A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization

Ilya Loshchilov

A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization

Ilya Loshchilov
Laboratory of Intelligent Systems
École Polytechnique Fédérale de Lausanne, Switzerland
ilya.loshchilov@epfl.ch

ABSTRACT

We propose a computationally efficient limited memory Covariance Matrix Adaptation Evolution Strategy for large scale optimization, which we call the LM-CMA-ES. The LM-CMA-ES is a stochastic, derivative-free algorithm for numerical optimization of non-linear, non-convex optimization problems in continuous domain. Inspired by the limited memory BFGS method of Liu and Nocedal (1989), the LM-CMA-ES samples candidate solutions according to a covariance matrix reproduced from m direction vectors selected during the optimization process. The decomposition of the covariance matrix into Cholesky factors allows to reduce the time and memory complexity of the sampling to $O(mn)$, where n is the number of decision variables. When n is large (e.g., $n > 1000$), even relatively small values of m (e.g., $m = 20, 30$) are sufficient to efficiently solve fully non-separable problems and to reduce the overall run-time.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Evolution strategies, CMA-ES, large scale optimization, Cholesky update

1. INTRODUCTION

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is designed to learn dependencies between decision variables by adapting a covariance matrix which defines the sampling distribution of candidate solutions [6]. This algorithm constantly demonstrates good performance at various platforms for comparing continuous optimizers such as the Black-Box Optimization Benchmarking (BBOB) workshop [2, 18] and the Special Session at Congress on Evolutionary Computation [5, 16]. The CMA-ES was also extended to noisy [7], expensive [1, 17] and multi-objective optimization [12].

The principle advantage of the CMA-ES, the learning of dependencies between n decision variables, also forms its main practical limitations such as $O(n^2)$ memory storage and $O(n^2)$ computational complexity per function evaluation [21]. These limitations may preclude the use of the CMA-ES for computationally cheap but large scale optimization problems (e.g., with $n > 100$) if the internal computational cost of CMA-ES is greater than the cost of one function evaluation. On non-trivial large scale problems with $n > 10000$ not only the internal computational cost of the CMA-ES becomes significant but it is becoming simply impossible to efficiently store the covariance matrix in memory. One may argue that there are very few known continuous domain real-world problems of that huge dimensionality. This situation probably will not change much before practitioners have a set of tools that are able to efficiently search in such huge search spaces.

Several evolution strategies (ESs) have been proposed to deal with large scale optimization problems: $O(n)$ time and space complexity algorithms such as separable CMA-ES (sep-CMA-ES [21]) and linear time Natural Evolution Strategy (R1-NES [23]), L-CMA-ES [14] with $O(m^2n)$ time and $O(mn)$ space complexity, where only m dominant eigen-pairs of the covariance matrix are computed. The sep-CMA-ES learns only the scaling of variables. The R1-NES learns only the predominant eigen-direction. The L-CMA-ES learns m dominant eigen-pairs, but its $O(m^2n)$ sampling complexity practically ends up with $O(n^2)$ when $m = \sqrt{n}$ as studied in [14] for non-separable problems where multiple adaptation directions are required.

The problem of growing time and space complexity when optimizing large scale problems is not new. It was addressed in gradient-based optimization community when it became clear that for $n > 1000$ the storage of the approximate inverse Hessian matrix precludes the use of quasi-Newton methods such as Broyden–Fletcher-Goldfarb–Shanno (BFGS) method [22]. As a solution, it was proposed not to store the matrix but to reconstruct it using information from the last m iterations [19]. The final algorithm called the limited memory BFGS algorithm (L-BFGS or LM-BFGS) is still considered to be the state-of-the-art of large scale gradient-based optimization [15]. In this paper, we demonstrate that a very similar idea can be used to reconstruct the covari-

ance matrix in the CMA-ES to reduce the time and space complexity to $O(mn)$.

The paper is organized as follows. Section 2 reviews Evolution Strategies (ESs) proposed for large scale optimization. The LM-CMA-ES algorithm is described in section 3. The experimental validation of LM-CMA-ES is reported and discussed in section 4. Section 5 concludes the paper.

2. EVOLUTION STRATEGIES FOR LARGE SCALE OPTIMIZATION

Historically, first Evolution Strategies [20] were designed to perform the search without learning dependencies between variables which is a more recent development that gradually led to the CMA-ES algorithm [8, 6]. In this section, we discuss in detail the CMA-ES algorithm and its state-of-the-art derivatives for large scale optimization. For a recent comprehensible overview of Evolution Strategies, the interested reader is referred to [11].

2.1 The CMA-ES

The Covariance Matrix Adaptation Evolution Strategy [8, 9, 6] is probably the most popular and in overall the most efficient Evolution Strategy.

The $(\mu/\mu_w, \lambda)$-CMA-ES is outlined in **Algorithm** 1. At iteration t of CMA-ES, a mean $\boldsymbol{m}^t$ of the mutation distribution (can be interpreted as an estimation of the optimum) is used to generate its k-th out of λ candidate solution $\boldsymbol{x}_k \in \mathbb{R}^n$ (line 5) by adding a random Gaussian mutation defined by a (positive definite) covariance matrix $\mathbf{C}^t \in \mathbb{R}^{n\times n}$ as

$$\boldsymbol{x}_k^t = \mathcal{N}\left(\boldsymbol{m}^t, {\sigma^t}^2\mathbf{C}^t\right) = \boldsymbol{m}^t + \sigma^t\mathcal{N}\left(\mathbf{0}, \mathbf{C}^t\right), \qquad (1)$$

where σ^t is a mutation step-size. These λ solutions then should be evaluated on an objective function f (line 6). The old mean of the mutation distribution is stored in $\boldsymbol{m}^t$ and a new mean $\boldsymbol{m}^{t+1}$ is computed as a *weighted sum* of the best μ parent individuals selected among λ generated offspring individuals (line 7). The weights $\boldsymbol{w}$ are used to control the impact of selected individuals, weights are usually higher for better ranked individuals (line 1).

The procedure of the adaptation of the step-size σ^t in CMA-ES is inherited from the Cumulative Step-Size Adaptation Evolution Strategy (CSA-ES) [8] and is controlled by evolution path $\boldsymbol{p}_\sigma^{t+1}$. Successful mutation steps $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ (line 8) are tracked in the space of sampling, i.e., in the isotropic coordinate system defined by principal components of the covariance matrix $\mathbf{C}^t$. To update the evolution path $\boldsymbol{p}_\sigma^{t+1}$ a decay/relaxation factor c_σ is used to decrease the importance of previously performed steps with time. The step-size update rule increases the step-size if the length of the evolution path $\boldsymbol{p}_\sigma^{t+1}$ is longer than the expected length of the evolution path under random selection $\mathbb{E}\,\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$, and decreases otherwise (line 13). Expectation of $\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$ is approximated by $\sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$. A damping parameter d_σ controls the change of the step-size.

The covariance matrix update consists of two parts (line 12): *rank-one update* [9] and *rank-μ update* [6]. The rank-one update computes evolution path $\boldsymbol{p}_c^{t+1}$ of successful moves of the mean $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ of the mutation distribution in the given coordinate system (line 10), in a similar way as the evolution path $\boldsymbol{p}_\sigma^{t+1}$ of the step-size. To stall the update of $\boldsymbol{p}_c^{t+1}$ when σ increases rapidly, a h_σ trigger is used (line 9).

Algorithm 1 The $(\mu/\mu_w, \lambda)$-CMA-ES

1: **given** $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3\ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $\boldsymbol{w}_i = \frac{\ln(\mu+\frac{1}{2})-\ln i}{\sum_{j=1}^{\mu}(\ln(\mu+\frac{1}{2})-\ln j)}$ for $i = 1\ldots\mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w-1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$, $c_1 = \frac{2\min(1,\lambda/6)}{(n+1.3)^2+\mu_w}$, $c_\mu = \frac{2\,(\mu_w-2+1/\mu_w)}{(n+2)^2+\mu_w}$
2: **initialize** $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=0} = \mathbf{0}, \boldsymbol{p}_c^{t=0} = \mathbf{0}, \mathbf{C}^{t=0} = \mathbf{I}, t \leftarrow 0$
3: **repeat**
4: **for** $k = 1, \ldots, \lambda$ **do**
5: $\boldsymbol{x}_k = \boldsymbol{m}^t + \sigma^t\mathcal{N}\left(\mathbf{0}, \mathbf{C}^t\right)$
6: $\boldsymbol{f}_k = f(\boldsymbol{x}_k)$
7: $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$ // the symbol $i:\lambda$ denotes i-th best individual on f
8: $\boldsymbol{p}_\sigma^{t+1} \leftarrow (1-c_\sigma)\boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2-c_\sigma)}\sqrt{\mu_w}\mathbf{C}^{t-\frac{1}{2}}\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$
9: $h_\sigma = \mathbb{1}_{\|p_\sigma^{t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4+\frac{2}{n+1})\,\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|}$
10: $\boldsymbol{p}_c^{t+1} \leftarrow (1-c_c)\boldsymbol{p}_c^t + h_\sigma\sqrt{c_c(2-c_c)}\sqrt{\mu_w}\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$
11: $\mathbf{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda}-\boldsymbol{m}^t}{\sigma^t} \times \frac{(\boldsymbol{x}_{i:\lambda}-\boldsymbol{m}^t)^T}{\sigma^t}$
12: $\mathbf{C}^{t+1} = (1-c_1-c_\mu)\mathbf{C}^t + c_1 \underbrace{\boldsymbol{p}_c^{t+1}\boldsymbol{p}_c^{t+1T}}_{\text{rank-one update}} + c_\mu \underbrace{\mathbf{C}_\mu^+}_{\text{rank-}\mu\text{ update}}$
13: $\sigma^{t+1} \leftarrow \sigma^t \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1))$
14: $t \leftarrow t + 1$
15: **until** *stopping criterion is met*

The rank-μ update computes a covariance matrix $\mathbf{C}_\mu$ as a weighted sum of covariances of successful steps of μ best individuals (line 11). The update of $\mathbf{C}$ itself is a replace of previously accumulated information by a new one with corresponding weights of importance (line 12): c_1 for covariance matrix $\boldsymbol{p}_c^{t+1}\boldsymbol{p}_c^{t+1T}$ of rank-one update and c_μ for $\mathbf{C}_\mu$ of rank-μ update [6] such that $c_1 + c_\mu \leq 1$. Recently it was proposed to also take into account unsuccessful mutations in the *"active" rank-μ update* [10, 13].

In CMA-ES, the factorization of the covariance $\mathbf{C}$ into $\mathbf{A}\mathbf{A}^T = \mathbf{C}$ is needed to sample the multivariate normal distribution (line 5). The eigendecomposition with $O(n^3)$ complexity is used for the factorization. Already in the original CMA-ES it was proposed to perform the eigendecomposition every $n/10$ generations (not shown in **Algorithm** 1) to reduce the complexity per function evaluation to $O(n^2)$

2.2 Large Scale Variants

The original CMA-ES has $O(n^2)$ time and space complexity that precludes its applications for large scale optimization with $n \gg 100$. To enable the algorithm for large scale optimization, a linear time and space version called sep-CMA-ES was proposed in [21]. The algorithm does not learn dependencies but the scaling of variables by restraining the covariance matrix update to the diagonal elements:

$$c_{jj}^{t+1} = (1-c_{cov})c_{jj}^t + \frac{1}{\mu_{cov}}\left(\boldsymbol{p}_c^{t+1}\right)_j^2 + c_{ccov}\left(1-\frac{1}{\mu_{ccov}}\right)\sum_{i=1}^{\mu} w_i c_{jj}^t \left(z_{i:\lambda}{}^{t+1}\right)_j^2, j = 1, \ldots, n \qquad (2)$$

where, for $j = 1, \ldots, n$ the c_{jj} are the diagonal elements of $\mathbf{C}^t$ and the $\left(z_{i:\lambda}{}^{t+1}\right)_j = \left(x_{i:\lambda}{}^{t+1}\right)_j / (\sigma^t\sqrt{(c_{jj})}$.

This update reduces the computational complexity to $O(n)$ and allows to exploit problem separability, thus the original property of being rotationally invariant is lost. The algorithm demonstrated good performance on separable problems and even outperformed CMA-ES on non-separable Rosenbrock function for $n > 100$.

A novel Natural Evolution Strategy (NES) variant, the Rank-One NES (R1-NES), which uses a low rank approximation of the search distribution covariance matrix was proposed recently by [23]. The algorithm adapts the search distribution according to the natural gradient with a particular parametrization of the covariance matrix,

$$\mathbf{C} = \sigma^2(\mathbf{I} + \boldsymbol{u}\boldsymbol{u}^T), \tag{3}$$

where u and σ are the parameters to be adjusted. The adaptation of the predominant eigen-direction $\boldsymbol{u}$ allows the algorithm to solve highly non-separable problems while maintaining only $O(n)$ time and space complexity.

A version of CMA-ES with a limited memory storage also called limited memory CMA-ES (L-CMA-ES) was proposed by [14]. The L-CMA-ES uses the m eigen-vectors and eigen-values spanning the m-dimensional dominant subspace of the n-dimensional covariance matrix $\mathbf{C}$. The authors adapted a singular value decomposition updating algorithm developed in [3] that allowed to avoid the explicit computation and storage of the covariance matrix. For $m < n$ the performance in terms of number of function evaluations gradually decreases while enabling the search in $\mathbb{R}^n$ for $n > 10000$. However, the computational complexity of $O(m^2 n)$ practically (for m in order of $\sqrt{n}$ [14]) leads to the same limitations as for the original CMA-ES.

The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES proposed in [24] is of special interest in this paper because the LM-CMA-ES is based on this algorithm. The Cholesky-CMA represents a version of CMA-ES with rank-one update where instead of performing the factorization of the covariance matrix $\mathbf{C}^t$ into $\mathbf{A}^t {\mathbf{A}^t}^T = \mathbf{C}^t$, the Cholesky factor $\mathbf{A}^t$ and its inverse ${\mathbf{A}^t}^{-1}$ are iteratively updated. From **Theorem 1** [24] it follows that if $\mathbf{C}^t$ is updated as

$$\mathbf{C}^{t+1} = \alpha\mathbf{C}^t + \beta\boldsymbol{v}^t{\boldsymbol{v}^t}^T, \tag{4}$$

where $\boldsymbol{v} \in \mathbb{R}^n$ is given in the decomposition form $\boldsymbol{v}^t = \mathbf{A}^t\boldsymbol{z}^t$, and $\alpha, \beta \in \mathbb{R}^+$, then for $\boldsymbol{z} \neq \boldsymbol{0}$ a Cholesky factor of the matrix $\mathbf{C}^{t+1}$ can be computed by

$$\mathbf{A}^{t+1} = \sqrt{\alpha}\mathbf{A}^t + \frac{\sqrt{\alpha}}{\|\boldsymbol{z}^t\|^2}\left(\sqrt{1 + \frac{\beta}{\alpha}\|\boldsymbol{z}^t\|^2} - 1\right)[\mathbf{A}^t\boldsymbol{z}^t]{\boldsymbol{z}^t}^T, \tag{5}$$

for $\boldsymbol{z}_t = \boldsymbol{0}$ we have $\mathbf{A}^{t+1} = \sqrt{\alpha}\mathbf{A}^t$. From the **Theorem 2** [24] it follows that if ${\mathbf{A}^{-1}}^t$ is the inverse of $\mathbf{A}^t$, then the inverse of $\mathbf{A}^{t+1}$ can be computed by

$${\mathbf{A}^{-1}}^{t+1} = \frac{1}{\sqrt{\alpha}}{\mathbf{A}^{-1}}^t - \frac{1}{\sqrt{\alpha}\|\boldsymbol{z}^t\|^2}\left(1 - \frac{1}{\sqrt{1 + \frac{\beta}{\alpha}\|\boldsymbol{z}^t\|^2}}\right)\boldsymbol{z}^t[{\boldsymbol{z}^t}^T{\mathbf{A}^{-1}}^t] \tag{6}$$

for $\boldsymbol{z}^t \neq \boldsymbol{0}$ and by ${\mathbf{A}^{-1}}^{t+1} = \frac{1}{\sqrt{\alpha}}{\mathbf{A}^{-1}}^t$ for $\boldsymbol{z}^t = \boldsymbol{0}$.

The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES is outlined in **Algorithm 2**. As well as in the original CMA-ES, Cholesky-CMA-ES proceeds by sampling λ candidate solutions (lines 4 - 7) and taking into account the most successful μ out of λ solutions in the evolution paths adaptation (lines 10 and 11). However, the eigen-decomposition procedure is not required anymore because the Cholesky factor and its inverse are updated incrementally (line 13 and 14). This simplifies a lot the implementation of the algorithm and reduces its time complexity to $O(n^2)$. A postponed update of the Cholesky factors every $O(n)$ iterations would not reduce the asymptotic complexity further (as it does in the original CMA-ES) because the quadratic complexity will remain due to matrix-vector multiplications needed to sample new individuals.

Algorithm 2 The $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES

1: **given** $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3\ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+1)-\ln(i)}{\mu\ln(\mu+1)-\sum_{j=1}^{\mu}\ln(j)}; i = 1\ldots\mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\sqrt{\mu_w}}{\sqrt{n}+\sqrt{\mu_w}}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w - 1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$, $c_1 = \frac{2}{(n+\sqrt{2})^2}$
2: **initialize** $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=0} = \boldsymbol{0}, \boldsymbol{p}_c^{t=0} = \boldsymbol{0}, \mathbf{A}^{t=0} = \mathbf{I}, \mathbf{A}_{inv}^{t=0} = \mathbf{I}, t \leftarrow 0$
3: **repeat**
4: **for** $k = 1, \ldots, \lambda$ **do**
5: $\boldsymbol{z}_k = \mathcal{N}(\boldsymbol{0}, \mathbf{I})$
6: $\boldsymbol{x}_k = \boldsymbol{m}^t + \sigma^t\mathbf{A}\boldsymbol{z}_k$
7: $\boldsymbol{f}_k = f(\boldsymbol{x}_k)$
8: $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i\boldsymbol{x}_{i:\lambda}$
9: $\boldsymbol{z}_w \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i\boldsymbol{z}_{i:\lambda}$
10: $\boldsymbol{p}_\sigma^{t+1} \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w}\boldsymbol{z}_w$
11: $\boldsymbol{p}_c^{t+1} \leftarrow (1 - c_c)\boldsymbol{p}_c^t + \sqrt{c_c(2 - c_c)}\sqrt{\mu_w}\mathbf{A}\boldsymbol{z}_w$
12: $\boldsymbol{v} \leftarrow \mathbf{A}_{inv}^t\boldsymbol{p}_c$
13: $\mathbf{A}^{t+1} = \sqrt{1 - c_1}\mathbf{A}^t + \frac{\sqrt{1-c_1}}{\|\boldsymbol{v}^t\|^2}\left(\sqrt{1 + \frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2} - 1\right)\boldsymbol{p}_c{\boldsymbol{v}^t}^T$
14: $\mathbf{A}_{inv}^{t+1} = \frac{1}{\sqrt{1-c_1}}\mathbf{A}_{inv}^t - \frac{1}{\sqrt{1-c_1}\|\boldsymbol{v}^t\|^2}\left(1 - \frac{1}{\sqrt{1 + \frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2}}\right)\boldsymbol{v}^t[{\boldsymbol{v}^t}^T\mathbf{A}_{inv}^t]$,
15: $\sigma^{t+1} \leftarrow \sigma^t\exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(\boldsymbol{0},\mathbf{I})\|} - 1))$
16: $t \leftarrow t + 1$
17: **until** *stopping criterion is met*

The non-elitist Cholesky-CMA is a good alternative to the original CMA-ES and demonstrates a comparable performance [24]. While it has the same computational and memory complexity, the lack of rank-μ update may deteriorate its performance on problems where it is essential.

3. THE LM-CMA-ES

In this section, we first present main components of the computationally cheap limited memory CMA-ES and then introduce the algorithm itself. The components are: a procedure for reconstruction of Cholesky factors of a covariance matrix using stored direction vectors, a procedure to store these vectors and a new procedure for step-size adaptation.

3.1 Reconstruction of Cholesky factors

The idea to reconstruct the inverse Hessian matrix in the BFGS method [19] enabled its application for large scale gradient-based optimization. While the CMA-ES is a gradient-free algorithm, the two algorithms are indeed similar with a difference that the latter estimates the gradient in a stochas-

Algorithm 3 Az(): Cholesky factor - vector update

1: **given** $\boldsymbol{z} \in \mathbb{R}^n, m \in \mathbb{Z}_+, \boldsymbol{j} \in \mathbb{Z}_+^m, \boldsymbol{P} \in \mathbb{R}^{m\times n}, \boldsymbol{V} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m, a \in [0,1]$
2: **initialize** $\boldsymbol{x} \leftarrow \boldsymbol{z}$
3: **for** $t = 1, \ldots, min(m, |\boldsymbol{j}|)$ **do**
4: $\quad k \leftarrow \boldsymbol{b}_{\boldsymbol{j}_t} \boldsymbol{V}_{(\boldsymbol{j}_t,:)} \cdot \boldsymbol{x}$
5: $\quad \boldsymbol{x} \leftarrow a\boldsymbol{x} + k\boldsymbol{P}_{(\boldsymbol{j}_t,:)}$
6: **return** $\boldsymbol{x}$

Algorithm 4 Ainvz(): inverse Cholesky factor - vector update

1: **given** $\boldsymbol{z} \in \mathbb{R}^n, m \in \mathbb{Z}_+, \boldsymbol{j} \in \mathbb{Z}^m, \boldsymbol{V} \in \mathbb{R}^{m\times n}, \boldsymbol{d} \in \mathbb{R}^m, c \in [0,1]$
2: **initialize** $\boldsymbol{x} \leftarrow \boldsymbol{z}$
3: **for** $t = 1, \ldots, min(m, |\boldsymbol{j}|)$ **do**
4: $\quad k \leftarrow \boldsymbol{d}_{\boldsymbol{j}_t} \boldsymbol{V}_{(\boldsymbol{j}_t,:)} \cdot \boldsymbol{x}$
5: $\quad \boldsymbol{x} \leftarrow c\boldsymbol{x} - k\boldsymbol{V}_{(\boldsymbol{j}_t,:)}$
6: **return** $\boldsymbol{x}$

Algorithm 5 UpdateSet(): direction vectors selection

1: **given** $m \in \mathbb{R}^+, \boldsymbol{j} \in \mathbb{Z}_+^m, \boldsymbol{l} \in \mathbb{Z}_+^m, t \in \mathbb{Z}_+, N_{steps} \in \mathbb{Z}_+$
2: **if** $t < m$ **then**
3: $\quad \boldsymbol{j}_t \leftarrow t$
4: **else**
5: $\quad i_{min} \leftarrow 1 + argmin_i \left(\boldsymbol{l}_{\boldsymbol{j}_{i+1}} - \boldsymbol{l}_{\boldsymbol{j}_i}\right), |1 \le i \le (m-1)$
6: $\quad$ **if** $\boldsymbol{l}_{\boldsymbol{j}_{i_{min}}} - \boldsymbol{l}_{\boldsymbol{j}_{i_{min}-1}} \ge N_{steps}$ **then**
7: $\quad\quad i_{min} \leftarrow 1$
8: $\quad$ **if** $i_{min} \neq m$ **then**
9: $\quad\quad \boldsymbol{j}_{tmp} \leftarrow \boldsymbol{j}_{i_{min}}$
10: $\quad\quad$ **for** $i = i_{min}, \ldots, m-1$ **do**
11: $\quad\quad\quad \boldsymbol{j}_i \leftarrow \boldsymbol{j}_{i+1}$
12: $\quad\quad \boldsymbol{j}_m \leftarrow \boldsymbol{j}_{tmp}$
13: $j_{cur} \leftarrow \boldsymbol{j}_{min(t+1,m)}$
14: $\boldsymbol{l}_{j_{cur}} \leftarrow t$
15: **return**: $\boldsymbol{j}_{cur}$, $\boldsymbol{j}$, $\boldsymbol{l}$

tic way. This observation inspired us to investigate whether a similar matrix reconstruction procedure can be used in CMA-ES as well to reduce its time and space complexity.

As can be seen, the only use of Cholesky factor $\mathbf{A}^t$ in **Algorithm** 2 is for sampling of new solutions after $\mathbf{A}^t\boldsymbol{z}_k$ or for its own update to $\mathbf{A}^{t+1}$. By setting $a = \sqrt{1-c_1}$ and $b^t = \frac{\sqrt{1-c_1}}{\|\boldsymbol{v}^t\|^2}\left(\sqrt{1+\frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2} - 1\right)$, one can rewrite the line (13) as

$$\mathbf{A}^{t+1} = a\mathbf{A}^t + b^t\boldsymbol{p}_c^t\boldsymbol{v}^{t^T}, \tag{7}$$

In the following, we show how the vectors needed to sample new candidate solutions can be obtained without an explicit storage of Cholesky factors. At iteration $t = 0$, $\mathbf{A}^0 = \mathbf{I}$ and $\mathbf{A}^0\boldsymbol{z} = \boldsymbol{z}$ in line (6) of **Algorithm** 2, the new updated Cholesky factor $\mathbf{A}^1 = a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T}$. At iteration $t = 1$, $\mathbf{A}^1\boldsymbol{z} = (a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T})\boldsymbol{z} = a\boldsymbol{z} + b^0\boldsymbol{p}_c^0(\boldsymbol{v}^{0^T}\boldsymbol{z})$ and $\mathbf{A}^2 = a(a\mathbf{I} + b^0\boldsymbol{p}_c^0\boldsymbol{v}^{0^T}) + b^1\boldsymbol{p}_c^1\boldsymbol{v}^{1^T}$. Thus, a very simple iterative procedure which scales as $O(mn)$ can be used to sample candidate solutions in $\mathbb{R}^n$ according to the Cholesky factor $\mathbf{A}^t$ reconstructed from m pairs of vectors $\boldsymbol{p}_c^t$ and $\boldsymbol{v}^t$.

The pseudo-code of the procedure to reconstruct $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{z}$ from m direction vectors[1] is given in **Algorithm** 3. At each iteration of reconstruction of $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{z}$ (lines 3 - 4), $\boldsymbol{x}$ is updated as a sum of a-weighted version of itself and b^t-weighted evolution path $\boldsymbol{p}_c^t$ scaled by the dot product of $\boldsymbol{v}^t$ and $\boldsymbol{x}$. As can be seen, the **Algorithm** uses $\boldsymbol{j}(t)$ indexation instead of t. This is simply a convenient way to have references to matrices $\boldsymbol{P}$ and $\boldsymbol{V}$ which store $\boldsymbol{p}_c^t$ and $\boldsymbol{v}^t$ vectors, respectively. In the next subsection, we will show how to efficiently manipulate these vectors.

A very similar approach can be used to reconstruct $\boldsymbol{x} = \mathbf{A}^{t^{-1}}\boldsymbol{z}$, for the sake of reproducibility the pseudo-code is given in **Algorithm** 4 for $c = 1/\sqrt{1-c_1}$ and $d^t = \frac{1}{\sqrt{1-c_1}\|\boldsymbol{v}^t\|^2}\times$ $\times\left(1 - \frac{1}{\sqrt{1+\frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2}}\right)$. The computational complexity of both procedures scales as $O(mn)$.

3.2 Direction Vectors Selection and Storage

It is an open question how to use only $m \ll n$ direction vectors to obtain a comparable amount of useful information as stored in the covariance matrix of the original CMA-ES. For large n and $\lambda \ll n$, evolution path vectors $\boldsymbol{p}_c^t$ from the last m iterations are likely to be quite similar and therefore to contain only some local information.

In this paper, we propose a simple approach which forces m selected vectors to be at approximately the same distance from each other in terms of number of iterations, but at most with the distance of N_{steps} from each other given that the m-th vector is the one from the last iteration. This selection procedure is outlined in **Algorithm** 5 which outputs an array of pointers $\boldsymbol{j}$ such that $\boldsymbol{j}_1$ points out to a row in matrices $\boldsymbol{P}$ and $\boldsymbol{V}$ with the oldest saved vectors $\boldsymbol{p}_c$ and $\boldsymbol{v}$ which will be taken into account during the reconstruction procedure. The higher the index i of $\boldsymbol{j}_i$ the more recent the corresponding direction vector is. The index j_{cur} points out to the oldest vector which will be replaced by the newest one in the same iteration when the procedure is called. The rule to choose a vector to be replaced is the following: find a pair of consecutively saved vectors with the closest distance (in terms of number of iterations, stored in $\boldsymbol{l}$) between each other (line 5), if this distance is smaller than N_{steps} then the most recent vector will be removed by assigning $j_{cur} \leftarrow i_{min}$, otherwise the oldest vector among m saved vectors should be removed. Thus, the procedure gradually replaces vectors in a way to keep them at approximately the same distance, but at most at distance of N_{steps} iterations.

3.3 Population Success Rule

An elegant success rule for step-size adaptation called the *median success rule* was recently proposed in [4]. It is applicable to non-elitist multi-recombinant evolution strategies. The median success rule compares the median fitness of the population to a fitness from the previous iteration. The comparison fitness is chosen to achieve a target success rate of 1/2. The empirical validation demonstrated that the median success rule is competitive to CSA [4].

In practice, one should count the number K_{succ} of individuals in the current population better than some j-th best individual of the previous population, where j depends on

[1] more precisely, we mean m evolution paths $\boldsymbol{p}_c$ and their inverses $\boldsymbol{v}$ but for brevity we say m direction vectors

n and λ but can be set to be 0.3λ [4]. Then, a normalized measurement

$$z \leftarrow \frac{2}{\lambda}\left(K_{succ} - \frac{\lambda+1}{2}\right) \tag{8}$$

can be computed such that $z \geq 0$ iff the median individual was successful.

The step-size is adapted as

$$\sigma \leftarrow \sigma \exp\left(\frac{s}{d_\sigma}\right), \tag{9}$$

where $s \leftarrow (1-c_\sigma)s + c_\sigma z$ and $d_s = 2(n-1)/n$.

We suppose that while being quite elegant the median success rule has a potential drawback that we will demonstrate on an example. Let us suppose that fitness values (to be minimized) of the previous population are say $\boldsymbol{f}_{t-1} = [2.1, 3.1, 4.1, 5.1, 6.1, 7.1, 8.1]$ while the fitness values of the current population are $\boldsymbol{f}_t = [1, 2, 3, 4, 5, 6, 7]$. According to the median success rule if j is chosen as, e.g., 3, the number of successful individual (with fitness values better than or equal to $\boldsymbol{f}_{t-1}(3) = 4.1$) is 4 (as $\boldsymbol{f}_t(1) = 1, \boldsymbol{f}_t(2) = 2, \boldsymbol{f}_t(3) = 3$ and $\boldsymbol{f}_t(4) = 4$). The computed value of K_{succ} is then will be used to adapt the step-size. However, its computation does not take into account the values of $\boldsymbol{f}_{t-1}(i)$ for $1 \leq i < j$ and even if all such $\boldsymbol{f}_{t-1}(i)$ are better than the best solution $\boldsymbol{f}_t(1)$, this information will not be taken into account.

This potential drawback is not the drawback in a sense that the *median* success rule was designed in this way. However, we suppose that the information omitted in the median success rule can be useful since it can provide a better estimate whether and by how much the new population is more successful than the previous one.

In this paper, we introduce *the population success rule* (PSR) for step-size adaptation for non-elitist multi-recombinant evolution strategies. To estimate the success of the current population we combine fitness function values from the previous and current population into a mixed set

$$\boldsymbol{f}_{mix} \leftarrow \boldsymbol{f}_{t-1} \cup \boldsymbol{f}_t \tag{10}$$

Then, we rank all individual in the mixed set to define two sets $\boldsymbol{r}_{t-1}$ and $\boldsymbol{r}_t$ containing ranks of individuals of the previous and current populations ranked in the mixed set.

We compute a normalized success measurement

$$z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} \boldsymbol{r}_t(i) - \boldsymbol{r}_{t-1}(i)}{\lambda^2} - z^*, \tag{11}$$

where z^* is a target success ratio. The step-size can be adapted as in (9).

The proposed *population success rule* takes into account all fitness function values from the previous and current generation. This success rule seems to represent a more general case of the 1/5th-rule which can be obtained when $\lambda = 1$.

3.4 The Algorithm

In the previous subsection we introduced all necessary components of the $(\mu/\mu_w, \lambda)$-LM-CMA-ES outlined in **Algorithm** 6. The algorithm represents a computationally efficient limited memory version of CMA-ES, where the Cholesky factor and its inverse are reconstructed from a set of stored direction vectors (lines 6 and 9). The mutation step-size is adapted using the population success rule (lines 15 - 13). The algorithm memory and time complexity scales as $O(mn)$.

Algorithm 6 The $(\mu/\mu_w, \lambda)$-LM-CMA-ES

1: **given** $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3\ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+1)-\ln(i)}{\mu\ln(\mu+1)-\sum_{j=1}^{\mu}\ln(j)}; i = 1 \ldots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = 0.3$, $d_\sigma = 1$, $m = 4 + \lfloor 3\ln n \rfloor$, $N_{steps} = m$, $c_c = \frac{1}{m}$, $c_1 = \frac{1}{10ln(n+1)}$
2: **initialize** $\boldsymbol{m}^{t=0} \in \mathbb{R}^n, \sigma^{t=0} > 0, \boldsymbol{p}_c^{t=0} = \boldsymbol{0}, s \leftarrow 0, t \leftarrow 0$
3: **repeat**
4: **for** $k = 1, \ldots, \lambda$ **do**
5: $\boldsymbol{z}_k = \mathcal{N}(\boldsymbol{0}, \mathbf{I})$
6: $\boldsymbol{x}_k = \boldsymbol{m}^t + \sigma^t A z(\boldsymbol{z}_k)$
7: $\boldsymbol{f}_k^t = f(\boldsymbol{x}_k)$
8: $\boldsymbol{m}^{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$
9: $\boldsymbol{p}_c^{t+1} \leftarrow (1-c_c)\boldsymbol{p}_c^t + \sqrt{c_c(2-c_c)}\sqrt{\mu_w}(\boldsymbol{m}^{t+1} - \boldsymbol{m}^t)/\sigma$
10: $\boldsymbol{v} \leftarrow Ainvz(\boldsymbol{p}_c^{t+1})$
11: $j_{cur} \leftarrow UpdateSet()$
12: $\boldsymbol{V}_{(j_{cur},:)} \leftarrow \boldsymbol{v}; \boldsymbol{P}_{(j_{cur},:)} \leftarrow \boldsymbol{p}_c^{t+1}$
13: $\boldsymbol{b}_{j_{cur}} \leftarrow \frac{\sqrt{1-c_1}}{\|\boldsymbol{v}^t\|^2}\left(\sqrt{1 + \frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2} - 1\right)$
14: $\boldsymbol{d}_{j_{cur}} = \frac{1}{\sqrt{1-c_1}\|\boldsymbol{v}^t\|^2}\left(1 - \frac{1}{\sqrt{1+\frac{c_1}{1-c_1}\|\boldsymbol{v}^t\|^2}}\right)$,
15: $\boldsymbol{r}^t, \boldsymbol{r}^{t-1} \leftarrow$ Ranks of $\boldsymbol{f}^t$ and $\boldsymbol{f}^{t-1}$ in $\boldsymbol{f}^t \cup \boldsymbol{f}^{t-1}$
16: $z_{PSR} \leftarrow \frac{\sum_{i=1}^{\lambda} \boldsymbol{r}^t(i) - \boldsymbol{r}^{t-1}(i)}{\lambda^2} - z^*$
17: $s \leftarrow (1-c_\sigma)s + c_\sigma z_{PSR}$
18: $\sigma^{t+1} \leftarrow \sigma^t \exp(s/d_\sigma)$
19: $t \leftarrow t+1$
20: **until** *stopping criterion is met*

4. SIMULATION RESULTS

In this section, we perform a set of numerical experiments to assess the performance of the proposed LM-CMA-ES on large scale optimization problem with $n = 128, 256, \ldots, 8096$. We investigate the performance on three basic problems: Sphere function $f_{Sphere}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i^2$, separable Ellipsoid function $f_{Elli}(\boldsymbol{x}) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} \boldsymbol{x}_i^2$ and its rotated version $f_{ElliRot}(\boldsymbol{x}) = f_{Elli}(\boldsymbol{Q}\boldsymbol{x})$, where $\boldsymbol{Q}$ is an orthogonal $n \times n$ matrix with each column vector $\boldsymbol{q}_i$ being a uniformly distributed unit vector implementing an angle-preserving transformation [21].

4.1 Experimental Setting

For the sake of reproducibility, the MATLAB/C++ source code of all tested algorithms is available at `https://sites.google.com/site/lmcmaeses/`.

In the order to estimate the performance of $(\mu/\mu_w, \lambda)$-LM-CMA-ES, we compare it with $(\mu/\mu_w, \lambda)$-Cholesky-CMA-ES and $(\mu/\mu_w, \lambda)$-Sep-CMA-ES. We use the default parameters for Cholesky-CMA-ES and Sep-CMA-ES as given in [24] and [21], respectively. The parameters of LM-CMA-ES are given in **Algorithm** 6. For all problems, the mean $\boldsymbol{m}^{t=0}$ is initialized in the range $[-5, 5]^n$, the population is sampled with initial step-size $\sigma^{t=0} = 5$ and using the same seed per run. Note that in all cases we use the default population size $\lambda = 4 + \lfloor 3\ln n \rfloor$.

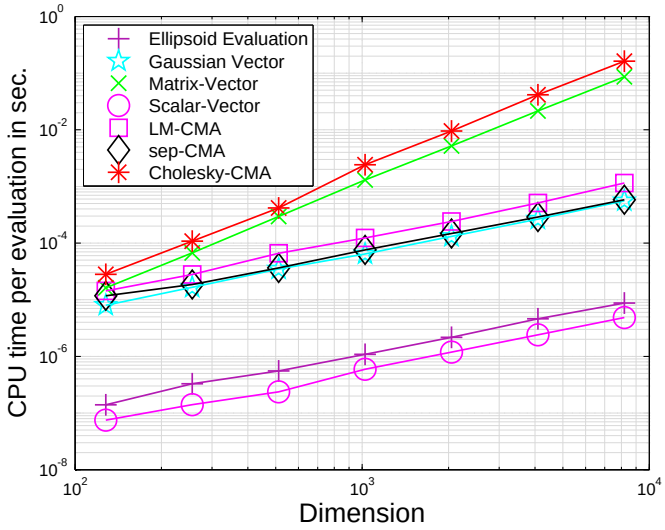

Figure 1: Timing results of LM-CMA-ES on the separable Ellipsoid compared to sep-CMA-ES and Cholesky-CMA-ES. The results were computed using at most 10^5 function evaluations for sep-CMA-ES and LM-CMA-ES and using at most 10^4 for Cholesky-CMA-ES.

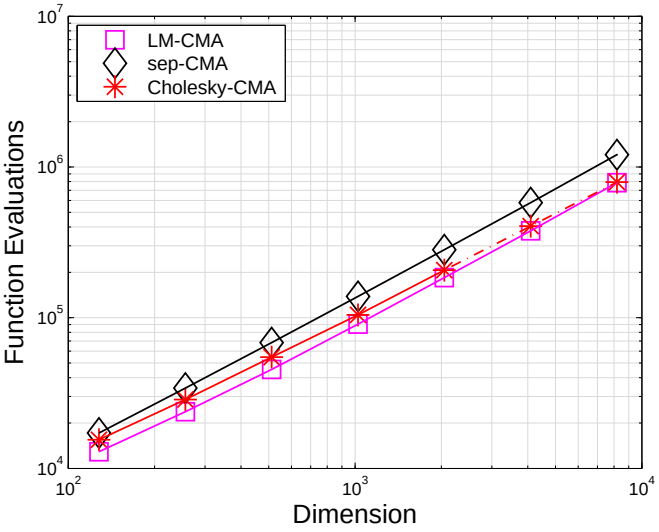

Figure 2: Results of LM-CMA-ES on the Sphere function compared to sep-CMA-ES and Cholesky-CMA-ES. Lines show the median of 11 runs for different problem dimensions to reach the target fitness value of 10^{-10}. The dotted line is an extrapolation.

4.2 Memory and Computational Complexity of LM-CMA-ES

The LM-CMA-ES has $O(mn)$ memory complexity and more specifically stores $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$, $\boldsymbol{V} \in \mathbb{R}^{m\times n}$ and λ solution vectors $\boldsymbol{x}_i$. For large n and $m = \lambda_{default} = \lambda = 4 + \lfloor 3\ln n \rfloor$ used in this paper, the algorithm stores approximately $3mn$ real-valued parameters. If a real-valued parameter requires 8 bytes of memory, then for $n = 8192$ the LM-CMA-ES will require 5.8 megabytes while the original CMA-ES would start to reach its limit by requiring 1 gigabyte of memory. Using the same amount of memory (more specifically, 1.03 gigabyte), the LM-CMA-ES will able to optimize a 1 million dimensional problem. Indeed, by taking $m = 1$ even less memory would be needed but the latter possibility makes sense only if the performance stays at a reasonable level.

Figure 1 shows how fast CPU time per evaluation scales for different operations (measured on a 2.0 GHz processor). Scalar-vector multiplication of a vector with n variables scales linearly with ca. $6 \cdot 10^{-10} n$ seconds, evaluation of the separable Ellipsoid is twice more expensive if a temporary data is used. Sampling of n normally distributed variables scales as ca. 100 vectors-scalar multiplications. As can be seen in Figure 1, sampling of $\boldsymbol{z}_k$ dominates the computational overhead of sep-CMA already after $n = 128$. The LM-CMA-ES scales almost linearly for $n \geq 1024$ as ca. $1.3 \cdot 10^{-7} n$ or ca. 200 scalar-vector multiplications. Matrix-vector multiplication scale quadratically with n and Cholesky-CMA-ES scales as ca. 1.5-2 matrix-vector multiplications.

Practically, the LM-CMA-ES is about 40 times faster (in terms of its internal computation cost per function evaluation) for $n = 2048$ and about 140 times faster for $n = 8192$ than Cholesky-CMA-ES. The LM-CMA-ES is only about 2 times slower than sep-CMA-ES, whose cost is dominated by sampling from normal distribution.

The computation cost of CMA-ES with full covariance matrix learning limits its applicability for $n \gg 100$ and makes it intractable because of memory for $n > 10000$.

4.3 Performance on Sphere and Ellipsoid

The Sphere function is often viewed in Evolutionary Computation to be the first function to look at when benchmarking evolutionary algorithms. Figure 2 demonstrates a comparable performance of LM-CMA-ES with population success rule, sep-CMA-ES with CSA and Cholesky-CMA-ES with CSA. It should be further studied what is the effect of the target population success rate (set to $z^* = 0.25$) whose value was chosen the same for all experiments in order to obtain a reasonable performance on Ellipsoid functions.

Figure 3-Left shows that both LM-CMA-ES and Cholesky-CMA-ES are rotationally invariant and therefore they optimization runs (one per function) are almost coincide (within the algorithm). The sep-CMA-ES is not rotationally invariant and therefore it performs better on the separable Ellipsoid than on its rotated version where the exploitation of the separability is not that useful. Importantly, the LM-CMA-ES often outperforms the Cholesky-CMA-ES in the beginning of optimization, while the adaptation of the full covariance matrix makes Cholesky-CMA-ES faster at later stages. Figure 3-Right shows that the loss of performance of LM-CMA-ES compared to Cholesky-CMA-ES is in order of a factor of 3-4 given that for $n = 2048$ the LM-CMA-ES uses only $m = 26$ direction vectors. It is important to keep in mind that for $n > 10000$ the Cholesky-CMA-ES becomes intractable both due to its memory and computational complexity. Then, the sep-CMA-ES becomes an alternative, however, it does not learn dependencies and might be therefore inefficient (see Figure 3-Left).

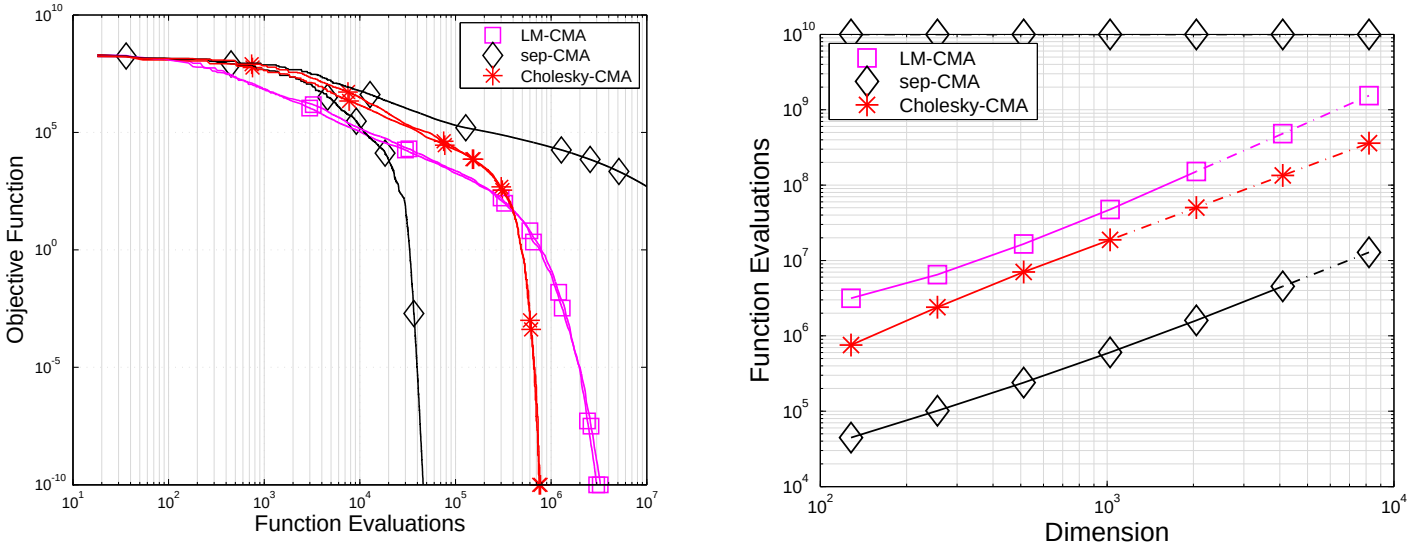

Figure 3: Left: Convergence plots of LM-CMA-ES, sep-CMA-ES and Cholesky-CMA-ES on 128-dimensional axis-parallel and rotated Ellipsoid functions. Right: The median of 11 runs on separable Ellipsoid function for different problem dimensions. The dotted lines correspond to extrapolated results by preserving the same scaling as between the last two actual estimations.

We discussed several large scale ESs in this paper: L-CMA-ES [14] and R1-NES [23]. We compared the LM-CMA-ES indirectly by analyzing the results from [14] and [23]. It takes about 6000 seconds for L-CMA-ES to solve 200-dimensional Ellipsoid after about $7e+6$ function evaluations with $m = \sqrt{n} = 14$ and 4000 seconds after $4e+6$ evaluations with $m = n/2 = 100$. The LM-CMA-ES solves the same problem after about 125 seconds and $5.3e+6$ function evaluations with $m = 19$. The performance is comparable while the LM-CMA-ES is about $32 - 48$ times faster that is unlikely to be only due to a different processor or implementation used. The L-CMA-ES has $O(m^2 n)$ computational complexity and therefore it is in order of m times computationally slower than LM-CMA-ES.

The R1-NES algorithm performs well on non-separable problems but tends to fail on problems where the learning of multiple principal components is essential, e.g., it fails on moderate dimensional rotated Ellipsoid function [23]. On Rosenbrock function the LM-CMA-ES is about 5 times faster (not shown) in terms of number of function evaluations for $n = 256, 512$. The R1-NES also samples from the normal distribution, and therefore the lower bound of its computational complexity is predefined (see Figure 1).

We performed an experiment on 100,000-dimensional separable Ellipsoid problems for 100,000 function evaluations (i.e., n evaluations). The original CMA-ES and Cholesky-CMA-ES cannot be applied due to memory requirements. The applicability of L-CMA-ES is also limited due to its $O(m^2 n)$ computational complexity. The results for sep-CMA-ES specifically designed for large scale optimization and the proposed LM-CMA-ES are shown in Figure 4. While the LM-CMA-ES gradually improves the fitness similarly as in Figure 3-Left, the sep-CMA-ES does not improve it because it diverges from the very first iterations. To investigate whether it is a mistake in our implementation, we launched the same experiment using the sep-CMA-ES author's MATLAB implementation where the divergence was also observed.

It should be noted that the separable Ellipsoid can be easily solved by various Evolutionary Algorithms which implicitly or explicitly exploit its separability, our purpose of its usage is to investigate how the LM-CMA-ES performs on problems with high dependencies between variables. Given that the LM-CMA-ES is rotationally invariant, its performance on both separable and non-separable problems is comparable, but the former is cheaper to compute.

5. DISCUSSION AND CONCLUSION

This paper presents a new approach to efficiently store and exploit the information about dependencies between decision variables of large scale optimization problems. It allows to reconstruct the Cholesky factor and its inverse using $m \ll n$ direction vectors that turns out to be sufficient to obtain good performance on large scale problems with highly-depended variables. The implementation of this approach in the LM-CMA-ES algorithm makes it possible to optimize a 1 million dimensional problem while learning dependencies between variables at a cost of about 0.1 second per function evaluation on an ordinary machine. Indeed, one should not plan to easily find a global optimum in such a huge search space, but some local optimization/tuning seems reasonable, e.g., in Machine Learning problems.

The proposed LM-CMA-ES algorithm is based on the *population success rule* which looks promising and requires further theoretical and empirical investigations. It should be studied as well whether it can be claimed to represent a general case of the 1/5th success rule. More experiments are required to investigate whether and when the lack of rank-μ update is a limitation.

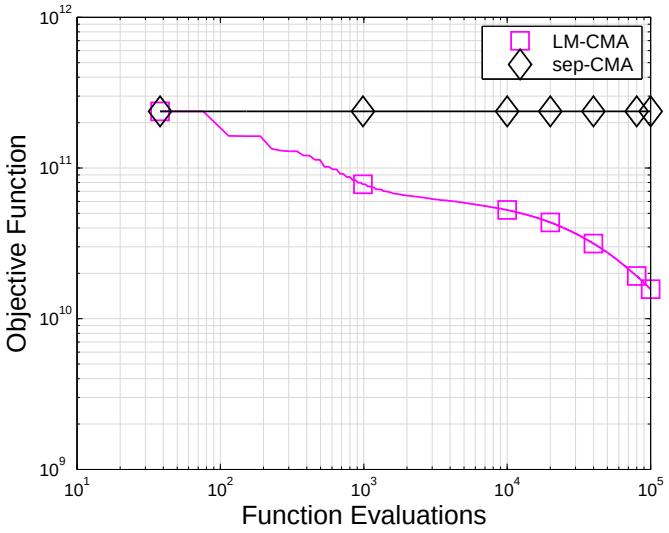

Figure 4: LM-CMA-ES and sep-CMA-ES on separable 100,000-dimensional Ellipsoid problem. The sep-CMA-ES divergences after the first generation (the best fitness is shown). Note that the LM-CMA-ES is rotationally invariant, therefore a similar performance is expected on 100,000-dimensional rotated Ellipsoid.

All parameters chosen for the algorithm were tuned only moderately and *specifically* for large n and might require a significant revision to address a wider set of optimization problems commonly used for EAs. However, we suppose that the performance on the Ellipsoid function is already worth a closer scientific investigation. We envision that several directions may further improve the algorithm: i) adaptation of m within a fixed range, the impact of m itself should be studied as well, ii) since the population success rule does not make any assumptions about the sampling distribution, the Gaussian sampling can be removed that would further speed-up the algorithm (e.g., to replace CSA by PSR in CMA-ES).

The speculations about a possibility of having CMA-ES like evolutionary processes going on in nature often end up around a hypothesis that there is no such a thing in natural evolution as a full covariance matrix and its update. One may suppose that only a limited number of direction vectors is stored to adjust the mutation in promising directions.

6. REFERENCES

[1] A. Auger, D. Brockhoff, N. Hansen, et al. Benchmarking the local metamodel CMA-ES on the noiseless BBOB'2013 test bed. In *GECCO (Companion), workshop on Black-Box Optimization Benchmarking (BBOB'2013)*, pages 1225–1232, 2013.

[2] A. Auger, S. Finck, N. Hansen, and R. Ros. BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RR-7215, INRIA, 2010.

[3] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[4] O. A. Elhara, A. Auger, and N. Hansen. A median success rule for non-elitist evolution strategies: Study of feasibility. In *Genetic and Evolutionary Computation Conference*, 2013.

[5] S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15:617–644, 2009.

[6] N. Hansen, S. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[7] N. Hansen, A. S. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *Evolutionary Computation, IEEE Transactions on*, 13(1):180–197, 2009.

[8] N. Hansen and A. Ostermeier. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In *International Conference on Evolutionary Computation*, pages 312–317, 1996.

[9] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evol. Comput.*, 9(2):159–195, June 2001.

[10] N. Hansen and R. Ros. Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed. In *Genetic and Evolutionary Computation Conference*, pages 1673–1680. ACM, 2010.

[11] N. Hansen, D. V. Arnold, and A. Auger. Evolution Strategies. In J. Kacprzyk and W. Pedrycz, editors, *Handbook of Computational Intelligence*. Springer, 2013.

[12] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation*, 15(1):1–28, 2007.

[13] G. A. Jastrebski and D. V. Arnold. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *IEEE Congress on Evolutionary Computation*, pages 2814–2821, 2006.

[14] J. N. Knight and M. Lunacek. Reducing the space-time complexity of the CMA-ES. In *Genetic and Evolutionary Computation Conference*, pages 658–665. ACM, 2007.

[15] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[16] I. Loshchilov. CMA-ES with restarts for solving CEC 2013 benchmark problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 369–376. IEEE, 2013.

[17] I. Loshchilov, M. Schoenauer, and M. Sebag. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Genetic and Evolutionary Computation Conference*, pages 321–328. ACM, 2012.

[18] I. Loshchilov, M. Schoenauer, and M. Sebag. Bi-population CMA-ES agorithms with surrogate models and line searches. In *Genetic and Evolutionary Computation Conference*, pages 1177–1184. ACM, 2013.

[19] J. Nocedal. Updating quasi-newton matrices with limited storage. *Math. of computation*, 35(151):773–782, 1980.

[20] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[21] R. Ros and N. Hansen. A simple modification in CMA-ES achieving and space complexity. In *Parallel Problem Solving from Nature–PPSN X*, pages 296–305. 2008.

[22] D. F. Shanno. Conditioning of Quasi-Newton Methods for Function Minimization. *Math. of Computation*, 24(111):647–656, 1970.

[23] Y. Sun, F. Gomez, T. Schaul, and J. Schmidhuber. A linear time natural evolution strategy for non-separable functions. *arXiv preprint arXiv:1106.1998*, 2011.

[24] T. Suttorp, N. Hansen, and C. Igel. Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 75(2):167–197, 2009.