Permuted Orthogonal Block-Diagonal Transformation Matrices for Large Scale Optimization Benchmarking

Ouassim Ait Elhara, Anne Auger, Nikolaus Hansen

Permuted Orthogonal Block-Diagonal Transformation Matrices for Large Scale Optimization Benchmarking

Ouassim Ait ElHara
LRI, Univ. Paris-Sud
Université Paris-Saclay
TAO Team, Inria
firstname.ait_elhara@inria.fr

Anne Auger
Inria
LRI, Univ. Paris-Sud
Université Paris-Saclay
lastname@lri.fr

Nikolaus Hansen
Inria
LRI, Univ. Paris-Sud
Université Paris-Saclay
lastname@lri.fr

ABSTRACT

We propose a general methodology to construct large-scale testbeds for the benchmarking of continuous optimization algorithms. Our approach applies an orthogonal transformation on raw functions that involve only a linear number of operations in order to obtain large scale optimization benchmark problems. The orthogonal transformation is sampled from a parametrized family of transformations that are the product of a permutation matrix times a block-diagonal matrix times a permutation matrix. We investigate the impact of the different parameters of the transformation on its shape and on the difficulty of the problems for separable CMA-ES. We illustrate the use of the above defined transformation in the BBOB-2009 testbed as replacement for the expensive orthogonal (rotation) matrices. We also show the practicability of the approach by studying the computational cost and its applicability in a large scale setting.

Keywords

Large Scale Optimization; Continuous Optimization; Benchmarking.

1. INTRODUCTION

The general context of this paper is numerical optimization in a so-called black-box scenario. More precisely one is interested in estimating the optimum of a function

$$\mathbf{x}_{\text{opt}} = \arg\min f(\mathbf{x}) \ , \tag{1}$$

with $\mathbf{x} \in S \subseteq \mathbb{R}^d$ and $d \geq 1$ the dimension of the problem. The black-box scenario means that the only information on f that an algorithm can use is the *objective function value* $f(\mathbf{x})$ at a given queried point $\mathbf{x}$. There are two conflicting objectives: (i) estimating $\mathbf{x}_{\text{opt}}$ as precisely as possible with (ii) the least possible cost, that is the least number of function evaluations.

Benchmarking is a compulsory task to evaluate the performance of optimization algorithms designed to solve (1). In the context of optimization, it consists in running an algorithm on a set of test functions and extracting different performance measures from the generated data. To be meaningful, the functions on which the algorithms are tested should relate to real-world problems. The COCO platform is a benchmarking platform for comparing continuous optimizers [3]. In COCO, the philosophy is to provide meaningful and quantitative performance measures. The test functions are comprehensible, modeling well-identified difficulties encountered in real-world problems such as ill-conditionning, multi-modality, non-separability, skewness and the lack of a structure that can be exploited.

In this paper, we are interested in designing a large-scale test suite for benchmarking large-scale optimization algorithms. For this purpose we are building on the BBOB-2009 testbed of the COCO platform. We consider a continuous optimization problem to be of large scale when the number of variables is in the hundreds or thousands. Here, we start by extending the dimensions used in COCO (up to 40) and consider values of d up to at least 640.

There are already some attempts on large scale continuous optimization benchmarking. The CEC special sessions and competitions on large-scale global optimization have extended some of the standard functions to the large scale setting [9, 8]. In the CEC large scale testbed, four classes of functions are introduced depending on their level of separability. These classes are: (a) separable functions; (b) partially separable functions with a single group of dependent variables; (c) partially separable functions with several independent groups, each group comprised of dependent variables; and (d) fully non-separable functions. On the partially separable functions, variables are divided into groups and the overall fitness is defined as the sum of the fitness of one or more functions on each of these groups independently. Dependencies are obtained by applying an orthogonal matrix to the variables of a group. By limiting the sizes of the groups, the orthogonal matrices remain of reasonable size, which allows their application in a large scale setting.

The approach we consider in this paper is to start with the BBOB-2009 testbed [6] from the COCO platform and replace the transformations that do not scale linearly with the problem dimension d with more efficient variants whilst trying to conserve these functions' properties.

The paper is organized as follows: in Section 2 we motivate and introduce our particular transformation matrix for large scale benchmarking and identify its different parameters. Section 3 treats the effects of the different parameters

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908937

on the transformed problem and its difficulty. We choose possible default values for the parameters in Section 4 and give some implementation details and complexity measures in Section 5 where we also illustrate how the transformation can be applied in COCO. We sum up in Section 6.

2. BBOB-2009 RATIONALE AND THE CASE FOR LARGE SCALE

Our building of large-scale functions within the COCO framework is based on the BBOB-2009 testbed [6]. We are explaining in this section how this testbed is built, and how we intend to make it large-scale friendly.

2.1 The BBOB-2009 Testbed

The BBOB-2009 testbed relies on the use of a number of raw functions from which 24 different problems are generated. Here, the notion of raw function designates functions in their basic form applied to a non-transformed (canonical base) search space. For example, we call the *raw ellipsoid function* the convex quadratic function with a diagonal Hessian matrix whose i^{th} element equals $2 \times 10^{6\frac{i-1}{d-1}}$, that is $f^{\text{Elli}}(\mathbf{x}) = \sum_{i=1}^{d} 10^{6\frac{i-1}{d-1}} x_i^2$. On the other hand, the separable ellipsoid function used in BBOB-2009 (f_2) sees the search space transformed by a translation and an oscillation T^{osz} (4), that is, it is called on $\mathbf{z} = T^{\text{osz}}(\mathbf{x} - \mathbf{x}_{\text{opt}})$ instead of $\mathbf{x}$, with $\mathbf{x}_{\text{opt}} \in \mathbb{R}^d$. Table 1 shows normalized and generalized examples of such widely used raw functions.

The 24 BBOB-2009 test functions are obtained by applying a series of transformations on such raw functions that serve two main purposes: (i) have non trivial problems that can not be solved by simply exploiting some of their properties (separability, optimum at fixed position...) and (ii) allow to generate different instances, ideally of similar difficulty, of a same problem. The second item is made possible by the random nature of some of the transformations (a combination of the function identifier and the instance number is provided as seed to the random number generator). All functions have, at least, one random transformation, which allows this instantiation to work on all of them.

First, the optimal solution $\mathbf{x}_{\text{opt}}$ and its fitness f_{opt} are generated randomly. These allow to have the optimal value and its fitness non-centered at zero:

$$f(\mathbf{x}) = f_{\text{raw}}(\mathbf{z}) + f_{\text{opt}} \quad , \tag{2}$$

with $\mathbf{z} = \mathbf{x} - \mathbf{x}_{\text{opt}}$ and f_{raw} a raw function.

In addition to the translation by $\mathbf{x}_{\text{opt}}$, the search space might be perturbed using one or both of the deterministic transformations T^{asy} and T^{osz} defined coordinate-wise for $i = 1 \dots d$ as

$$\left[T_\beta^{\text{asy}}(\mathbf{x})\right]_i = \begin{cases} x_i^{1+\beta\frac{i-1}{d-1}\sqrt{x_i}}, & \text{if } x_i > 0 \\ x_i, & \text{otherwise} \end{cases} \quad , \tag{3}$$

$$\left[T^{\text{osz}}(\mathbf{x})\right]_i = \text{sign}(\hat{x}_i + 0.049(\sin(c_1\hat{x}_i) + \sin(c_2\hat{x}_i))) \quad , \tag{4}$$

where $[\mathbf{x}]_i = x_i$, $\hat{x}_i = \log(|x_i|)$ if $x_i \neq 0$, and 0 otherwise and $(c_1, c_2) = (10, 7.9)$ when $x_i > 0$, $(c_1, c_2) = (5.5, 3.1)$ otherwise. T^{osz} applies a non linear oscillation to the coordinates while T^{asy} introduces a non-symmetry between the positive and negative values.

The variables are also scaled using a diagonal matrix Λ^α whose i^{th} diagonal element equals $\alpha^{\frac{1}{2}\frac{i-1}{d-1}}$.

In order to introduce non-separability and coordinate system independence, another transformation consists in applying an orthogonal matrix to the search space: $\mathbf{x} \mapsto \mathbf{z} = \mathbf{R}\mathbf{x}$, with $\mathbf{R} \in \mathbb{R}^{d\times d}$ an orthogonal matrix.

If we take the Rastrigin function (f_{15}) from [6] (and normalize it in our case), it combines all the transformations defined above:

$$f_{15}(\mathbf{x}) = f_{\text{raw}}^{\text{Rastrigin}}(\mathbf{z}) + f_{\text{opt}} \quad , \tag{5}$$

where $\mathbf{z} = \mathbf{R}\Lambda^{10}\mathbf{Q}T_{0.2}^{\text{asy}}\left(T^{\text{osz}}\left(\mathbf{R}\left(\mathbf{x} - \mathbf{x}_{\text{opt}}\right)\right)\right)$, $f_{\text{raw}}^{\text{Rastrigin}}$ as defined in Table 1 and $\mathbf{R}$ and $\mathbf{Q}$ two $d \times d$ orthogonal matrices.

2.2 Extension to Large Scale

Complexity-wise, all the transformations above, *bar the application of an orthogonal matrix*, can be computed in linear time, and thus remain applicable in practice in a large scale setting. The problems in COCO are defined, in most cases, by combining the above defined transformations. Additional transformations are of linear cost and can be applied in a large scale setting without affecting computational complexity.

Our idea is to derive a computationally feasible large scale optimization test suite from the BBOB-2009 testbed [6], while preserving the main characteristics of the original functions. To achieve this goal, we replace the computationally expensive transformations that we identified above, namely full orthogonal matrices, with orthogonal transformations of linear computational complexity: *permuted orthogonal block-diagonal matrices.*

2.2.1 Objectives

The main properties we want the replacement transformation matrix to satisfy are to:

1. **Have (almost) linear cost:** both the amount of memory needed to store the matrix and the computational cost of applying the transformation matrix to a solution must scale, ideally, linearly with d or at most in $d\log(d)$ or $d^{1+\varepsilon}$ with $\varepsilon \ll 1$.

2. **Introduce non-separability:** the desired scenario is to have a parameter/set of parameters that allows to control the difficulty and level of non-separability of the resulting problem in comparison to the original, non-transformed, problem.

3. **Preserve, apart from separability, the properties of the raw function:** as in the case when using a full orthogonal matrix, we want to preserve the condition number and eigenvalues of the original function when it is convex-quadratic.

2.2.2 Block-Diagonal Matrices

Sparse matrices, and more specifically band-matrices satisfy Property 1. However, a *full* band matrix (all elements in the band are non-zero) can not be orthogonal, and thus does not satisfy Property 3, unless the band-width is 1 (see appendix). This means that the matrix is diagonal, which contradicts Property 2.

A special case of band matrices that can be non-diagonal and still orthogonal are *block-diagonal matrices.* A block-

diagonal matrix $\mathbf{B}$ is a matrix of the form

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 & \cdots & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}_{n_b} \end{pmatrix} \quad , \tag{6}$$

where n_b is the number of blocks and $\mathbf{B}_i, 1 \leq i \leq n_b$ are square matrices of sizes $s_i \times s_i$ satisfying $s_i \geq 1$ and $\sum_{i=1}^{n_b} s_i = d$.

Property 1 relies upon the number of non-zero entries of $\mathbf{B}$ which is

$$\sum_{i=1}^{d} \sum_{j=1}^{d} \mathbb{1}_{b_{(i,j)} \neq 0} = \sum_{i=1}^{n_b} s_i^2 \quad , \tag{7}$$

where $\mathbb{1}_{b_{(i,j)} \neq 0}$ is the indicator function that equals 1 when $b_{(i,j)} \neq 0$ and 0 otherwise and $b_{(i,j)}$ the i^{th} row j^{th} column element of $\mathbf{B}$. If we simplify and consider blocks of equal sizes s, bar possibly the last block, then $n_b = \lceil d/s \rceil$. We end up with at most $d \times s$ non-zero entries. Then, in order to satisfy Property 1 with a number of non-zero elements linear in d, s needs to be independent of d. In the case of different block-sizes, the same reasoning can be applied to the largest block-size instead of s.

For Property 2, as long as the matrices $\mathbf{B}_i$ in (6) are not diagonal, the functions that are originally separable become non-separable since correlations between variables are introduced.

Now for Property 3, $\mathbf{B}$ is orthogonal if and only if all the matrices $\mathbf{B}_i$ are orthogonal. We want to have the matrices $\mathbf{B}_i$ uniformly distributed in the set of orthogonal matrices of the same size (the orthogonal group $\mathrm{O}(s_i)$). We first generate square matrices with entries i.i.d. standard normally distributed. Then we apply the Gram-Schmidt process to orthogonalize these matrices. The resulting $\mathbf{B}_i$, and thus $\mathbf{B}$, are orthogonal and satisfy Property 3.

Orthogonal block-diagonal matrices are the raw transformation matrices for our large scale functions. Their parameters are

- d, defines the size of the matrix,
- $\{s_1, \ldots, s_{n_b}\}$, the block sizes where n_b is the number of blocks.

It is also possible to consider rank deficient matrices with a limited number, d_{eff}, of non-zero columns. We end up with $d \times d_{\text{eff}}$ non zero entries. Such matrices result in transformed problems with a low effective dimension. This approach was introduced, tested and discussed in another work.

2.2.3 Permutations

With the model above, we have seen that we do introduce non-separability. However, dependencies remain limited to the blocks of $\mathbf{B}$, so variables from different blocks have no chance of being correlated (unless they already are in the raw function). Such a property seems to be rather artificial and could be exploited by algorithms.

To prevent this, two permutations are applied, one to the rows of $\mathbf{B}$ and the other to its columns. The purpose of these permutations is to *hide* the block-diagonal structure of $\mathbf{B}$. The overall transformation matrix $\mathbf{R}$ becomes

$$\mathbf{R} = \mathbf{P}_{\text{left}} \mathbf{B} \mathbf{P}_{\text{right}} \quad , \tag{8}$$

with $\mathbf{P}_{\text{right}}, \mathbf{P}_{\text{left}}$ two permutation matrices, that is, matrices that have one and only one non-zero element (equal to 1) per row and per column. Property 3 remains satisfied since permutation matrices are orthogonal, and the product of orthogonal matrices is an orthogonal matrix.

In effect, the permutation $\mathbf{P}_{\text{left}}$ shuffles the variables in which the raw function is defined[1]. Hence, regularities coming from the ordering of variables in this definition, e.g. monotonously increasing coefficients, are perturbed. As a result, the permutation determines which coefficients are used within the blocks defined by $\mathbf{B}$ and, consequently, the block condition numbers and the difficulty of the problem (see the appendix). The permutation $\mathbf{P}_{\text{right}}$ shuffles the variables which are accessed by the optimization algorithm, thereby hiding the block-structure of $\mathbf{B}$. Most numerical optimization algorithms are invariant to the choice of $\mathbf{P}_{\text{right}}$.

Generating the Random Permutations.

When applying the permutations, especially $\mathbf{P}_{\text{left}}$, one wants to remain in control of the difficulty of the resulting problem. Ideally, the permutation should have a parameterization that easily allows to control the difficulty of the transformed problem.

We define our permutations as series of n_s successive swaps. To have some control over the difficulty, we want each variable to *travel*, in average, a fixed distance from its starting position. For this to happen, we consider *truncated uniform swaps*.

In a truncated uniform swap, the second swap variable is chosen uniformly at random among the variables that are within a fixed range r_s of the first swap variable. Let i be the index of the first variable to be swapped and j be that of the second swap variable, then

$$j \sim \mathcal{U}\left(\{l_b(i), \ldots, u_b(i)\} \setminus \{i\}\right) \quad , \tag{9}$$

where $\mathcal{U}(S)$ the uniform distribution over the set S and $l_b(i) = \max(1, i - r_s)$ and $u_b(i) = \min(d, i + r_s)$.

When $r_s \leq (d-1)/2$, the average distance between the first and second swap variable ranges from $(\sqrt{2}-1)r_s + 1/2$ to $r_s/2 + 1/2$. It is maximal when the first swap variable is at least r_s away from both extremes or is one of them.

Algorithm 1 describes the process of generating a permutation using a series of truncated uniform swaps. The parameters for generating these permutations are:

- d, the number of variables,
- n_s, the number of swaps. Values proportional to d will allow to make the next parameter the only free one,
- r_s, the swap range and eventually the only free parameter. The swap range can be equivalently defined in the form $r_s = \lfloor r_r\, d \rfloor$, with $r_r \in [0, 1]$. Each variable moves in average about $r_r \times 50\%$ of the maximal distance d.

The indexes of the variables are taken in a random order thanks to the permutation π. This is done to avoid any bias with regards to which variables are selected as first swap

[1] Given a separable quadratic function $f(\mathbf{x}) = \mathbf{x}^T \mathbf{D} \mathbf{x}$, then $f(\mathbf{P}_{\text{left}} \mathbf{B} \mathbf{P}_{\text{right}} \mathbf{x}) = (\mathbf{P}_{\text{right}} \mathbf{x})^T \mathbf{B}^T [\mathbf{P}_{\text{left}}^T \mathbf{D} \mathbf{P}_{\text{left}}] \mathbf{B} (\mathbf{P}_{\text{right}} \mathbf{x})$ which illustrates that $\mathbf{P}_{\text{left}}$ permutes the coefficients of the diagonal matrix $\mathbf{D}$ ("variables in which the raw function is defined").

Algorithm 1 Truncated Uniform Permutations

Inputs: problem dimension d, number of swaps n_s, swap range r_s.

Output: a vector $\mathbf{p} \in \mathbb{N}^d$, defining a permutation.

1: $\mathbf{p} \leftarrow (1, \ldots, d)$
2: generate a uniformly random permutation π
3: **for** $1 \leq k \leq n_s$ **do**
4: $\quad i \leftarrow \pi(k)$, $x_{\pi(k)}$ is the first swap variable
5: $\quad l_b \leftarrow \max(1, i - r_s)$
6: $\quad u_b \leftarrow \min(d, i + r_s)$
7: $\quad S \leftarrow \{l_b, \ldots, u_b\} \setminus \{i\}$
8: $\quad$ Sample j uniformly in S
9: $\quad$ swap p_i and p_j
10: **end for**
11: return $\mathbf{p}$

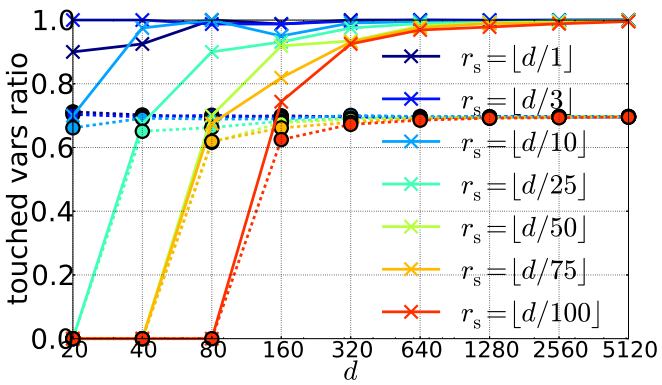

Figure 1: Proportion of variables that are moved from their original position when applying Algorithm 1 versus problem dimension d. Each graph corresponds to a different swap range r_s indicated in the legend. Solid lines: d swaps; dashed lines: $d/2$ swaps. When $r_s = 0$, no swap is performed. The averages of 51 repetitions per data point are shown. Standard deviations (not shown) are at most 10% of the mean.

variables when less than d swaps are applied. We start with $\mathbf{p}$ initially the identity permutation. We apply the swaps defined in (9) taking $p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n_s)}$, successively, as first swap variable (that replace i in (9)). The resulting vector $\mathbf{p}$ is returned as the desired permutation.

Impact of the Number of Swaps and Variable Pool Type.

Given the way the permutations are generated, some variables may end up being swapped more than once. This leads to variables potentially moving further than r_s away from their starting positions or (with low probability) even ending up in their original position after being swapped back.

In Figure 1, we look into the proportion of variables that are affected by the swap strategy. That is variables that end up in positions different from their initial ones. A different approach (that we call *static* while the current version is *dynamic*) was tried where Line 9 in Algorithm 1 swaps, instead, p_i^{-1} with p_j^{-1}, where $\mathbf{p}^{-1}$ is the, dynamically updated, inverse permutation of $\mathbf{p}$. This has shown no significantly different results.

We see that as d increases, the ratio of moved variables approaches what seem to be stationary values at 100% and 70% for respectively d and $d/2$ swaps. These stationary values are independent of the relative swap range r_r, except for relatively small dimensions in particular with d swaps. This is due to the fact that the probability of swapping a variable back to its original position increases as the swap range decreases. This probability is also affected by the actual value of the swap range $r_s = \lfloor r_r\, d \rfloor$: the larger the swap range, the smaller the probability of a variable being swapped back to its original position.

With this, we know that for large enough dimensions, only few variables are left unmoved. We fix the number of swaps to $n_s = d$, leaving only r_s as free parameter.

3. DIFFICULTY SCALING WITH MATRIX AND FUNCTION FEATURES

Now that we have decided upon the transformation matrix that we want to apply to the raw functions, namely permuted orthogonal block-diagonal matrix (8), and identified its free parameters, $(s_i)_{1 \leq i \leq n_b}$ and r_s, we are interested in how these two parameters, in addition to the problem dimension d, affect the transformed problem.

3.1 Impact of the Parameters on Structure of the Transformation Matrix

First, we are interested in the effect of $(s_i)_{1 \leq i \leq n_b}$, r_s and d on the shape of the sparse orthogonal matrix defined in (8). To this end, we investigate the dependency of the sum of the distances to the diagonal of the non-zero elements (d_{ToD}) on these parameters:

$$d_{\mathrm{ToD}}(\mathbf{R}) = \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbb{1}_{r_{(i,j)} \neq 0} |i - j| \ , \qquad (10)$$

where $\mathbb{1}_{r_{(i,j)} \neq 0}$ is the indicator function on the condition $r_{(i,j)} \neq 0$ and $r_{(i,j)}$ is element (i, j) of $\mathbf{R}$. This helps us measure how *different* from a diagonal matrix the transformed matrix is, which gives us an idea of the non-separability of the resulting problem.

In Figure 2, we show a normalized d_{ToD} of a matrix $\mathbf{R} = \mathbf{P}_{\mathrm{left}}\mathbf{B}$ with blocks of equal sizes depending on the values of d, s and r_s. For a full square matrix of size d, the corresponding d_{ToD} is

$$d_{\mathrm{ToD}}(\mathrm{full}) = \frac{1}{3} d(d-1)(d+1) \ . \qquad (11)$$

Which means that for a block-diagonal matrix $\mathbf{B}$ with blocks of equal sizes s:

$$d_{\mathrm{ToD}}(\mathbf{B}) = \left\lfloor \frac{d}{s} \right\rfloor \frac{s}{3}(s-1)(s+1) + \frac{1}{3} c(c-1)(c+1) \ , \qquad (12)$$

where $c = d - s \times \lfloor d/s \rfloor$ (which accounts in case for the last block of size $c < s$).

In order to better understand the effect of each parameter, we considered the following model for $r_r \geq 1/d$:

$$\hat{d}_{\mathrm{ToD}}(\alpha, d, s, r_r) = \alpha_0 (d + \alpha_1)^{\alpha_2} \times (s + \alpha_3)^{\alpha_4} \times (r_r + \alpha_5)^{\alpha_6} \times \left(\frac{s}{r_r\, d} + \alpha_7\right)^{\alpha_8} \ , \qquad (13)$$

where $\alpha = (\alpha_0, \ldots, \alpha_8)$ are the free parameters of the fit to be optimized.

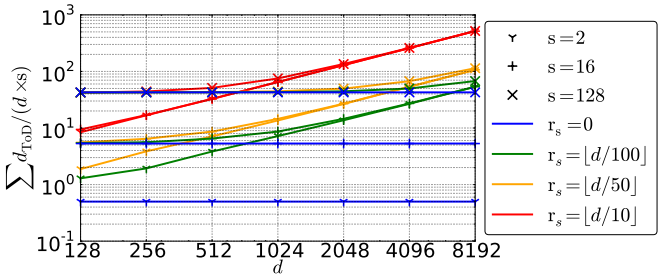

Figure 2: Normalized sum of distances $d_{\mathrm{ToD}}(\mathbf{B})$ (see (10)) of a full block-diagonal matrix after applying truncated uniform permutations (Algorithm 1) with $n_s = d$ versus problem dimension d. Different block-sizes s and swap-ranges r_s are considered. Each symbol is the average of 19 repetitions.

CMA-ES is then used to find an optimal fit on α of the empirical data by minimizing

$$\alpha_{\mathrm{opt}} = \arg\min_{\alpha} \sum_{i=1}^{\#data} \left(\log\left(\hat{d}_{\mathrm{ToD}}(\alpha, d^i, s^i, r_{\mathrm{r}}^i)\right) - \log\left(d_{\mathrm{ToD}}(\mathbf{R}^i)\right)\right)^2 , \quad (14)$$

where $\mathbf{R}^i$ is the i^{th} generated matrix and $d^i, s^i, r_{\mathrm{r}}^i$ are its corresponding parameter values. The logarithm is used to avoid biasing in favor of larger values of d. The resulting fit reads

$$0.15 \times (d - 2.25)^{2.03} \times (s + 0.07)^{0.95} \times (r_{\mathrm{r}} - 0.00)^{1.01} \times \left(\frac{s}{r_{\mathrm{r}}\, d} + 2.85\right)^{1.25} . \quad (15)$$

After simplifying, rounding, and re-running with increasingly more fixed parameter values, we end up with

$$d_{\mathrm{ToD}}(\mathbf{P}_{\mathrm{left}}\mathbf{B}) \approx 0.1 \times d^2 \times s \times r_{\mathrm{r}} \times \left(\frac{s}{r_{\mathrm{r}}\, d} + 4\right)^{1.3} . \quad (16)$$

We see a quadratic dependency on the problem dimension d and a linear one on the block-size s. In addition, the last term tends to be constant if $s = O(d)$. This means the scaling of d_{ToD} is $d \times s^2$, which reduces to d^3 for a full matrix $\mathbf{B}$ (11). The effect of the permutation can be estimated by the difference between the value in (16) and the value using the expression for the non permuted block-matrix (12).

3.2 Measure of Difficulty

We estimate the difficulty associated with a problem by the Expected Running Time (ERT) of sep-CMA-ES on this problem, that is the expected number of function evaluations needed (by the restarted algorithm) to reach a target value for the first time [5]. sep-CMA-ES is a linear time and space complexity variant of CMA-ES [7] that considers a diagonal covariance matrix instead of the full covariance matrix of default CMA-ES. However, and because of the restricted, diagonal, covariance matrix model, it only manages to solve problems with either no dependencies between the parameters or limited ones.

We use sep-CMA-ES as the basis for the difficulty estimation since it performs well at solving separable and block-separable functions when the condition numbers of the blocks are relatively small. It is invariant to the coordinate permutation $\mathbf{P}_{\mathrm{right}}$ which allows us to restrict our study to $\mathbf{P}_{\mathrm{left}}$ and $\mathbf{B}$. By comparing to the original, non transformed and separable problem, we get an idea of how *non-separable* the problem becomes. This helps us decide on the parameter choice. Considering that sep-CMA-ES is a rather trivial algorithm, we aim for parameter values that make the problem unsolvable by sep-CMA-ES in reasonable time.

4. PARAMETER CHOICE FOR THE BENCHMARKS

In this section, we choose the block-size and swap range parameters of the transformation matrix $\mathbf{P}_{\mathrm{left}}\mathbf{B}\mathbf{P}_{\mathrm{right}}$. Asuming only a single block-size s, the number of non-zero elements in the orthogonal block-matrix $\mathbf{B}$ is about $d \times s$. To achieve linear time and space complexity in d (see also Section 5.1), $s = O(1)$ is required. This means, firstly, we look for a *constant* value to set the block-size s. Secondly, s should be large enough to render the underlying problem non-trivial. This is, in our estimation, already the case when $s \geq 10$. Thirdly, the numerical costs of large-scale benchmarking suggest to chose s as small as possible. We estimate the relative numerical burden of introducing $\mathbf{B}$ in the order of $1 + s/5$. Fourthly, the largest orthogonal matrices used in BBOB-2009 are of size 40^2. A backwards compatible setting requires $s \geq \min(d, 40)$, which, in summary, makes $s = \min(d, 40)$ the favorite candidate setting.

Next, we consider the permutation or swap range. On the one hand, to induce a relevant exchange between blocks, a necessary requirement seems $r_s \not\ll s$. On the other hand, we consider the ratio between two eigenvalues, λ_i, λ_j, of the Ellipsoid function, which obey $\log_{10}(\lambda_i/\lambda_j) = 6(i-j)/(d-1)$. Swapping these eigenvalues with a constant swap range r_s, which determines $i - j$, the maximal ratio between swapped eigenvalues approaches one for $d \to \infty$. On the ellipsoid, swaps are only relevant if the eigenvalues differ significantly, where ratios ≤ 10 are rather irrelevant. For large values of d, only if $r_r = r_s/d$ is a constant, eigenvalues of significantly different values remain to be swapped. The median ratio between larger and smaller swapped eigenvalue is about 10^{3r_r}. For $r_r \leq 1/5$, the median ratio is ≤ 4, hence r_r should be larger. In order to still retain some of the original block structure, $r_r \leq 1/3$ seems advisable, therefore $r_r = 1/3$ is our favorite candidate setting.

We expect the rotated problems to remain non-solvable by separable algorithms such as sep-CMA-ES. Thus, we look into the performance of sep-CMA-ES on a transformed Ellipsoid function (see Table 1).

In Figure 3, the ERT of sep-CMA-ES is plotted against different values of r_s, s and d.

We notice two distinct effects of the parameters. Both s and r_s increase the ERT as their values increase. sep-CMA-ES manages to solve all non permuted problems ($r_s = 0$). This is mainly due to the relatively low condition number in each block ($c_b \approx 10^{6/n_b}$). However, as the swap range increases, we notice a change in phase in the ERT between $r_s = \lfloor d/10 \rfloor$ and $r_s = \lfloor d/3 \rfloor$. The permutation $\mathbf{P}_{\mathrm{left}}$ changes the ratios between the eigenvalues associated to each set of dependent variables, dependencies being defined by the block-diagonal matrix $\mathbf{B}$. Which in turn affects the *condition number* within each block of dependent variables, an effect similar to changing the block condition numbers of non-

$$
\begin{aligned}
f_{\text{raw}}^{\text{CigarGen}}(\mathbf{z}) &= \gamma(d)\times\left(\sum_{i=1}^{\lceil d/40\rceil} z_i^2 + 10^6 \sum_{i=\lceil d/40\rceil+1}^{d} z_i^2\right)\\
f_{\text{raw}}^{\text{DiffPow}}(\mathbf{z}) &= \gamma(d)\times\sum_{i=1}^{d} |z_i|^{\left(2+4\times\frac{i-1}{d-1}\right)}\\
f_{\text{raw}}^{\text{Elli}}(\mathbf{z}) &= \gamma(d)\times\sum_{i=1}^{d} 10^{6\frac{i-1}{d-1}} z_i^2\\
f_{\text{raw}}^{\text{Rosen}}(\mathbf{z}) &= \gamma(d)\times\sum_{i=1}^{d} \left(100\left(z_i^2 - z_{i+1}\right)^2 + \left(1 - z_i^2\right)\right)\\
f_{\text{raw}}^{\text{TabletGen}}(\mathbf{z}) &= \gamma(d)\times\left(10^6 \sum_{i=1}^{\lceil d/40\rceil} z_i^2 + \sum_{i=\lceil d/40\rceil+1}^{d} z_i^2\right)\\
f_{\text{raw}}^{\text{Rastrigin}}(\mathbf{z}) &= \gamma(d)\times 10\left(\left(d - \sum_{i=1}^{d}\cos\left(2\pi z_i\right)\right) + \|\mathbf{z}\|^2\right)
\end{aligned}
$$

Table 1: Raw definitions of the test functions used in this paper. The problem dimension is d. All functions are multiplied by $\gamma(d) = \min(1, 40/d)$ such that a constant target value (e.g., 10^{-8}) represents the *same level of difficulty* across all dimensions $d \geq 40$.

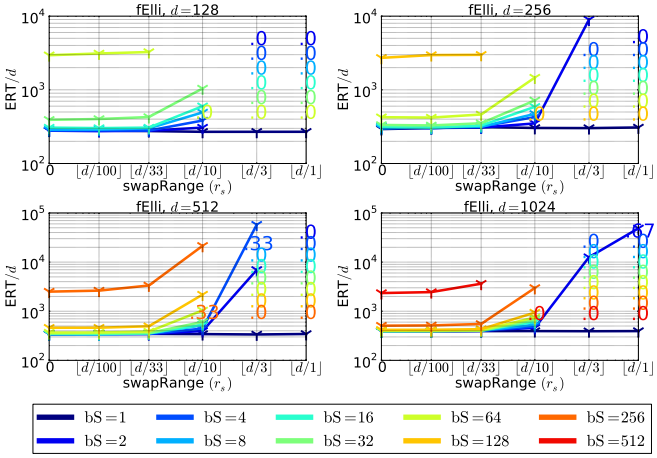

Figure 3: ERT of sep-CMA-ES on the transformed ellipsoid function. A budget of $10^5 \times d$ is used with 3 runs on each configuration. The target value is set to 10^{-8}. The setting $r_s = 0$ means no swap is applied ($\mathbf{P}_{\text{left}} = \mathbf{P}_{\text{right}} = I_d$).

permuted block-diagonal matrices as seen in the appendix. This increases the difficulty of the problem for sep-CMA-ES.

For $r_s = \lfloor d/3 \rfloor$ with a budget of $10^5 \times d$, sep-CMA-ES solves only problems with up to $s = 4$ across the tested dimensions.

In Figure 4, we do the same experiments as in Figure 3 on transformed versions of the Cigar, Tablet, Sum of Different Powers and Rosenbrock functions (see Table 1 with $\mathbf{z} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}$).

On the Sum of Different Powers function, the results are similar to what was observed on the transformed Ellipsoid function. The generalized Cigar and Tablet functions seem to be difficult enough with the block-matrix transformation alone. The only successes we see on these two functions (for $s > 1$) are for $s = 2$ with the smaller swap ranges. On the other hand, the Rosenbrock function is not convex quadratic and multi-modal, and is, in its raw version, a non-separable problem (partially separable with a tri-band structure). Failed runs of sep-CMA-ES are observed even on the raw function ($s = 1, r_s = 0$). We still observe an effect of s and r_s on the ERT. There is also a change in phase between the same two values of swap range $r_s = \lfloor d/10 \rfloor$ and $r_s = \lfloor d/3 \rfloor$, but that only happens for relatively higher

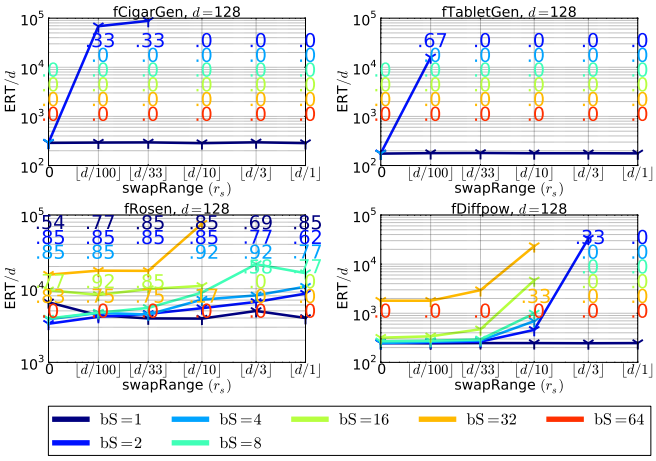

Figure 4: ERT of sep-CMA-ES on the transformed functions from Table 1 in dimension 128. The experimental setup is the same as in Figure 3 except on the transformed Rosenbrock functions where the ERT is computed from 19 runs.

block-sizes. For $r_s = \lfloor d/3 \rfloor$, successes are observed only as far as $s = 8$.

On the generalized Tablet and Cigar functions, and for $d = 128$, 4 eigenvalues are different from the rest. These eigenvalues are adjacent (the non permuted versions have them in the first 4 positions, see Table 1). Thus, the largest block condition number is equal to the overall condition number, except for the case $s \in \{1, 2, 4\}$ and $n_s = 0$ where it is equal to 1. This is why the only all-successful runs are observed on these configurations. Further confirming results on Tablet and Cigar functions with smaller overall condition numbers can be found in the appendix.

All unsuccessful runs in Figures 3 and 4 terminate with the stop flag *tolupsigma*. This indicates (taken from the documentation of the CMA-ES algorithm [4]): "creeping behavior with usually minor improvements". We interpret this as the model of sep-CMA-ES being unable to fit these problems. In such a case, we consider that sep-CMA-ES fails to solve the problem.

According to these experiments, a parameter setting $r_s = \lfloor d/3 \rfloor$ and $s = 16$ is sufficient to make the problems hard enough that sep-CMA-ES can not exploit the block-diagonal structure. This parameter setting is also in accordance with the considerations taken in the beginning of this section so we can safely choose

$$r_s = \lfloor d/3 \rfloor, s = \min(d, 40) \quad . \tag{17}$$

5. THE LARGE SCALE BENCHMARK

For the extension to the large scale setting, we do two main modifications to the raw functions in BBOB-2009 (see Table 1 for some examples of such modified raw functions). First, functions are normalized to have uniform target values that are comparable over a wide range of dimensions. Second, the Cigar and Tablet functions are generalized such that they have a constant proportion of distinct axes that remain consistent with the BBOB-2009 testbed.

As previously mentioned, for each function, we consider the same transformations that are used in [6] except the full orthogonal matrices that we replace by $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$. Ta-

$f^{\text{CigarGen}}:$	$\mathbf{z} = \mathbf{R}T_{0.5}^{\text{asy}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f^{\text{DiffPow}}:$	$\mathbf{z} = \mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$
$f^{\text{Elli}}:$	$\mathbf{z} = T^{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}}))$
$f^{\text{TabletGen}}:$	$\mathbf{z} = \mathbf{R}(\mathbf{x} - \mathbf{x}_{\text{opt}})$

Table 2: Example of transformed functions used in [6]. The functions T^{osz} and T^{asy} are defined in Section 2 and R is an orthogonal matrix. The raw definitions of the functions can be found in Table 1.

ble 2 shows examples of how the values of $\mathbf{z}$ in some of the functions in Table 1 are obtained in order to apply the raw functions and obtain the transformed benchmark functions.

5.1 Implementation Details and Cost of Applying the Transformation

Here we briefly present some of the technical implementation details that allow to store and compute the transformation introduced in this paper in an efficient way. The values we want to compute in order to apply the transformation are for each $i \in \{1, \ldots, d\}$:

$$z_i = [\mathbf{R}\mathbf{x}]_i = [\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}\mathbf{x}]_i \ , \tag{18}$$

where $[\mathbf{x}]_i$ designate the i^{th} coordinate of $\mathbf{x}$, $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are permutation matrices and $\mathbf{B}$ is a block-diagonal matrix (orthogonality is not important here) with block-sizes $(s_i)_{1 \le i \le n_b}$.

The matrix $\mathbf{B}$ is composed of n_b blocks that are square matrices $\mathbf{B}_i, 1 \le i \le n_b$ (see (6)). We store it in a list of vectors $\bar{\mathbf{B}} = (\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2, \ldots, \bar{\mathbf{b}}_d)$ that are the rows of the matrices $\mathbf{B}_1, \ldots, \mathbf{B}_{n_b}$. We denote $\bar{b}_{(i,j)}$ the coordinate j of $\bar{\mathbf{b}}_i$ and $\mathbf{b}_k$ the k^{th} row of $\mathbf{B}$. Since $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are permutation matrices, they are stored as two index (integer) vectors $\mathbf{p}^{\text{left}}$ and $\mathbf{p}^{\text{right}}$ respectively. They are used to index the coordinates to be permuted ($z_i^{\text{new}} = z_{p_i}$).

We will also keep two lists of d elements, $\mathbf{s}^{\text{row}}$ and $\mathbf{j}^{\text{first}}$, where for each $k \in \{1, 2, \ldots, d\}$

- s_k^{row} is the size of the vector $\bar{\mathbf{b}}_k$,
- j_k^{first} is the number of leading zeros, excluding the zeros in $\bar{\mathbf{b}}_k$, of $\mathbf{b}_k$. It allows to know the original position of each element $\bar{b}_{(k,j)}$ in $\mathbf{b}_k$ and is needed in order to multiply $\bar{b}_{(k,j)}$ by the corresponding coordinate of $\mathbf{x}$ (see (20)).

This amounts to a total number of entries needed to store $\bar{\mathbf{B}}, \mathbf{p}^{\text{right}}, \mathbf{p}^{\text{left}}, \mathbf{j}^{\text{first}}$ and $\mathbf{s}^{\text{row}}$ of

$$4d + \sum_{i=1}^{n_b} s_i^2 = 4d + n_b \text{avg}(s_i^2) \ , \tag{19}$$

where $\text{avg}(s_i^2)$ the average value of s_i^2. With similar block sizes s_i, we have $n_b\text{avg}(s_i^2) \approx n_b\text{avg}(s_i)^2 = d \times \text{avg}(s_i)$, and the number of entries scales linearly in d only if $\text{avg}(s_i) = O(1)$.

The vector $\mathbf{p}^{\text{left}}$ permutes the rows of $\mathbf{B}$ (and $\bar{\mathbf{B}}$) and $\mathbf{p}^{\text{right}}$ the coordinates of $\mathbf{x}$ and thus z_i satisfies

$$z_i = \sum_{j=\text{start}(p_i^{\text{left}})}^{\text{end}(p_i^{\text{left}})} \bar{b}_{(p_i^{\text{left}}, j - \text{start}(p_i^{\text{left}}))} x_{p_j^{\text{right}}} \ , \tag{20}$$

where $\text{start}(i) = j_i^{\text{first}}$ and $\text{stop}(i) = \text{start}(i) + s_i^{\text{row}}$. The computational cost of the transformation is then at worse in $O(d \times \max_i(s_i))$.

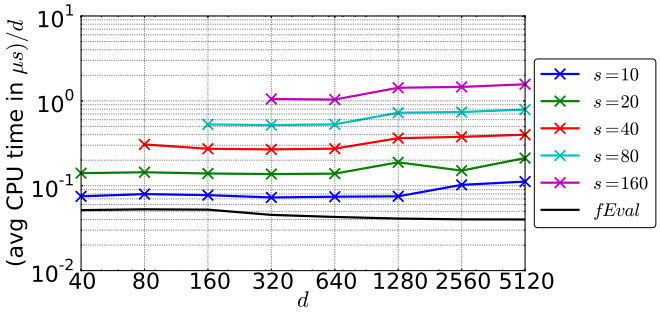

Figure 5: Average (over 10^4 samples) time needed to apply the transformation matrix (8) to a solution in micro-seconds, μs, divided by dimension for different values of the dimension d and block-size s. The solid black line shows the average cost of computing the raw ellipsoid function (see Table 1) in each dimension.

5.2 CPU Timing

We now look into the cost of computing the introduced transformation in terms of CPU time. We are particularly interested in its scaling with the problem dimension d. We use the C version of the COCO code [2].

Figure 5 shows the scaling of the CPU time per function evaluation spent in the transformation defined in (18) and implemented as explained in Section 5.1. Our objective is to estimate the added cost of applying the permuted orthogonal block-diagonal transformation for different values of d and s, regardless of what other transformations are involved in defining a problem. We also show the average CPU time spent computing the raw Ellipsoid function ($f_{\text{raw}}^{\text{Elli}}$) on the same machine (solid black graph). Experiments were run on a MacBook Pro with a 2.3GHz quad-core Intel Core i7 processor and 8 GB of RAM.

As expected, the needed CPU time scales linearly with d when the block size is kept constant and linearly with the block-size s for each given dimension. The latter can be deducted from the distance between the different graphs. Since s is constant, the overall CPU usage of the transformation is linear in d, thus satisfying Property 1.

A linear scaling can still be unusable in practice because of a large constant multiplier. In Figure 5, transformation with the largest considered block-size $s = 160$ takes around 40 times longer than computing the raw function value. The factor is around 10 for the chosen maximal block-size in (17), $s = 40$.

5.3 Result Post-Processing Example

Figure 6 is an example of a plot generated through the post-processing of the data produced by the COCO platform. It shows the ERT of sep-CMA-ES and default CMA-ES (we use the Python implementation from [4], version 1.0.09, for both algorithms) on the Sum of Different Powers function as defined in Tables 1 and 2 (f_{14} of BBOB-2009 with $\mathbf{R}$ as in (8)). The ERTs are computed on the first five instances of the function on each dimension. The ERTs for different target precisions are plotted where the precision is defined as the difference to the optimal fitness f_{opt}.

Within the budget of $10^4 \times d$, sep-CMA-ES reaches the target precision of 10^{-3} in dimension up to 80, and precision 10^{-5} in larger dimension, but never the final target. On

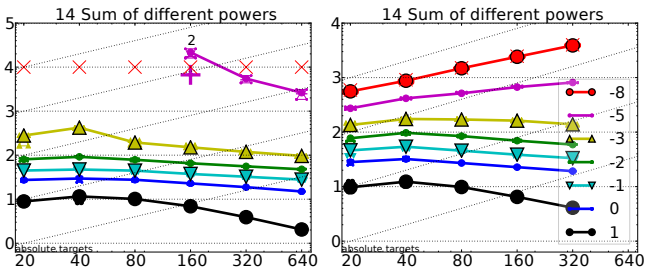

Figure 6: Number of f-evaluations of sep-CMA-ES (left) and standard CMA-ES (right) on five instances of the Sum of Different Powers function (large-scale extension of f_{14} of BBOB-2009 with permuted block-rotation (8) instead of full rotation) versus problem dimension d. Shown is $\log_{10}(\mathrm{ERT}/d)$ to reach $f_{\mathrm{opt}}+\Delta f$ where $\log_{10}(\Delta f)$ is given in the legend, e.g., the green line shows ERT$/d$ to reach a target value of $f_{\mathrm{opt}}+10^{-2}$. Notched boxes (most of them hidden behind the main symbols) show interquartile range and median, light red crosses (×) depict the used budget.

the other hand, CMA-ES reaches the final target in about $70 \times d^{1.7}$ function evaluations, which is $10^{3.8}d$ in dimension 640.

6. SUMMARY

In this paper, we introduced an orthogonal transformation to replace full orthogonal matrices as a means to generate non-trivial large scale problems. First, we outline a number of properties that we want the matrix to satisfy, among which (almost) linear computational and memory costs, in order for it to qualify as a large-scale benchmarking transformation.

The transformation in question that satisfies these properties consists in the product of a permutation matrix times an orthogonal block-diagonal matrix times another permutation matrix. The block-diagonal matrix introduces partial non-separability by making variables interact with other variables close to them. It is constructed from independently generated orthogonal matrices distributed uniformly in the set of orthogonal matrices that constitute the blocks. The permutation matrices allow to add another layer of complexity to the problem by allowing variables further away to interact. In addition, the permutation that is applied to the rows of the block-matrix determines the eigenvalues associated with the variables of a same block. This permutation is a determining factor for the difficulty of the problem. In order to have control over the difficulty, we introduced the notion of truncated uniform swaps. These swaps generate permutations where each variable travels in average a distance that is controlled by a single free parameter, the swap range.

The transformation matrix then replaces a *full* orthogonal matrix applied on functions that are commonly used for continuous optimization benchmarking.

The performance of sep-CMA-ES was investigated on a number of these transformed problems. This allowed to showcase a set of parameter values that is usable in a large scale setting and renders the transformed Ellipsoid function in our assessment non-trivial.

The introduced transformation is implemented in the COCO platform as the core component in its extension with a large scale test-suite. We gave some implementation details and the CPU timing of the proposed transformation. We have also shown example results produced with the COCO framework on a large scale problem.

Acknowledgments.

This work was supported by the grant ANR-12-MONU-0009 (NumBBO) of the French National Research Agency.

7. REFERENCES

[1] O. Ait Elhara, A. Auger, and N. Hansen. Permuted orthogonal block-diagonal transformation matrices for large scale optimization benchmarking. Inria, 2016. Preprint with appendix on https://hal.inria.fr/hal-01308566.

[2] Numbbo/coco: Comparing continuous optimizers, github repository. https://github.com/numbbo/coco. Accessed: 2016-01-26.

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

[4] Nikolaus Hansen. CMA-ES, Covariance Matrix Adaptation Evolution Strategy for non-linear numerical optimization in python. https://pypi.python.org/pypi/cma. Accessed: 2016-01-26.

[5] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Research Report RR-7215, INRIA, March 2010.

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

[7] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[8] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin, and Hefei China. Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. Technical report, Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013.

[9] Ke Tang, Xın Yáo, Ponnuthurai Nagaratnam Suganthan, Cara MacNish, Ying-Ping Chen, Chih-Ming Chen, and Zhenyu Yang. Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. *Nature Inspired Computation and Applications Laboratory, USTC, China*, pages 153–177, 2007.

APPENDIX

A. SPARSE MATRICES

A first idea is to have sparse transformation matrices , that is matrices which have a limited (ideally linear in d, the problem dimension) number of non-zero elements, and apply such the transformation in an efficient way that takes advantage of this sparse structure.

Some possibilities of the form in which the transformation matrix comes were considered. A first idea was to consider band matrices which are matrices which have all their non zero elements constrained in a fixed number of the diagonals of the matrix. For example, a tri-diagonal matrix B has all its entries at 0 but those that are at positions $(i, j), 1 \leq i, j \leq d$ with $|i - j| \leq 1$. The problem with such matrices is that only diagonal matrices ($B_{i,j} = 0, \forall i \neq j$) can be orthogonal, and diagonal matrices do not introduce dependencies between variables (in our case, and since we want all eigenvalues to be equal, the only effect would be to multiply all the variables by a given factor).

Take the example of a 3×3 tri-band matrix:

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & 0 \\ b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & b_{3,2} & b_{3,3} \end{pmatrix} .$$

For B to be orthogonal, the following system of equations must hold:

$$\begin{cases} b_{1,1}b_{1,2} + b_{2,1}b_{2,2} = 0 \\ b_{1,1}b_{1,2} + b_{2,1}b_{2,2} = 0 \\ b_{2,1}b_{2,3} = 0 \end{cases} .$$

By setting (because of the third equality) either of $b_{2,1}$ or $b_{2,3}$, we propagate the zeros to end up with $b_{i,j}, \forall i \neq j$. This proof can easily be generalized to any $d \times d$ tri-diagonal matrix.

Another possibility is to have rank deficient matrices with a limited number, d_{eff} (we end up with $d \times d_{\text{eff}}$ non zero entries) of non-zero columns. Such matrices result in a transformed problem with a low effective dimension. This approach was treated in another work . In addition, low effective dimension problems can be a subset of the testbed on large scale optimization, representing its corresponding class of problems.

The model we consider, and that allows us to introduce non-separability whilst keeping control over the eigenvalues, is *permuted orthogonal block-diagonal matrices.*

B. PERMUTATIONS

The point is then to have a swap strategy whose parameterization allows to control the difficulty of the problem. We call a *swap* exchange of positions of two variables. On the other hand, we call the resulting order, after applying a given number of swaps, the permutation, that is the *final* permutation matrix $\mathbf{P}_{\text{left}}$ or $\mathbf{P}_{\text{right}}$.

A first idea would be to apply a random uniformly chosen permutation of the variables, which is equivalent to applying an infinite number of random uniform swaps (in which the variables to be swapped are chosen uniformly at random). This strategy, however, resulted in the transformed problem being too difficult for sep-CMA-ES and lacks a parameter one can vary to achieve different levels of difficulty.

If we vary the number of random uniform swaps one applies to obtain the final permutation, we end up with a parameter that one can vary. However, the problem with this approach is that the difficulty of the resulted problem (tested by applying the transformation on an ellipsoid function and running sep-CMA-ES) is subject to a high amount of variance for a low number of swaps. In fact, depending on which variables are swapped, the resulting problem can differ substantially. Swapping variables which belong to the same group has no effect on the difficulty while swapping distant variables increases it considerably. The number of swaps needed to have a reasonable amount of variance is at least linear.

Other strategies were the variables to be swapped were no longer chosen uniformly at random but negatively correlated to the distances (that is the differences in the indexes) between these variables were tried but did not produce convincing results.

C. CONDITION NUMBER CONTROL OF CONVEX QUADRATIC FUNCTION

We obtain this control over the condition numbers (c_b and c_o) by setting the values of the diagonal entries of $\mathbf{D}$. Since $\mathbf{D}$ is multiplied by orthogonal matrices ($\mathbf{B}^T$ and $\mathbf{B}$), its eigenvalues are preserved.

Inside what corresponds to each block of $\mathbf{D}$, the eigenvalues are distributed uniformly in the logarithmic scale. All this is translated in the following equation:

$$\mathbf{D}_{i,i} = c_b^{\left(\frac{i\%s}{s-1}\right)} \times c_{ib}^{\lfloor i/s \rfloor} , \tag{21}$$

where % designates the modulo operator (rest of euclidean division) and c_{ib} a constant that ensures the overall condition number is preserved ($c_o = c_b \times c_{ib}^{(n_b - 1)}$, n_b being the number of blocks in $\mathbf{B}$).

D. IMPACT OF THE BLOCK CONDITION NUMBERS OF THE HESSIAN MATRIX ON THE PERFORMANCE OF SEP-CMA-ES

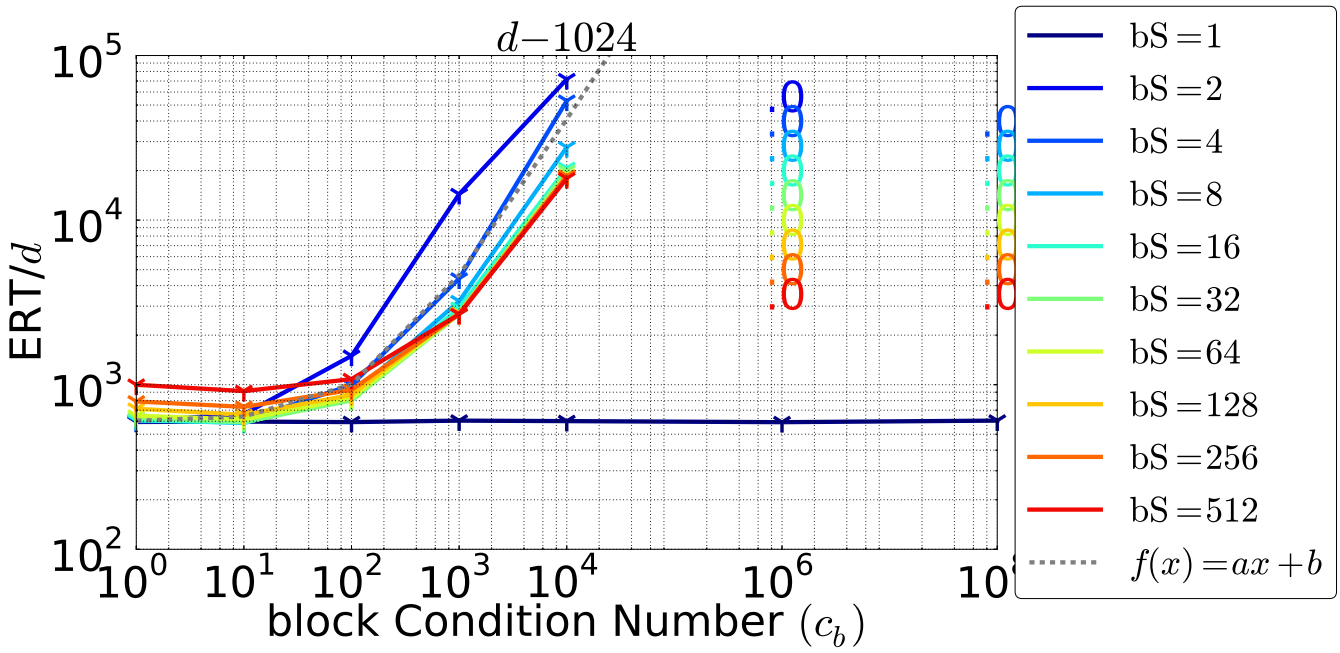

Figure 7: ERT of sep-CMA-ES on the function defined in (??) for different values of the block size. The overall condition number of the problem is $c_o = 10^8$, $d = 1024$ and the target value is set to 10^{-8}. Three runs are done per data point, each with a budget of $10^5 \times d$ function evaluations. The numbers with the color codes represent the success rates of the corresponding configurations (same color) when these are lower than 1. The dashed gray line shows a linear scaling.

We now discard the permutation matrices and look into the relationship between problem difficulty (in comparison to non-transformed problems) and block condition number of the Hessian matrix of the transformed problem.

To do so, we define a specific convex-quadratic function that suits the block structure of our matrix $\mathbf{B}$:

$$f_{\mathbf{B},\mathbf{D}}(\mathbf{x}) = \mathbf{x}^T \mathbf{B}^T \mathbf{D} \mathbf{B} \mathbf{x} \ , \tag{22}$$

where $\mathbf{D} \in \mathbb{R}^{d \times d}$ is a diagonal matrix and $\mathbf{B}$ an orthogonal block-diagonal matrix with blocks of equal size s. We define the diagonal entries of $\mathbf{D}$ such that we can control both the overall condition number c_o and the block condition number c_b of the Hessian matrix of (??).

In Figure ??, we plot the ERT of sep-CMA-ES while varying the block size s and the block condition number c_b of the function defined in (??). The graph for $s = 1$ serves as a baseline comparison to the case where no transformation matrix is applied (entries of the diagonal $\mathbf{B}$ are -1 and 1). There, the value of c_b is irrelevant.

We see a linear scaling between the ERT and the block condition number (compare with the dashed gray line). For $c_b > 10^4$, the algorithm does not manage to solve the problem. Low values of c_b (< 10) have little effect on the difficulty (except for $s = 2$).

The block condition number has a direct effect on the difficulty. Even without permutations, a high enough condition number results in a difficult problem for sep-CMA-ES, even though it manages to solve the non transformed version with the same overall condition number c_o.

E. ERT VS TRANSFORMED STANDARD FUNCTIONS

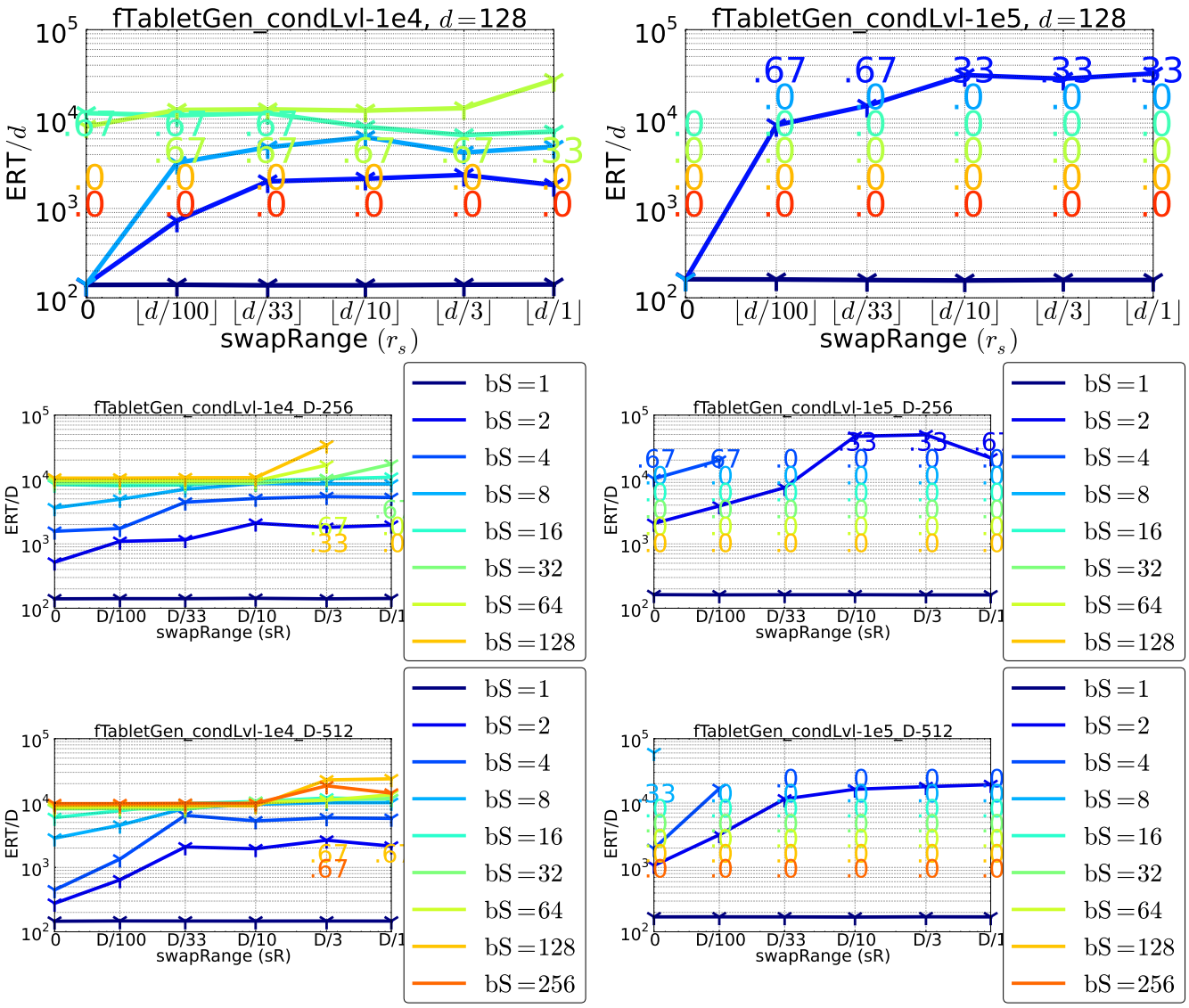
fTabletGen_condLvl-1e4, $d=128$
fTabletGen_condLvl-1e5, $d=128$
ERT/d
swapRange (r_s)
0
$\lfloor d/100 \rfloor$
$\lfloor d/33 \rfloor$
$\lfloor d/10 \rfloor$
$\lfloor d/3 \rfloor$
$\lfloor d/1 \rfloor$
fTabletGen_condLvl-1e4_D-256
fTabletGen_condLvl-1e5_D-256
fTabletGen_condLvl-1e4_D-512
fTabletGen_condLvl-1e5_D-512
ERT/D
swapRange (sR)
D/100
D/33
D/10
D/3
D/1
bS $=1$
bS $=2$
bS $=4$
bS $=8$
bS $=16$
bS $=32$
bS $=64$
bS $=128$
bS $=256$

Figure 8:

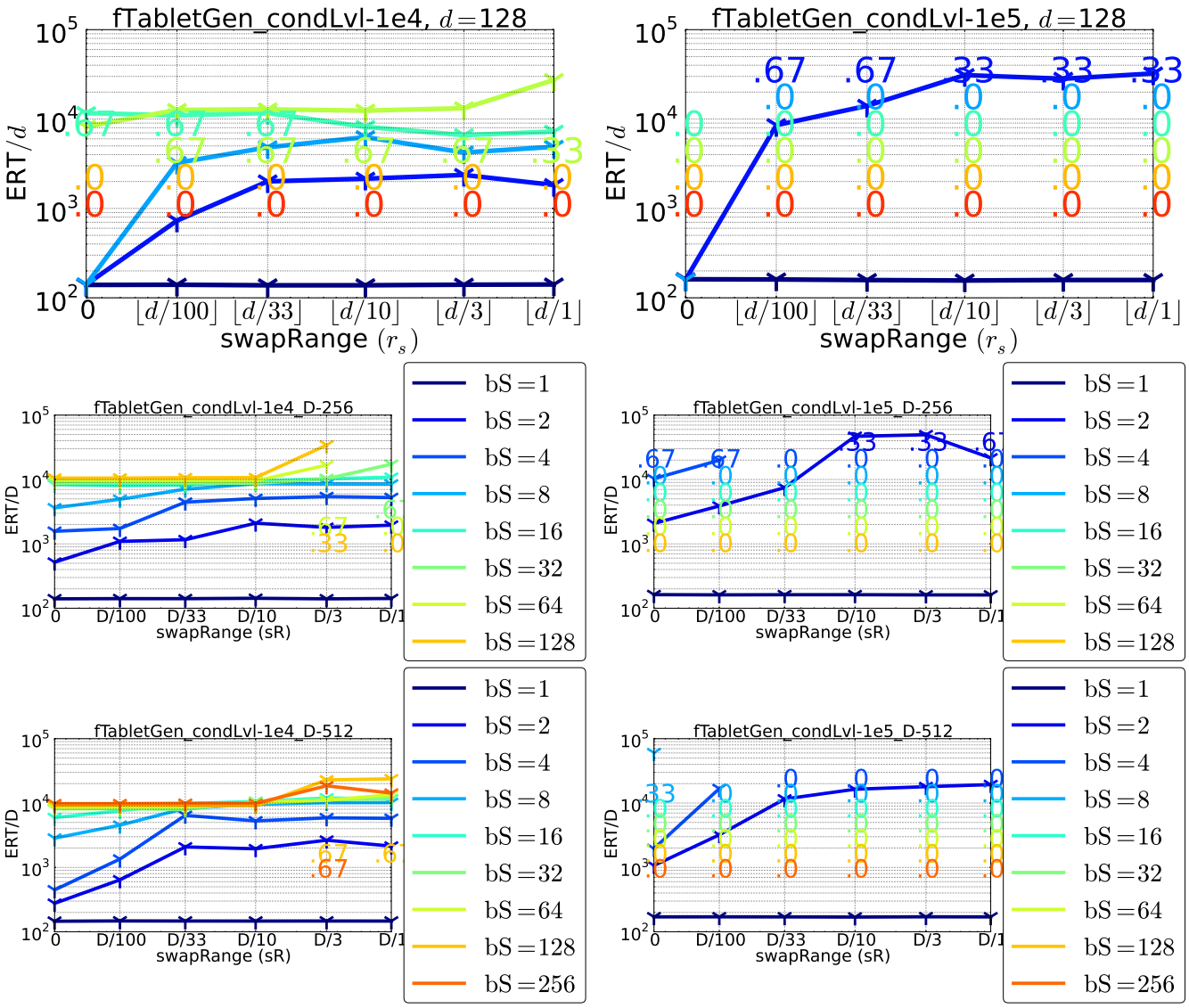
fTabletGen_condLvl-1e4, $d=128$
fTabletGen_condLvl-1e5, $d=128$
ERT/d
swapRange (r_s)
0
$\lfloor d/100 \rfloor$
$\lfloor d/33 \rfloor$
$\lfloor d/10 \rfloor$
$\lfloor d/3 \rfloor$
$\lfloor d/1 \rfloor$
fTabletGen_condLvl-1e4_D-256
fTabletGen_condLvl-1e5_D-256
fTabletGen_condLvl-1e4_D-512
fTabletGen_condLvl-1e5_D-512
ERT/D
swapRange (sR)
D/100
D/33
D/10
D/3
D/1
bS $=1$
bS $=2$
bS $=4$
bS $=8$
bS $=16$
bS $=32$
bS $=64$
bS $=128$
bS $=256$

Figure 9: