERTIES

In order to investigate the properties of the proposed test problems, we visualize and discuss, in this section, level sets for the bbob-mixint functions, heat maps for the ellipsoid function, and approximations of the Pareto set and front for the bi-objective double sphere function.

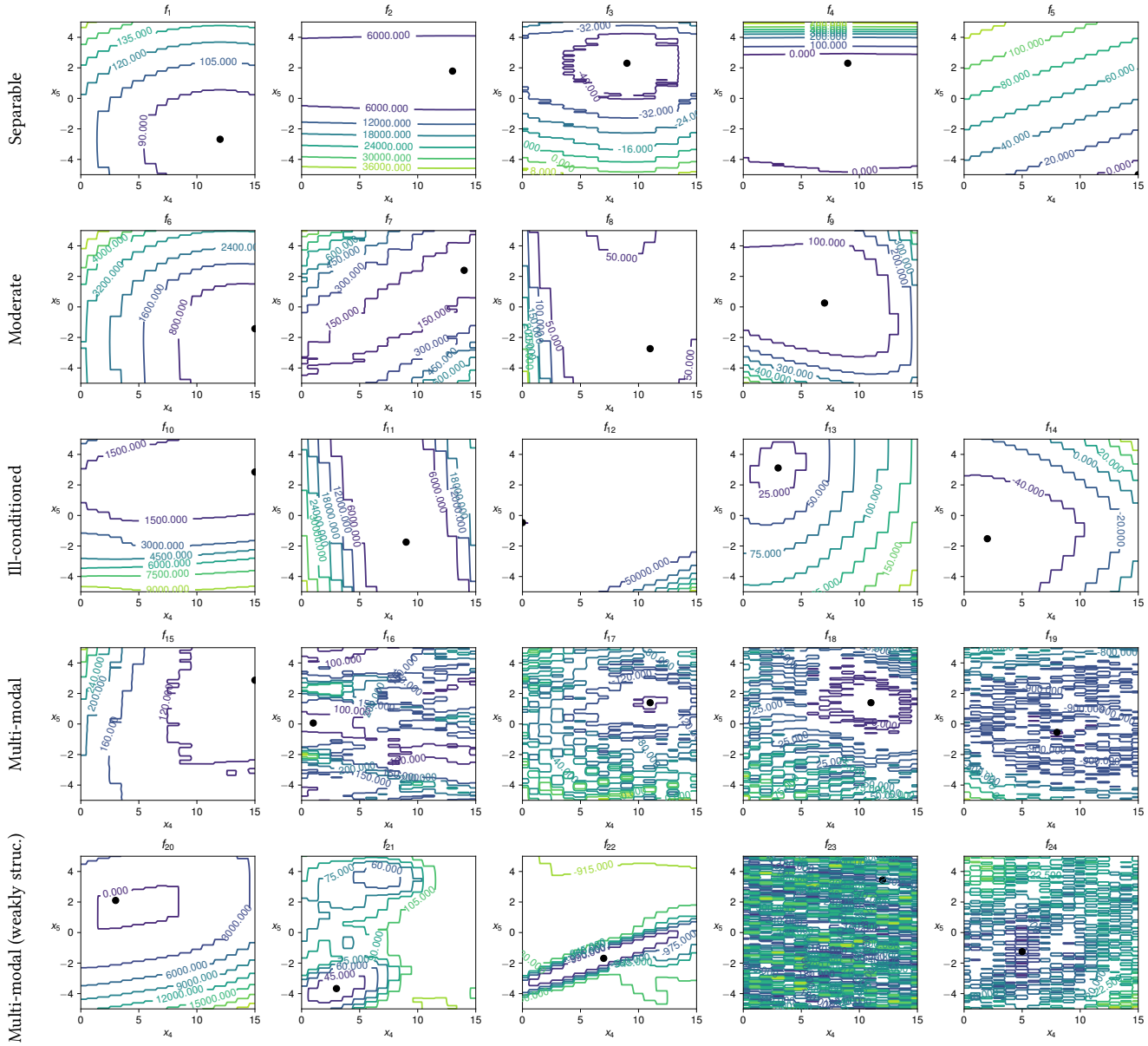


Figure 3: Level sets for the `bbob-mixint` suite problems of dimension 5, instance 1. The functions are organized by groups. The variables chosen for the x and y axis represent the last integer and first continuous variable, respectively. Lighter colors denote higher values. The black dot represents the optimal solution.

5.1 Level Sets of the `bbob-mixint` Functions

Figure 3 shows level sets of equal function value of instance 1 of all 24 `bbob-mixint` functions within the axis-parallel plane through the optimum, spanned by the last two variables (in 5-D). Note that in 5-D, only the last variable is continuous and the second-to-last one is discrete and has an arity of 16.

At first sight, the level sets show the expected behavior: compared to the level sets of the continuous versions from the bbob suite, the level sets for the `bbob-mixint` functions are piece-wise linear due to the discretization of 80% of the variables.

5.2 A Discretized Unimodal Function Can Become Multimodal

At closer inspection, however, we see how the high ill-conditioning of some of the problems together with a search space rotation, for

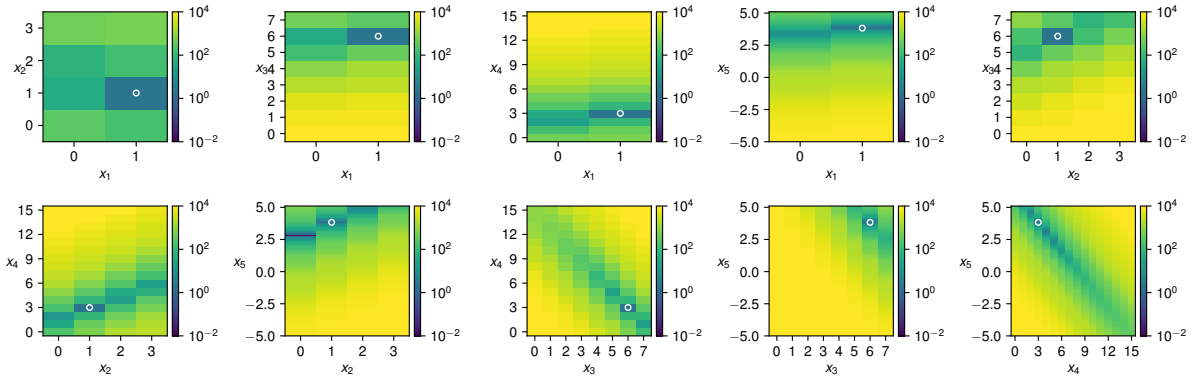


Figure 4: Heat maps of the `bbob-mixint` rotated ellipsoid function (f_{10}) in dimension 5, instance 6. The white circle indicates the solution at $(x_1^{\text{opt}}, 1, x_3^{\text{opt}}, x_4^{\text{opt}}, 3.833)$, which is a local optimum with regard to the discrete variables.

example on the ellipsoid function f_{10}, can result in a local optimum in the discretized version although the underlying continuous function is unimodal.

As an example, consider instance 6 in dimension 5 of the ellipsoid function f_{10}. Figure 4 shows heat maps in all axis-parallel planes through the search point with $x_2 = 1$, $x_5 = 3.833$ (white circle) and all other variable values set as in the global optimum. This selected search point is obviously not optimal (see the x_2-x_5 plot in Figure 4), however, changing any number of discrete values to their neighboring values results in worse function values (this cannot be seen from Figure 4, but has been verified). This situation appears if the discretization around the main axis of the high-conditioned ellipsoid is coarse enough such that better search points are only located along the diagonal of the search space but not along the coordinate axes. This is not possible if the original ill-conditioned function is axis-parallel itself (such as function f_2)—where always one of the neighboring discrete values is improving the objective function.

5.3 Pareto Sets and Fronts for the Discrete Double Sphere Function

Combining the single-objective bbob functions to form the functions of the `bbob-biobj` suite already results in complicated and unusual Pareto sets and fronts [2]. Here, we showcase the Pareto sets and fronts of the simplest `bbob-biobj-mixint` function, the discretized double sphere, in order to see which additional difficulties appear when the objective functions of a bi-objective problem are discretized. We chose this function because the underlying continuous problem is the only `bbob-biobj` function for which an analytic expression for the Pareto set is known [2].

The Pareto set of the continuous double sphere function is a straight line between the two single-objective optima. In order to visualize the discretized version of it (f_1 of the `bbob-biobj-mixint` suite), we evaluate the search points closest to this (discretized) line. The continuous variables are thereby discretized as well, using 801 points between the two extremes for each reported instance.

Figure 5 shows the resulting non-dominated points for three selected instances in dimension 5 and for two instances in dimension 10. The top row presents the projection (in decision space) onto the plane spanned by the two variables with the largest number of distinct values (within the set of non-dominated solutions). The middle row shows the two first principal components of a Principal Component Analysis (PCA) on those non-dominated solutions, while the bottom row displays the corresponding objective vectors in objective space.

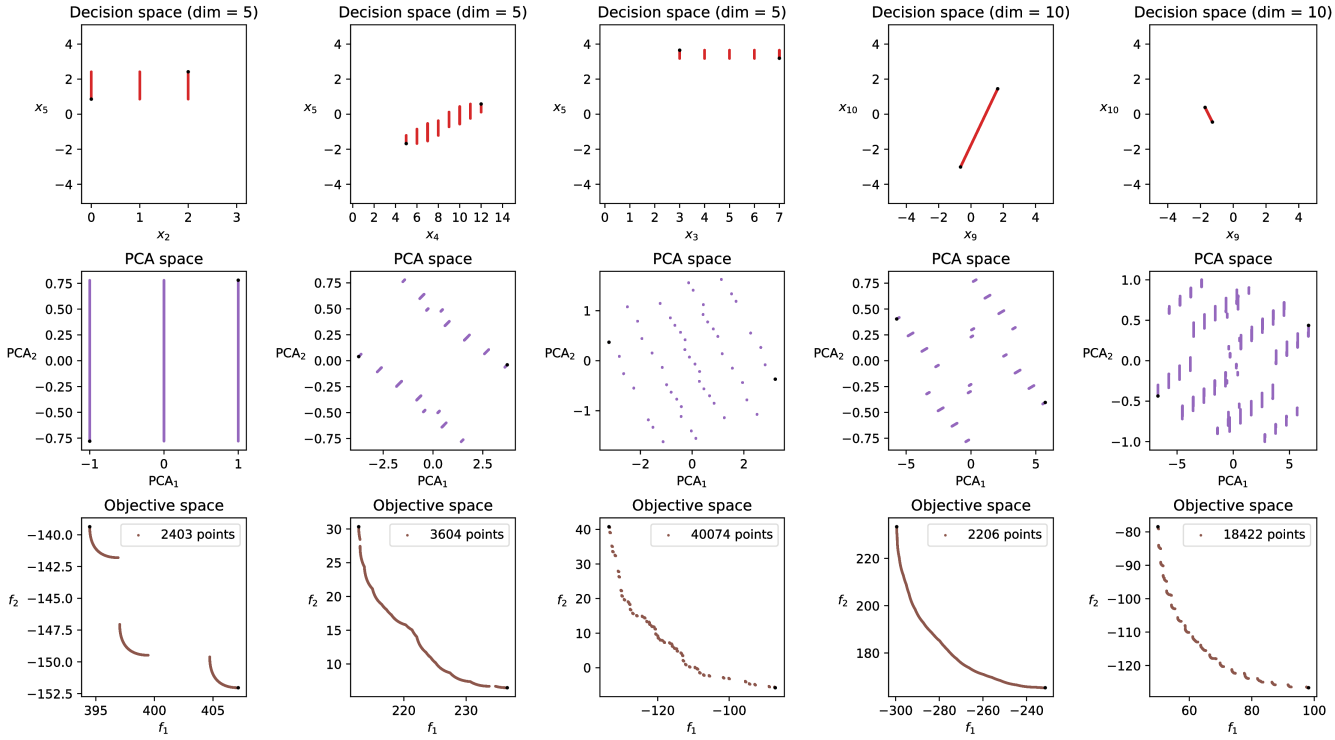
We observe the following properties of the function instances: Pareto sets and fronts are discretized and are made up from piecewise linear parts (in the case of the Pareto set) or piece-wise convex parts (in the case of the Pareto front). The projection of the Pareto set can thereby look connected (in the two rightmost columns) and the Pareto front, globally, is not convex everywhere (see for example the middle part of the Pareto front in Figure 5 (b)). The number of non-connected Pareto set and Pareto front parts differs widely from instance to instance. This seems to depend on the number of distinct variable values among the non-dominated solutions in the considered solution sets and thus indirectly on the placement of the two single-objective optima with respect to the discretization grid, which is exactly the difference among instances of the double sphere function.

6 EXAMPLE ALGORITHM PERFORMANCE

In this section we investigate the newly proposed `bbob-mixint` suite from an algorithm standpoint: how difficult is it to solve the `bbob-mixint` problems in terms of the number of function evaluations to reach certain target difficulties? To this end, we implemented the `bbob-mixint` functions in the COCO platform, run four algorithms on them and report and discuss the results here in terms of empirical cumulative distribution functions of the recorded runtimes.

6.1 Algorithm Details

Overall, we chose four algorithms to run on the `bbob-mixint` suite—a selection which contains algorithms from various domains but which is also not exhaustive or in any way representative of the entire set of available algorithms for mixed-integer optimization.



(a) Dim 5, instance 1 (b) Dim 5, instance 7 (c) Dim 5, instance 10 (d) Dim 10, instance 8 (e) Dim 10, instance 11

Figure 5: Optimal solutions to the `bbob-mixint` suite double sphere problems (function F_1) of different dimensions and instances depicted in the decision (top), PCA (middle) and objective (bottom) spaces. The decision space plots are projections to the 2-D space of the two variables that have the highest number of distinct values. The PCA plots show the projection using the first two principal components. In addition to showing the Pareto fronts, the objective space plots contain information of the number of visualized solutions (equal for all three plots of the same problem).

The main focus here is still the introduction of a new test suite and not the benchmarking of a large portion of the existing algorithms.

The four selected algorithms are

- Random Search as a baseline, sampling each variable uniformly at random within their bounds (i.e., from [-5,5] for the continuous variables, from 0 to the maximum value for the integer ones),
- Differential Evolution (DE) [21] with the strategy 'rand/1/bin' and population size $10 \cdot (4 + \lfloor 3 \log(n) \rfloor)$, implemented in the `scipy` Python module [12], handling the problems as if they were continuous and bounded by their region of interest,
- a CMA-ES variant with simplistic integer handling from `pycma` as showcased in Figure 2 with boundary handling and a variable-wise initial step size that is chosen relative to the difference between the smallest and largest value, and finally
- TPE (Tree-structured Parzen Estimator, [1]), a common algorithm for hyperparameter tuning, as implemented in the Python module `hyperopt`.

For TPE, the discrete variables have been represented with a uniform distribution on their discrete values and for the continuous variables, we used a normal distribution with mean 0 and standard deviation 2. The experiments were run for a budget of $10^4 n$ for Random Search, DE, and CMA-ES and for a budget of $100n$ for TPE (due its long internal runtime). If not mentioned otherwise, we use the default settings for each algorithm.

6.2 Display Settings

For the displayed algorithm performance, we fall back on the empirical cumulative distribution plots (ECDF) of the recorded runtimes that the COCO framework provides by default for new test suites. Because of the scaling of the `bbob-mixint` functions (see Section 3.1.3), we can keep the standard target difficulties of 51 values, uniformly chosen on a logscale between 100 and 10^{-8}. All experiments have been run on instances 1–15.

6.3 Algorithm Performances

The ECDFs for the four selected algorithms, aggregated over all 24 `bbob-mixint` functions, can be found in the top row of Figure 6 for dimensions 5, 10, 20, and 40. Selected single-function ECDFs are shown in the middle row for dimension 5 and in the bottom row for dimension 20. We see that:

- Over all `bbob-mixint` problems, the algorithms perform worse with increasing dimension.

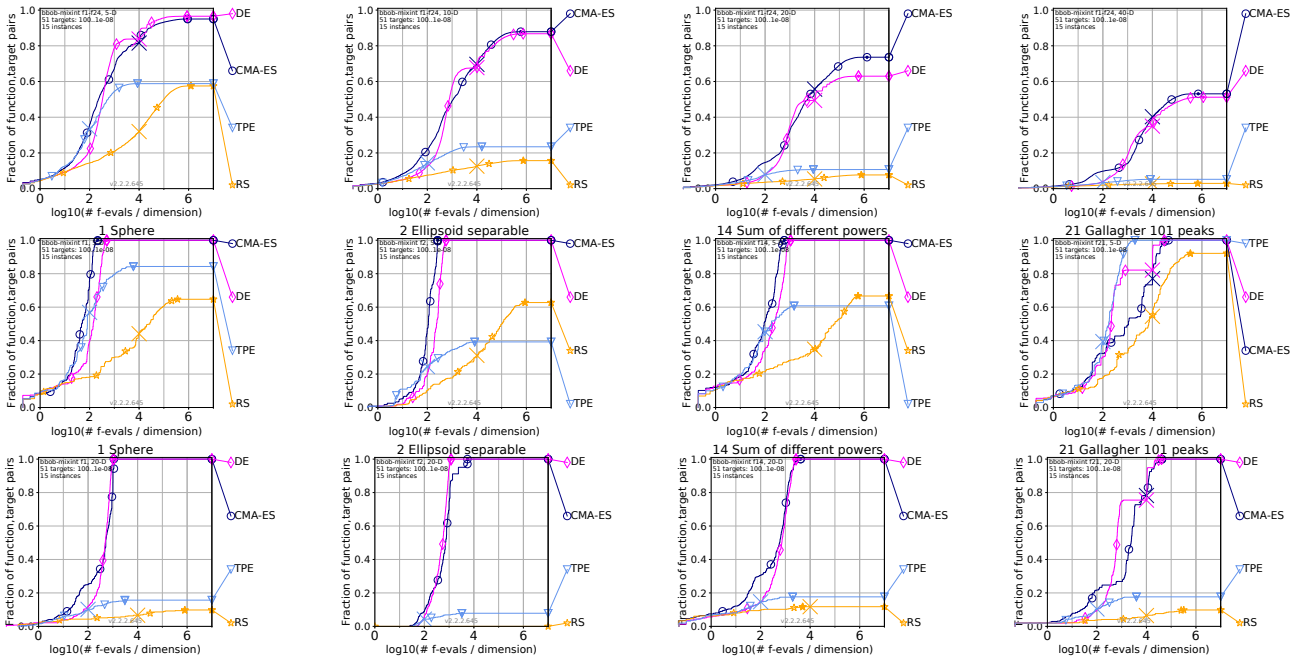


Figure 6: Empirical cumulative distribution of simulated runtimes over 51 targets for four selected mixed-integer algorithms. The first row shows the results aggregated over all 24 `bbob-mixint` functions in dimensions 5, 10, 20, and 40 (from left to right). The second (in dimension 5) and third row (in dimension 20) shows results for four selected `bbob-mixint` functions.

- Random Search and TPE (with its small budget of $100n$ function evaluations) are effected the most by increasing dimension such that only very few targets are reached in dimension 40. For the functions with low or moderate conditioning (f_6–f_9), not a single target ≤ 100 is reached in dimensions 40 and higher (plot not shown here). DE and CMA-ES, contrary to Random Search, still solve about 35–40% of the targets up to their budget in dimension 40. These trends continue in higher dimensions (not shown).
- TPE is mainly hampered by the low budget but performs particularly well on the multimodal Gallagher function in dimension 5.

7 CONCLUSIONS

In this paper we have introduced two mixed-integer benchmark testbeds with scalable functions between dimensions 5 and 160. Our testbeds are based on two benchmark suites available in the COCO benchmarking platform and have been implemented within the platform as `bbob-mixint` and `bbob-biobj-mixint` suites.

We investigated the properties of the functions of the new suites and found that discretization has a remarkable impact on the function properties in some cases. Taking a closer look at some unimodal functions of the single-objective `bbob-mixint` suite, we found that discretization of non-separable ill-conditioned convex-quadratic functions is likely to introduce local optima in the originally unimodal problem. We also can understand the mechanism: given a direction of improvement is diagonal, we have to change several coordinates to get to the next better setting. We conjecture that these new discretized functions are much harder problems to optimize than their continuous counterparts.

Investigating the Pareto fronts and sets of instances of the discretized double sphere function, we found a rich variety of disconnected shapes already for this supposedly simplest of all bi-objective function, standing in stark contrast to the simple Pareto front/set of its continuous version. Hence it will take some effort to identify the Pareto fronts of the new bi-objective functions to make them available for comparative benchmarking.

Both proposed suites use a single setting for variable arities and of the number of variables with these given arities. This choice has been made because problems with notably different variable arities may well require different strategies. For example, in case of a larger number of continuous variables and a low number of all possible discrete values, the best strategy might be to separately solve each corresponding continuous problem, which is not a viable approach when their number is high. Therefore, this setting should preferably not change within a problem suite which, as a desired side effect, also keeps the number of functions in reasonable limits.

ACKNOWLEDGMENTS

The first author acknowledges the financial support from the Slovenian Research Agency (project No. Z2-8177). This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences. The authors are grateful to the BBOB team for its decade long dedicated work.

REFERENCES

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS 2011)*. 2546–2554.

[2] D. Brockhoff, T. Tušar, A. Auger, and N. Hansen. 2019. Using well-understood single-objective functions in multiobjective black-box optimization test suites. *ArXiv e-prints* arXiv:1604.00359v3 (2019).

[3] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. 2016. Biobjective performance assessment with the COCO platform. *ArXiv e-prints* arXiv:1605.01746 (2016).

[4] M. R. Bussieck, A. S. Drud, and A. Meeraus. 2003. MINLPLib – A collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing* 15, 1 (2003), 114–119.

[5] K. Deep, K. P. Singh, M. L. Kansal, and C. Mohan. 2009. A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Appl. Math. Comput.* 212, 2 (2009), 505–518.

[6] S. Finck, N. Hansen, R. Ros, and A. Auger. 2009. *Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions.* Technical Report 2009/20. Research Center PPE, Austria. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.

[7] N. Hansen, Y. Akimoto, and P. Baudis. 2019. CMA-ES/pycma on Github. Zenodo. (Feb. 2019). https://doi.org/10.5281/zenodo.2559634

[8] N. Hansen and A. Auger. 2014. Principled design of continuous stochastic search: From theory to practice. In *Theory and principled methods for the design of metaheuristics*, Y. Borenstein and A. Moraglio (Eds.). Springer, Berlin, Heidelberg, 145–180.

[9] N. Hansen, A Auger, D. Brockhoff, D. Tušar, and T. Tušar. 2016. COCO: Performance assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[10] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[11] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle. 2014. AClib: A benchmark library for algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION 8) (Lecture Notes in Computer Science)*, Vol. 8426. Springer, 36–40.

[12] E. Jones, T. Oliphant, P. Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. (2001–). http://www.scipy.org/ [Online; accessed 5. 2. 2019].

[13] S. A. Kauffman. 1993. *The Origins of Order: Self-Organization and Selection in Evolution.* OUP USA.

[14] R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. 2013. Mixed integer evolution strategies for parameter optimization. *Evolutionary computation* 21, 1 (2013), 29–64.

[15] R. Li, M. T. M. Emmerich, J. Eggermont, E. G. P. Bovenkamp, T. Bäck, J. Dijkstra, and J. H. C. Reiber. 2006. Mixed-integer NK landscapes. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN IX) (Lecture Notes in Computer Science)*, Vol. 4193. Springer, 42–51.

[16] T. Liao, K. Socha, M. A. M. de Oca, T. Stützle, and M. Dorigo. 2014. Ant colony optimization for mixed-variable optimization problems. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014), 503–518.

[17] T. W. Liao. 2010. Two hybrid differential evolution algorithms for engineering design optimization. *Applied Soft Computing* 10, 4 (2010), 1188–1199.

[18] K. McClymont and E. Keedwell. 2011. Benchmark multi-objective optimisation test problems with mixed encodings. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011)*. IEEE, 2131–2138.

[19] J. J. Moré and S. M. Wild. 2009. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* 20, 1 (2009), 172–191.

[20] J. Müller. 2016. MISO: Mixed-integer surrogate optimization framework. *Optimization and Engineering* 17, 1 (2016), 177–203.

[21] R. Storn and K. V. Price. 1997. Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359.

[22] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. 2005. *Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization.* KanGAL report 2005005. IIT Kanpur, India.

[23] T. Tušar, D. Brockhoff, N. Hansen, and A. Auger. 2016. COCO: The bi-objective black-box optimization benchmarking (bbob-biobj) test suite. *ArXiv e-prints* arXiv:1604.00359 (2016).

[24] K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. A. ElHara, Y. Semet, R. Kassab, and F. Barbaresco. 2018. A comparative study of large-scale variants of CMA-ES. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN XV) (Lecture Notes in Computer Science)*, Vol. 11101. Springer, 3–15.