A Comparative Study of Large-Scale Variants of CMA-ES

Konstantinos Varelas[1,2(✉)], Anne Auger[1], Dimo Brockhoff[1], Nikolaus Hansen[1], Ouassim Ait ElHara[1], Yann Semet[3], Rami Kassab[2], and Frédéric Barbaresco[2]

[1] Inria, RandOpt team, CMAP, École Polytechnique, Palaiseau, France
{konstantinos.varelas,anne.auger,dimo.brockhoff,nikolaus.hansen, ouassim.elHara}@inria.fr
[2] Thales LAS France SAS - Limours, Limours, France
[3] Thales Research Technology, Palaiseau, France
yann.semet@thalesgroup.com

Abstract. The CMA-ES is one of the most powerful stochastic numerical optimizers to address difficult black-box problems. Its intrinsic time and space complexity is quadratic—limiting its applicability with increasing problem dimensionality. To circumvent this limitation, different large-scale variants of CMA-ES with subquadratic complexity have been proposed over the past ten years. To-date however, these variants have been tested and compared only in rather restrictive settings, due to the lack of a comprehensive large-scale testbed to assess their performance. In this context, we introduce a new large-scale testbed with dimension up to 640, implemented within the COCO benchmarking platform. We use this testbed to assess the performance of several promising variants of CMA-ES and the standard limited-memory L-BFGS. In all tested dimensions, the best CMA-ES variant solves more problems than L-BFGS for larger budgets while L-BFGS outperforms the best CMA-ES variant for smaller budgets. However, over all functions, the cumulative runtime distributions between L-BFGS and the best CMA-ES variants are close (less than a factor of 4 in high dimension).

Our results illustrate different scaling behaviors of the methods, expose a few defects of the algorithms and reveal that for dimension larger than 80, LM-CMA solves more problems than VkD-CMA while in the cumulative runtime distribution over all functions the VkD-CMA dominates LM-CMA for budgets up to 10^4 times dimension and for all budgets up to dimension 80.

1 Introduction

The CMA-ES is a stochastic derivative-free optimization algorithm, recognized as one of the most powerful optimizers for solving difficult black-box optimization problems, i.e., non-linear, non quadratic, non-convex, non-smooth, and/or noisy problems [6]. Its intrinsic complexity in terms of memory and internal computational effort is quadratic in the dimensionality, n, of the black-box objective function to be solved, denoted in a generic manner as: $f : x \in \mathbb{R}^n \mapsto \mathbb{R}$. This complexity restricts its application when the number n of variables is in the order of a few hundred. For this reason, different "large"-scale variants of CMA-ES have been introduced over the past ten years. They all aim at a sub-quadratic space and time complexity [3,7,9,11,12,14,15]. The common feature of the variants is to restrict the model of the covariance matrix and provide a sparse representation that can be stored, sampled and updated in $\mathcal{O}(n \times m)$ operations with $m \ll n$. Yet the approaches to do so are quite different. On the one-hand, the seminal limited memory BFGS, L-BFGS [10], inspired the introduction of the limited memory CMA (LM-CMA, [11,12]) where the main idea is to approximate at iteration $t \gg m$ the sum over t terms composing the covariance matrix by a sum over m terms. This same approach is used in the RmES algorithm [9]. On the other-hand, the sep-CMA [14] and VkD-CMA [3] algorithms enforce a predefined structure of the covariance matrix (for instance diagonal for the sep-CMA) and project at each iteration the updated matrix onto the restricted space.

After designing a novel algorithm, the next step is to assess its performance and compare it with its competitors. This benchmarking step is crucial but is known to be non-trivial and tedious. For this reason, during the past ten years, an important effort went into the development of the COCO platform to introduce a thorough benchmarking methodology and to automatize the tedious benchmarking process [4]. With COCO, algorithms are put at a standardized test and performance assessment is greatly facilitated as users can download and compare datasets of 180+ previously benchmarked algorithms.[1]

Yet so far, the testbeds provided with COCO are not suitable for benchmarking large-scale algorithms. One bottleneck with the current suite is the use of full orthogonal matrices with n^2 coefficients in the definition of many of the functions which makes the computation too expensive for thorough benchmarking studies. For this reason, it was proposed to replace these matrices by orthogonal matrices with a sparse structure: permuted block-diagonal matrices [1]. We utilize this idea to introduce a large-scale test suite with search space dimensions from 20 up to 640.

In this context, the main contributions of this paper are (i) the introduction of a large-scale testbed within the COCO framework and (ii) the comparative review and performance assessment of the currently most promising large-scale variants of CMA-ES and their comparison to the well established L-BFGS algorithm. Besides the general performance quantification and comparison, the benchmarking allows to identify defects of the algorithms or of their implementations (that shall be fixed in the near future).

[1] All raw datasets are available for download at http://coco.gforge.inria.fr/doku.php?id=algorithms while already postprocessed results are available (without the need to install COCO) at http://coco.gforge.inria.fr/ppdata-archive.

2 The bbob-Largescale COCO Testbed

Performance assessment is a crucial part of algorithm design and an important aspect when recommending algorithms for practical use. Choosing a representative testbed and setting up an assessment methodology are non-trivial and tedious. Hence, it is desirable to automatize the assessment in a standardized benchmarking process. In recent years, the Comparing Continuous Optimizers platform (COCO, [4]) has been developed particularly for this purpose and became a quasi-standard in the optimization community in the case of medium-scale unconstrained black-box optimization. The specific aspects of the COCO platform are: (i) a *quantitative performance assessment* by reporting runlengths which results in a *budget-free* experimental setting, (ii) *fully scalable* test functions in standard dimensions 2–40, (iii) *full automation* of the experiments with example code in C/C++, Java, MATLAB/Octave, python, and R, (iv) the availability of (pseudo-random) *instances of parametrized functions* which allow to naturally compare deterministic and stochastic algorithms, (v) *extensive post-processing functionalities* to visualize and analyze the experimental data, and finally, (vi) a *large amount of publicly available results* to compare with (from running, so far, 180+ algorithm implementations).

Each test problem in the COCO platform comes in the form of instances which are constructed in an “onion-style” through basic pseudo-random transformations of a raw function: $f(x) = H_1 \circ \ldots \circ H_{k_1}(f_{\text{raw}}(T_1 \circ \ldots \circ T_{k_2}(x)))$, where f_{raw} is the underlying raw function—usually the simplest representative of the function class (like the sphere function with optimum in zero). The $T_i : \mathbb{R}^n \to \mathbb{R}^n$ are search space transformations and $H_i : \mathbb{R} \to \mathbb{R}$ are function value transformations. Examples of the former are rotations of the search space or translations of the optimum. An example of the latter are strictly increasing (monotone) functions. The transformations applied to the raw function are actually (pseudo)-random, rendering an *instance* of a parametrized transformation [4].

All currently available test suites of COCO such as the noiseless, single-objective `bbob` suite with its 24 functions [5] are scalable in the problem dimension and could be used for benchmarking in a large-scale setting. However, their internal computation scales quadratically with the dimension due to the search space rotations applied in most functions—rendering the experiments in higher dimension too costly to be practicable. Also, real-world problems in higher dimension will, most likely, not have quadratically many degrees of freedom. In consequence, artificial test functions, that aim at capturing the typical real-world challenges, shall likely also not have quadratically many internal parameters.

In [1], the authors therefore suggest search space rotations that have linear internal computation costs by being less “rich” than the rotation matrices of the standard `bbob` test suite. Full rotation matrices $\mathbf{R}$ are replaced by a sequence of three matrices $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$ in which $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are permutation matrices (with exactly one “1” per row and column) and $\mathbf{B}$ is an orthogonal block-diagonal matrix. The permutation matrices $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are constructed by n_s so-called *truncated uniform swaps* [1]: Each swap chooses a first variable i uniform at random and the second variable within the vicinity of the first variable, i.e., uniformly at random within the set $\{lb(i), \ldots, ub(i)\}$ with $lb(i) = \max(1, i - r_s)$ and $ub(i) = \min(n, i + r_s)$ and where r_s is a parameter indicating the distance *range* between the two swapped variables. The computation of $\mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}$ can be done in linear time, see [1] for details.

In this paper, we introduce the new large-scale variant of the standard `bbob` test suite of COCO, denoted as `bbob-largescale`, based on the above ideas. Implemented in the COCO platform[2], it is built on the same 24 raw functions of `bbob` with the default dimensions 20, 40, 80, 160, 320, and 640—the first two overlapping with the original `bbob` suite for compatibility and consistency reasons. The full rotation matrices of the `bbob` suite are replaced by the above construction of permutation and block matrices. Following the recommendations of [1], we chose to do $n_s = n$ swaps with a range of $r_s = \lfloor n/3 \rfloor$ and to have all blocks of the same size of $\min\{40, n\}$ except for the last, possibly smaller block.

One additional change concerns functions with distinct axes: three of the `bbob` functions, namely the Discus, the Sharp Ridge and the Bent Cigar function, have been modified in order to have a constant proportion of distinct axes when the dimension increases [1].

All function instances have their (randomly chosen) global optimum in $[-5, 5]^n$ and for all but the linear function also the entire (hyper-)ball of radius 1 with the optimum as center lies within this range. Except for the Schwefel, Schaffer, Weierstrass, Gallagher and Katsuura functions, the function value is corrected by $\min\{1, 40/n\}$ to make the target values comparable over a large range of dimensions. The optimal function value offset is randomly drawn between -1000 and 1000.

Compared to the CEC'08 testbed [17], the `bbob-largescale` test suite has a wider range of problems and difficulties, allows to investigates scaling, applies various regularity-breaking transformations and provides pseudo-random instances to compare naturally deterministic and stochastic algorithms.

3 The CMA-ES Algorithm and Some Large-Scale Variants

We introduce in this section the CMA-ES algorithm and give then an overview of large-scale variants that have been introduced in recent years, with an emphasis on the variants that are later empirically investigated.

3.1 The $(\mu/\mu_w, \lambda)$-CMA-ES

The $(\mu/\mu_w, \lambda)$-CMA-ES algorithm samples $\lambda \geq 2$ candidate solutions from a multivariate normal distribution $\mathcal{N}(\mathbf{m}_t, {\sigma_t}^2\mathbf{C}_t)$ where the mean $\mathbf{m}_t \in \mathbb{R}^n$ is the incumbent solution, σ_t is a scalar referred to as step-size and $\mathbf{C}_t \in \mathbb{R}^{n \times n}$ is a positive definite covariance matrix. The algorithm adapts mean, step-size and

[2] The source code of the new test suite (incl. adaptations in COCO's postprocessing) can be found in the devel-LS-development branch of the COCO Github page.

covariance matrix so as to learn second order information on convex-quadratic functions. The CMA-ES is hence a stochastic counterpart of quasi-Newton methods like the BFGS algorithm [10].

The sampling of the candidate solutions $(\mathbf{x}_t^i)_{1\le i\le\lambda}$ is typically done by computing the eigen-decomposition of the covariance matrix as $\mathbf{C}_t = \mathbf{B}_t\mathbf{D}_t^2\mathbf{B}_t^\top$ where $\mathbf{B}_t$ contains an orthonormal basis of eigenvectors, and $\mathbf{D}_t$ is a diagonal matrix containing the square roots of the corresponding eigenvalues. The square root of $\mathbf{C}_t$ is computed as $\mathbf{C}_t^{1/2} = \mathbf{B}_t\mathbf{D}_t\mathbf{B}_t^\top$ and used for sampling the candidate solutions as $\mathbf{x}_t^i = \mathbf{m}_t + \sigma_t\mathbf{C}_t^{1/2}\mathbf{z}_t^i$ with $\mathbf{z}_t^i \sim \mathcal{N}(0,\mathbf{I})$, where $\mathcal{N}(0,\mathbf{I})$ denotes a multivariate normal distribution with mean zero and covariance matrix identity. The eigendecomposition has a complexity of $\mathcal{O}\left(n^3\right)$ but is done only every $\mathcal{O}(n)$ evaluations (*lazy*-update) reducing the complexity of the sampling to $\mathcal{O}(n^2)$.

The candidate solutions are then evaluated on f and ranked from the best to the worse, $f(\mathbf{x}_t^{1:\lambda}) \le \ldots \le f(\mathbf{x}_t^{\lambda:\lambda})$. Mean, step-size and covariance matrix are then updated using the ranked solutions. More precisely the new mean equals $\mathbf{m}_{t+1} = \sum_{i=1}^{\mu} w_i\mathbf{x}_t^{i:\lambda}$ where μ (typically) equals $\lfloor\lambda/2\rfloor$ and w_i are weights satisfying $w_1 \ge w_2 \ge \ldots \ge w_\mu > 0$. Two mechanisms exist to update the covariance matrix, namely the rank-one and rank-mu update. The rank one update adds the rank-one matrix $\mathbf{p}_{t+1}^c[\mathbf{p}_{t+1}^c]^\top$ to the current covariance matrix, where $\mathbf{p}_{t+1}^c$ is the evolution path and defined as $\mathbf{p}_{t+1}^c = (1-c_c)\,\mathbf{p}_t^c + \sqrt{c_c\,(2-c_c)\,\mu_{\text{eff}}}\,(\mathbf{m}_{t+1} - \mathbf{m}_t)/\sigma_t$, with $\mu_{\text{eff}} = 1/\sum_{i=1}^{\mu} w_i^2$ and $c_c < 1$. The rank-mu update adds the matrix $\mathbf{C}_{t+1}^{\mu} = \sum_{i=1}^{\mu} w_i\mathbf{z}_t^{i:\lambda}[\mathbf{z}_t^{i:\lambda}]^\top$ with $\mathbf{z}_t^{i:\lambda} = (\mathbf{x}_t^{i:\lambda} - \mathbf{m}_t)/\sigma_t$ such that overall the update of the covariance matrix reads

$$\mathbf{C}_{t+1} = (1 - c_1 - c_\mu)\,\mathbf{C}_t + c_1\mathbf{p}_{t+1}^c[\mathbf{p}_{t+1}^c]^\top + c_\mu\mathbf{C}_{t+1}^{\mu} \tag{1}$$

where c_1, c_μ belong to $(0,1)$. The step-size is updated using the *Cumulative step-size adaptation* (CSA) that utilizes an evolution path cumulating steps of the mean in the isotropic coordinate system with principal axes of $\mathbf{C}_t$: $\mathbf{p}_{t+1}^\sigma = (1-c_\sigma)\,\mathbf{p}_t^\sigma + \sqrt{c_\sigma\,(2-c_\sigma)\,\mu_{\text{eff}}}\mathbf{C}_t^{-\frac{1}{2}}\frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$, where $c_\sigma < 1$ and compares the length of the evolution path with its expected length under random selection in order to increase the step-size when the first is larger, or decrease it otherwise. The update of the step-size reads $\sigma_{t+1} = \sigma_t \exp(\frac{d_\sigma}{c_\sigma}(\|\mathbf{p}_{t+1}^\sigma\|/E[\|\mathcal{N}(0,\mathbf{I})\|]))$ with $d_\sigma > 0$. Remark that the computation of $\mathbf{C}_t^{-\frac{1}{2}}$ is immediate via $\mathbf{C}_t^{-\frac{1}{2}} = \mathbf{B}_t\mathbf{D}_t^{-1}\mathbf{B}_t^\top$ (it is done at the same time than the eigendecomposition of $\mathbf{C}_t$ every $\mathcal{O}(n)$ iterations with a complexity of $\mathcal{O}(n^3)$).

Cholesky-CMA. An alternative to the previous algorithm was proposed in [16]. Instead of using the eigendecomposition of the covariance matrix to sample candidate solutions, it uses a decomposition of $\mathbf{C}_t$ as $\mathbf{C}_t = \mathbf{A}_t\mathbf{A}_t^\top$. Indeed assume that $\mathbf{A}_t$ is known, then sampling $\mathbf{x}_i^t$ as $\mathbf{m}_t + \sigma_t\mathbf{A}_t\mathbf{z}_t^i$ with $\mathbf{z}_t^i \sim \mathcal{N}(0,\mathbf{I})$ results in a vector following $\mathcal{N}(\mathbf{m}_t, \sigma_t^2\mathbf{C}_t)$. When $\mathbf{A}_t$ is lower (or upper) triangular the decomposition is unique and called Cholesky factorization. However, in [16] the term Cholesky factorization is used without assuming that the matrix $\mathbf{A}_t$ is triangular. We will continue to use Cholesky-CMA for the ensuing algorithm to be consistent with the previous algorithm name.

The key idea for the Cholesky-CMA is that instead of adapting the covariance matrix $\mathbf{C}_t$, the Cholesky factor $\mathbf{A}_t$ is directly updated (and hence sampling does not require factorization a matrix). The method solely conducts the rank-one update of the covariance matrix, $\mathbf{C}_{t+1} = (1 - c_1)\,\mathbf{C}_t + c_1 \mathbf{p}^c_{t+1}[\mathbf{p}^c_{t+1}]^\top$, by updating the matrix $\mathbf{A}_t$ such that $\mathbf{C}_{t+1} = \mathbf{A}_{t+1}\mathbf{A}_{t+1}^\top$. Indeed, let $\mathbf{v}_{t+1}$ be defined implicitly via $\mathbf{A}_t \mathbf{v}_{t+1} = \mathbf{p}^c_{t+1}$, then the update of $\mathbf{A}_t$ reads

$$\mathbf{A}_{t+1} = \sqrt{1-c_1}\mathbf{A}_t + \frac{\sqrt{1-c_1}}{\|\mathbf{v}_{t+1}\|^2}\left(\sqrt{1+\frac{c_1}{1-c_1}\|\mathbf{v}_{t+1}\|^2} - 1\right)\mathbf{p}^c_{t+1}\mathbf{v}_{t+1}{}^\top , \qquad (2)$$

if $\mathbf{v}_{t+1} \neq \mathbf{0}$ and $\mathbf{A}_{t+1} = \sqrt{1-c_1}\mathbf{A}_t$ if $\mathbf{v}_{t+1} = \mathbf{0}$ (see [16, Theorem 1]). A similar expression holds for the inverse $\mathbf{A}_{t+1}^{-1}$ (see [16, Theorem 2]). Sampling of a multivariate normal distribution using the Cholesky factor still requires $\mathcal{O}(n^2)$ operations due to the matrix-vector multiplication. However, the Cholesky-CMA has been used as foundation to construct numerically more efficient algorithms as outlined below. Recently, a version of CMA using Cholesky factorization enforcing triangular shapes for the Cholesky factors has been proposed [8].

3.2 Large-Scale Variants of CMA-ES

The quadratic time and space complexity of CMA-ES (both the original and Cholesky variant) becomes critical with increasing dimension. This has motivated the development of large-scale variants with less rich covariance models, i.e., with $o(n^2)$ parameters. Reducing the number of parameters reduces the memory requirements and, usually, the internal computational effort, because fewer parameters must be updated. It also has the advantage that learning rates can be increased. Hence, learning of parameters can be achieved in fewer *number of evaluations*. Given the model is still rich enough for the problem at hand, this further reduces the computational costs to solve it in particular even when the f-computation dominates the overall costs. Hence, in the best case scenario, reducing the number of parameters from n^2 to n reduces the time complexity to solve the problem from n^2 to n if f-computations dominate the computational costs and from n^4 to n^2 if internal computations dominate.

We review a few large-scale variants focussing on those benchmarked later in the paper.

sep-CMA-ES [14]. The separable CMA-ES restricts the full covariance matrix to a diagonal one and thus has a linear number of parameters to be learned. It loses the ability of learning the dependencies between decision variables but allows to exploit problem separability. The sep-CMA-ES achieves linear space and time complexity.

VkD-CMA-ES [2,3]. A richer model of the covariance matrix is used in the VkD-CMA-ES algorithm where the eligible covariance matrices are of the form $\mathbf{C}_t = \mathbf{D}_t(\mathbf{I} + \mathbf{V}_t\mathbf{V}_t^\top)\mathbf{D}_t$ where $\mathbf{D}_t$ is a n-dimensional positive definite diagonal matrix and $\mathbf{V}_t = [\mathbf{v}_t^1 \ldots \mathbf{v}_t^k]$ where $\mathbf{v}_t^i \in \mathbb{R}^n$ are orthogonal vectors [3]. The parameter k ranges from 0 to $n-1$: when $k = 0$ the method recovers the separable CMA-ES while for $k = n-1$ it recovers the (full)-CMA-ES algorithm. The elements of $\mathbf{C}_{t+1}$ are determined by projecting the covariance matrix updated by CMA-ES given in (1) denoted as $\hat{\mathbf{C}}_{t+1}$ onto the set of eligible matrices. This projection is done by approximating the solution of the problem $\underset{(\mathbf{D},\mathbf{V})}{\operatorname{argmin}} \|\mathbf{D}\left(\mathbf{I}+\mathbf{V}\mathbf{V}^{\top}\right)\mathbf{D} - \hat{\mathbf{C}}_{t+1}\|_F$ where $\|\cdot\|_F$ stands for the Frobenius norm. This projection can be computed without computing $\hat{\mathbf{C}}_{t+1}$. The space complexity of VkD-CMA-ES is $\mathcal{O}(nr)$ and the time complexity is $\mathcal{O}(nr \max(1, r/\lambda))$, where $r = k + \mu + \lambda + 1$. Note that the algorithm exploits both the rank-one and rank-mu update of CMA-ES as the projected matrices result from the projection of the matrix $\hat{\mathbf{C}}_{t+1}$ updated with both updates.

A procedure for the online adaptation of k has been proposed in [2]. It tracks in particular how the condition number of the covariance matrix varies with changing k. The variant with the procedure of online adaptation of k as well as with fixed $k = 2$ is benchmarked in the following. The VkD-CMA algorithm uses *Two Point Adaptation* (TPA) to adapt the step-size. The TPA is based on the ranking difference between two symmetric points around the mean along the previous mean shift.

The limited-memory (LM) CMA [11,12]. The LM-CMA is inspired by the gradient based limited memory BFGS method [10] and builds on the Cholesky CMA-ES. If $\mathbf{A}_0 = \mathbf{I}$, setting $a = \sqrt{1-c_1}$ and $b_t = \frac{\sqrt{1-c_1}}{\|\mathbf{v}_{t+1}\|^2}\left(\sqrt{1+\frac{c_1}{1-c_1}\|\mathbf{v}_{t+1}\|^2} - 1\right)$, then (2) can be re-written as $\mathbf{A}_{t+1} = a^t\mathbf{I} + \sum_{i=1}^{t} a^{t-i} b_{i-1} \mathbf{p}_i^c \mathbf{v}_i^{\top}$. This latter equation is approximated by taking m elements in the sum instead of t. Initially, m was proposed to be fixed to $\mathcal{O}(\log(n))$. Later, better performance has been observed with m in the order of $\sqrt{n}$ [11], imposing $\mathcal{O}(n^{3/2})$ computational cost. Sampling can be done without explicitly computing $\mathbf{A}_{t+1}$ and the resulting algorithm has $\mathcal{O}(mn)$ time and space complexity. The choice of the m elements of the sum to approximate $\mathbf{A}_{t+1}$ seems to be essential. In L-BFGS the last m iterations are taken while for LM-CMA the backward $N_{\text{steps}} \times k$ iterations for $k = 0, \ldots, m-1$ are considered (that is we consider the current iteration, the current iteration minus N_{steps} and so on). The parameter N_{steps} is typically equal to n. Since $\mathbf{A}_t \mathbf{v}_{t+1} = \mathbf{p}_{t+1}^c$, the inverse factor $\mathbf{A}_t^{-1}$ is employed for the computation of $\mathbf{v}_{t+1}$, but an explicit computation is not needed, similarly as for $\mathbf{A}_t$. To adapt the step-size, the LM-CMA uses the *population success rule* (PSR) [12].

A variant of LM-CMA was recently proposed, the LM-MA, which is however not tested here because (i) the code is not available online and (ii) the performance of LM-MA seems not to be superior to LM-CMA [13].

The RmES [9]. The idea for the RmES algorithm is similar to the LM-CMA algorithm. Yet, instead of using the Cholesky-factor, the update of $\mathbf{C}_t$ is considered. Similarly as for LM-CMA, if $\mathbf{C}_0 = \mathbf{I}$ and solely the rank-one update is used for CMA-ES we can write the update as $\mathbf{C}_t = (1-c_1)^m \mathbf{I} + c_1 \sum_{i=1}^{m} (1-c_1)^{m-i} \hat{\mathbf{p}}_i^c \hat{\mathbf{p}}_i^{c\top}$. In RmES, m terms of the sum are considered and $m = 2$ is advocated. Additionally, like in LM-CMA, the choice of terms entering the sum is by maintaining a temporal distance between generations. Sampling of new solutions is done from the m vectors without computing the covariance matrix explicitly. The RmES adapts the step-size similarly to PSR.

A main difference to LM-CMA is that RmES is formulated directly on the covariance matrix, thus an inverse Cholesky factor is not needed. This does not improve the order of complexity, though, which is $\mathcal{O}(mn)$ as in LM-CMA.

The presented algorithms do not of course form an exhaustive list of proposed methods for large-scale black-box optimization. We refer to [13] for a more thorough state-of-the-art and point out that our choice is driven by variants that currently appear to be the most promising or by variants like sep-CMA, important to give baseline performance.

4 Experimental Results

We assess the performance of implementations of the algorithms presented in the previous section on the `bbob-largescale` suite. We are particularly interested to identify the scaling of the methods, possible algorithm defects, and to quantify the impact of population size. Because we benchmark algorithm *implementations*, as opposed to mathematical algorithms, observations may be specific to the investigated implementation only.

Experimental Setup. We run the algorithms sep-CMA, LM-CMA, VkD-CMA, RmES on the default `bbob` test suite in dimensions $2, 3, 5, 10$ and on the proposed `bbob-largescale` suite implemented in COCO. Additionally, we run the limited memory BFGS, L-BFGS, still considered as the state-of-the-art algorithm for gradient based optimization [10]. Gradients are estimated via finite-differences.

For VkD-CMA, the Python implementation from pycma, version 2.6.0, was used, for sep-CMA the version from sites.google.com/site/ecjlmcma, and for L-BFGS the optimization toolbox of scipy 0.12.1. We consider two versions of LM-CMA provided by the author at sites.google.com/site/ecjlmcma and .../lmcmaeses related to the articles [12] denoted LM-CMA'14 and [11] denoted LM-CMA. The implementation of RmES was kindly provided by its authors [9].

Experiments were conducted with default[3] parameter values of each algorithm and a maximum budget of $5 \cdot 10^4 n$. Automatic restarts are conducted once a default stopping criterion is met until the maximum budget is reached. For each function, fifteen instances are presented. For the first run and for all (automatic) restarts, the initial point was uniform at random between $[-4, 4]^n$ for all algorithms, while the initial step-size was set to 2 for all CMA variants.

For LM-CMA, sep-CMA and RmES, population sizes of $4 + \lfloor 3 \log n \rfloor$, $2n + \lfloor 10/n \rfloor$ and $10n$ were tested and the experiments were conducted for the same budget and instances. A suffix P2 (P10) is used to denote the respective algorithms. For VkD-CMA, a second experiment has been run where the number of vectors was fixed to $k = 2$, denoted as V2D-CMA.

[3] Except L-BFGS, where the `factr` parameter was set to `1.0` for very high precision.

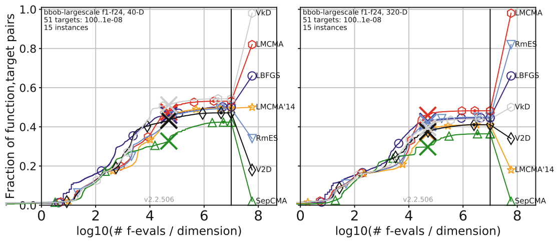

Fig. 1. Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for all functions in 40-D (left) and 320-D.

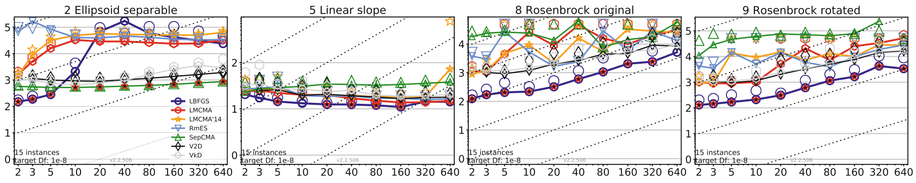

Fig. 2. Scaling graphs: Average Runtime (aRT) divided by dimension to reach a target of 10^{-8} versus dimension for selected functions. Light symbols give the maximum number of evaluations from the longest trial divided by dimension.

Performance assessment. We measure the number of function evaluations to reach a specified target function value, denoted as *runtime*, RT. The average runtime, aRT, for a single function and target value is computed as the sum of all evaluations in unsuccessful trials plus the sum of runtimes in all successful trials, both divided by the number of successful trials. For Empirical Cumulative Distribution Functions (ECDF) and in case of unsuccessful trials, runtimes are computed via simulated restarts [4] (bootstrapped ECDF). The *success rate* is the fraction of solved problems (function-target pairs) under a given budget as denoted by the y-axis of ECDF graphs. Horizontal differences between ECDF graphs represent runtime ratios to solve the same respective fraction of problems (though not necessarily the same problems) and hence reveal how much faster or slower an algorithm is.

Overview. A complete presentation of the experimental results is available at cocoexprm.gforge.inria.fr. Figure 1 presents for each algorithm the runtime distribution aggregated over all functions. Overall, the distributions look surprisingly similar in particular in larger dimension. After $5{\cdot}10^4 n$ evaluations in 320-D, between 30% (sepCMA) and 46% (LMCMA) of all problems have been solved. In all dimensions, for a restricted range of budgets, the success rate of L-BFGS is superior to all CMA variants. The picture becomes more diverse with increasing budget where L-BFGS is outperformed by CMA variants. We emphasize that even domination over the entire ECDF does not mean that the algorithm is

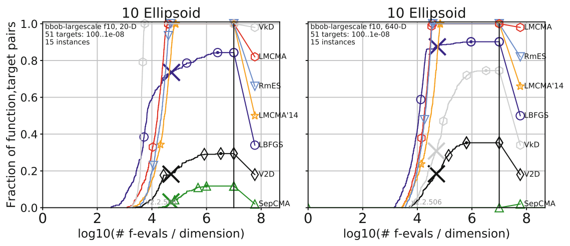

Fig. 3. Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for the ellipsoid function in 20-D and 640-D.

faster on every single problem, because runtimes are shown in increasing order *for each algorithm*, hence the order of problems as shown most likely differs.

Up to a budget of $10^4 n$, the performance similarity between LM-CMA and RmES is striking. The performance is almost identical on the Sphere, Ellipsoid, Linear Slope and Sum of Different Powers functions in dimensions equal or larger to 20. On the Bent Cigar function in dimensions greater or equal to 80 and for a budget larger than $10^4 n$, LM-CMA is notably superior to RmES.

Scaling with dimension. Fig. 2 shows the average runtime scaling with dimension on selected functions. On the separable Ellipsoid for $n \geq 20$ sep-CMA with population size $\geq 2n$ (not shown in Fig. 2) and VkD scale worse than linear. Starting from dimension 20, LM-CMA and RmES show runtimes of aRT $\approx$ 2–7 $\times 10^4 n$. With default population size, sep-CMA performs overall best and is for $n \geq 20$ even more than twenty times faster than L-BFGS. The latter scales roughly quadratically for small dimensions and (sub-)linear (with a much larger coefficient) for large dimensions. This behavior is a result of a transition when the dimension exceeds the rank (here 10) of the stored matrix. On the linear function, algorithms scale close to linear with a few exceptions. With population size $2n + \lfloor 10/n \rfloor$ or larger (not shown in Fig. 2), the scaling becomes worse in all cases (which means a constant number of iterations is not sufficient to solve the "linear" problem). In particular, sep-CMA reveals in this case a performance defect due to a diverging step-size (which disappears with option `'AdaptSigma': 'CMAAdaptSigmaTPA'`), as verified with single runs. On both Rosenbrock functions, L-BFGS scales roughly quadratically.

Restricting the model. The particular case of the ill-conditioned non-separable ellipsoidal function in Fig. 3 illustrates interesting results: in 20D, VkD-CMA solves the function, i.e. reaches the best target value faster (by a factor of 10 at least) than any other method. In 640-D any other CMA variant with default parameter values except sep-CMA outperforms it.

On the Ellipsoid function only VkD-CMA scales quadratically with the dimension. All other algorithms either scale linearly or do not solve the problem for larger dimension. On the Discus function (with a fixed proportion of

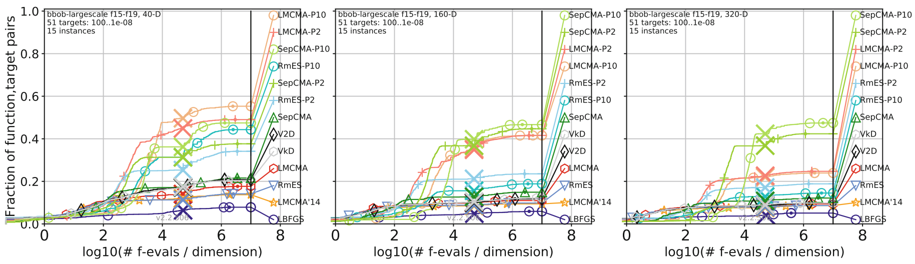

Fig. 4. Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for the group of multimodal functions with adequate structure in 40-D (left), 160-D (middle) and 320-D (right).

short axes), VkD-CMA slows down before to reach the more difficult targets and exhausts the budget. An unusual observation is that LM-CMA performs considerably better on the Attractive Sector function in the smallest *and largest dimensions*. We do not see this effect on LM-CMA'14, where the choice of the number of the direction vectors is smaller and random. Thus, these effects indicate the importance of properly choosing m [12]. Even though the covariance matrix model provided by VkD-CMA is richer, the method is outperformed by RmES and LM-CMA, e.g. on the Discus and Ellipsoid functions in dimension greater than 80. This suggests that k is adapted to too large values thereby impeding the learning speed of the covariance matrix.

Fixed versus adapted k. In order to investigate the effect of k-adaptation, we compare VkD-CMA with adaptive and fixed $k = 2$. Only in few cases the latter shows better performance. This is in particular true for the intrinsically not difficult to solve Attractive Sector function, indicating that the procedure of k adaptation could impose a defect.

Impact of population size. In Fig. 4, the effect of larger populations is illustrated for the multimodal functions with adequate global structure. The CMA variants with default population size and L-BFGS are clearly outperformed, solving less than half as many problems. That is, increased population size variants reach better solutions. Yet, the overall performance drops notably with increasing dimension. As expected, on the weakly-structured multimodal functions `f20-f24`, larger populations do not achieve similar performance improvements.

5 Discussion and Conclusion

This paper has (i) introduced a novel large-scale testbed for the COCO platform and (ii) assessed the performance of promising large-scale variants of CMA-ES compared to the quasi-Newton L-BFGS algorithm. We find that in all dimensions, L-BFGS generally performs best with lower budgets and is outperformed by CMA variants as the budget increases. On multi-modal functions with global structure, CMA-ES variants with increased population size show the expected decisive advantage over L-BFGS. For larger dimension, the performance on these multi-modal functions is however still unsatisfying. The study has revealed some potential defects of algorithms (k-adaptation in VkD-CMA on the Attractive Sector, Ellipsoid and Discus) and has confirmed the impact and criticality of the choice of the m parameter in LM-CMA. The VkD-CMA that appears to be a more principled approach and includes a diagonal component and the rank-μ update of the original CMA-ES, overall outperforms LM-CMA and RmES in smaller dimension, while LM-CMA overtakes for the large budgets in larger dimensions. On single functions, the picture is more diverse, suggesting possible room for improvement in limited memory and VkD-CMA approaches.

Acknowledgement. The PhD thesis of Konstantinos Varelas is funded by the French MoD DGA/MRIS and Thales Land & Air Systems.

References

1. Ait ElHara, O., Auger, A., Hansen, N.: Permuted orthogonal block-diagonal transformation matrices for large scale optimization benchmarking. In: Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 189–196. ACM (2016)
2. Akimoto, Y., Hansen, N.: Online model selection for restricted covariance matrix adaptation. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 3–13. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_1
3. Akimoto, Y., Hansen, N.: Projection-based restricted covariance matrix adaptation for high dimension. In: Genetic and Evolutionary Computation Conference (GECCO 2016), pp. 197–204. Denver, USA, July 2016
4. Hansen, N., Auger, A., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: A platform for comparing continuous optimizers in a black-box setting (2016). arXiv:1603.08785
5. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Research Report RR-6829, INRIA (2009)
6. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
7. Knight, J.N., Lunacek, M.: Reducing the space-time complexity of the CMA-ES. In: Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 658–665. ACM (2007)
8. Krause, O., Arbonès, D.R., Igel, C.: CMA-ES with optimal covariance update and storage complexity. In: NIPS Proceedings (2016)
9. Li, Z., Zhang, Q.: A simple yet efficient evolution strategy for large scale black-box optimization. IEEE Trans. Evol. Comput. (2017, accepted)
10. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**(3), 503–528 (1989)
11. Loshchilov, I.: LM-CMA: an alternative to L-BFGS for large scale black-box optimization. Evol. Comput. **25**, 143–171 (2017)

12. Loshchilov, I.: A computationally efficient limited memory CMA-ES for large scale optimization. In: Genetic and Evolutionary Computation Conference (GECCO 2014), pp. 397–404 (2014)
13. Loshchilov, I., Glasmachers, T., Beyer, H.: Limited-memory matrix adaptation for large scale black-box optimization. CoRR abs/1705.06693 (2017)
14. Ros, R., Hansen, N.: A simple modification in CMA-ES achieving linear time and space complexity. In: Rudolph, G., Jansen, T., Beume, N., Lucas, S., Poloni, C. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 296–305. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87700-4_30
15. Sun, Y., Gomez, F.J., Schaul, T., Schmidhuber, J.: A linear time natural evolution strategy for non-separable functions. CoRR abs/1106.1998 (2011)
16. Suttorp, T., Hansen, N., Igel, C.: Efficient covariance matrix update for variable metric evolution strategies. Mach. Learn. **75**(2), 167–197 (2009)
17. Tang, K., et al.: Benchmark functions for the CEC 2008 special session and competition on large scale global optimization (2007)