Reducing the Space-Time Complexity of the CMA-ES

James N. Knight and Monte Lunacek
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
nate@cs.colostate.edu, lunacek@cs.colostate.edu

ABSTRACT

A limited memory version of the covariance matrix adaptation evolution strategy (CMA-ES) is presented. This algorithm, L-CMA-ES, improves the space and time complexity of the CMA-ES algorithm. The L-CMA-ES uses the m eigenvectors and eigenvalues spanning the m-dimensional dominant subspace of the n-dimensional covariance matrix, C, describing the mutation distribution. The algorithm avoids explicit computation and storage of C resulting in space and time savings. The L-CMA-ES algorithm has a space complexity of $\mathcal{O}(nm)$ and a time complexity of $\mathcal{O}(nm^2)$. The algorithm is evaluated on a number of standard test functions. The results show that while the number of objective function evaluations needed to find a solution is often increased by using $m < n$ the increase in computational efficiency leads to a lower overall run time.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; G.1.6 [**Numerical Analysis**]: Optimization

General Terms

Algorithms

Keywords

Evolution Strategy, Covariance Matrix Adaptation, Complexity

1. INTRODUCTION

The CMA-ES algorithm, introduced by Hansen *et al.* [7], is an evolutionary strategy designed to detect and exploit the local structure of a function being optimized. Its design also makes it insensitive to rotations of the search space that can cause serious degradation in performance for other local search methods and evolutionary algorithms. CMA-ES has proved to be very effective on many well known test functions and on several real world applications. The design of CMA-ES, however, makes its application to high dimensional optimization problems expensive. The algorithm stores and updates an $n \times n$ covariance matrix where n is the problem dimension. Also, CMA-ES computes an eigen-value decomposition (EVD) of this covariance matrix, an $\mathcal{O}(n^3)$ operation. This EVD computation can make the application of the CMA-ES to high dimensional problems impractical.

To reduce the algorithm's complexity, Hansen *et al.* [7] suggest that the EVD only be computed every $\frac{n}{10}$ generations, instead of every generation. This effectively reduces the complexity of CMA-ES to $\mathcal{O}(n^2)$. However, computational complexity and data efficiency are often at odds in optimization. Reducing the computational complexity of each generation will likely increase the adaptation time of the distribution, resulting in an increased number of function evaluation calls. Thus, a trade-off exists between computation time due to evaluation of the function and computation time due to algorithm complexity. It is still unclear how this technique scales with problem dimension, and Hansen *et al.* [7] admit that for practical applications an update every generation is often necessary.

In this paper, we propose a modification to CMA-ES that reduces its space and time complexity in a more general way. A tunable integer parameter, m, is introduced that controls the computational complexity of CMA-ES. The computational complexity of the resulting algorithm is $\mathcal{O}(nm^2)$ and the storage requirements are $\mathcal{O}(nm)$. With $m = n$ the algorithm behaves like CMA-ES, and in fact is essentially the same algorithm. Choosing $m \ll n$ decreases the algorithm's computational complexity dramatically, but the data efficiency—the number of evaluations necessary—can be affected negatively. We explore this trade-off by directly controlling m, and comparing these results with CMA-ES on several benchmark functions. Our results show that the data efficiency of the algorithm degrades gracefully as m decreases. That is, there is an increase in the number of evaluations required to reach the cutoff as the complexity of the algorithm decreases. We also show that, in higher dimensional test problems, reducing the complexity of CMA-ES results in a more efficient overall algorithm, despite requiring more evaluation calls.

In the following section it is shown how the essential computations of the CMA-ES can be viewed as an eigen-vector decomposition updating problem. The limited memory evolution strategy with covariance matrix adaptation (L-CMA-ES) algorithm, built on this update procedure, is then intro-

GECCO'07, July 7-11, 2007, London, England, United Kingdom

duced. In Section 4, the L-CMA-ES is analyzed on a set of standard test functions in high dimension. The possibility of further reducing the algorithm's complexity is discussed in Section 5. We conclude the paper by discussing the relation of the L-CMA-ES to other evolution strategies in Section 6.

2. THE CMA-ES

The canonical *evolution strategy* is an iterative process where a population of μ distinct parents produce λ offspring based on mutation distributions around the parents. Intermediate recombination creates a single parent based on the average position of the current population. A (μ, λ) selection strategy considers only the best μ of the current population for recombination.

An evolution strategy with *Covariance Matrix Adaptation*, or CMA-ES, uses a fully parametrized multivariate normal distribution to represent the mutation distribution. The orientation and shape of the distribution are directly calculated based on the *evolution path* [7]. Given an n-dimensional optimization problem, the CMA-ES mutation distribution is described by a symmetric matrix $C \in \mathbb{R}^{n\times n}$, a mean vector $s \in \mathbb{R}^n$, and a step-size parameter $\sigma \in \mathbb{R}$. A random population is generated by drawing samples from the n-dimensional normal distribution, $\mathcal{N}(s, \sigma^2 C)$. Since the covariance matrix can be written in terms of its eigenvector decomposition, $C = BD^2B^T$, the generation of a sample population for the CMA-ES can be written as

$$\begin{aligned} z_i^{(g+1)} &\sim \mathcal{N}(0, I) \quad i \in [1, \lambda] \\ x_i^{(g+1)} &= s^{(g)} + \sigma^{(g)} B^{(g)} D^{(g)} z_i^{(g+1)}. \end{aligned} \tag{1}$$

Each iteration of the algorithm consists of this population generation step and a parameter adaptation step. The covariance matrix, C, is updated by

$$C^{(g+1)} = (1 - c_{\text{cov}})C^{(g)} + c_{\text{cov}} p_c^{(g+1)} (p_c^{(g+1)})^T \tag{2}$$

where c_{cov} is an algorithm-specific parameter. The path cumulation variable, p_c, is initialized to zero and is updated by

$$p_c^{(g+1)} = (1 - c_c)p_c^{(g)} + c_c^u \frac{c_w}{\sigma^{(g)}} \left(\langle x \rangle_w^{(g+1)} - \langle x \rangle_w^{(g)} \right)$$

where c_c^u and c_w are algorithm parameters and $\langle \cdot \rangle_w^{(g)}$ denotes a weighted mean of the μ best individuals in the gth population. The complete algorithm is listed in Figure 1. As given, CMA-ES requires $\mathcal{O}(n^2)$ space for the storage of C and $\mathcal{O}(n^3)$ time for the computation of B and D.

The covariance matrix adaptation in CMA-ES is similar to subspace tracking problems found in the signal processing literature. We will now show that it is not necessary to recompute a full EVD at each iteration. Given the EVD of $C^{(g)}$ as BD^2B^T the update to C can be written as

$$\begin{aligned} &(1 - c_{\text{cov}})C^{(g)} + c_{\text{cov}} p_c^{(g)} (p_c^{(g)})^T = \\ &\begin{bmatrix} B^{(g)} & p_c^{(g)} \end{bmatrix} \begin{bmatrix} (1 - c_{\text{cov}})(D^{(g)})^2 & 0 \\ 0 & c_{\text{cov}} \end{bmatrix} \begin{bmatrix} B^{(g)} & p_c^{(g)} \end{bmatrix}^T \end{aligned} \tag{3}$$

The EVD of (3) can be computed using the algorithm in Figure 2 which is adapted from [3].

The singular value decomposition updating algorithm developed in [3] can be derived for the current problem as follows. We drop the superscript (g) for the remainder of the discussion. The matrix $\begin{bmatrix} B & p_c \end{bmatrix}$ in (3) is rewritten as

$$\begin{bmatrix} B & p_c \end{bmatrix} = \begin{bmatrix} B & P \end{bmatrix} \begin{bmatrix} I & q \\ 0 & R_a \end{bmatrix}$$

where P, q, and R_a are defined in Figure 2. Note that the columns of B and the vector p are orthogonal. Now, (3) can be written as

$$\begin{aligned} &(1 - c_{\text{cov}})C + c_{\text{cov}} p_c p_c^T = \\ &\begin{bmatrix} B & P \end{bmatrix} \begin{bmatrix} I & q \\ 0 & R_a \end{bmatrix} \begin{bmatrix} (1 - c_{\text{cov}})D^2 & 0 \\ 0 & c_{\text{cov}} \end{bmatrix} \begin{bmatrix} I & q \\ 0 & R_a \end{bmatrix}^T \begin{bmatrix} B & P \end{bmatrix}^T. \end{aligned}$$

The matrix K is defined by

$$K = \begin{bmatrix} I & q \\ 0 & R_a \end{bmatrix} \begin{bmatrix} (1 - c_{\text{cov}})D^2 & 0 \\ 0 & c_{\text{cov}} \end{bmatrix} \begin{bmatrix} I & q \\ 0 & R_a \end{bmatrix}^T$$

or more conveniently as

$$K = \begin{bmatrix} \beta D^2 & 0 \\ 0 & 0 \end{bmatrix} + c_{\text{cov}} \begin{bmatrix} q \\ R_a \end{bmatrix} \begin{bmatrix} q \\ R_a \end{bmatrix}^T$$

with $\beta = 1 - c_{\text{cov}}$. Since K is symmetric it has an eigenvalue decomposition, $K = US^2U^T$. Finally, the eigen-decomposition of $(1 - c_{\text{cov}})C + c_{\text{cov}} p_c p_c^T$ can be computed as

$$\tilde{B} S^2 \tilde{B}^T = \left(\begin{bmatrix} B & P \end{bmatrix} U \right) S^2 \left(\begin{bmatrix} B & P \end{bmatrix} U \right)^T.$$

The final step of the algorithm is to select out the leading n-dimensional submatrix of $\tilde{B}$ and S.

As written the algorithm is less efficient than the standard CMA-ES procedure, since K has dimension $(n+1)\times(n+1)$. The special structure of K, however, can be exploited to reduce the complexity of the EVD. Computation of the EVD of a diagonal plus rank one matrix, like K, has been extensively researched. The computation is a key part of efficient algorithms for finding the EVD of symmetric tridiagonal matrices (see [5] and references therein). The resulting computation takes on the order of n^2 computations and does not require an explicit computation of K. Note however that the overall asymptotic complexity is still $\mathcal{O}(n^3)$ due to the

---

$s \in \mathbb{R}^n,\ \sigma \in \mathbb{R},\ C = I,\ B = I,\ D = I,\ p_c = 0,\ p_\sigma = 0$

$\chi_N = \text{E}\{||\mathcal{N}(0, I)||\}$

Until Converged

$$\begin{aligned}
&(1) \quad z_i &&\sim \mathcal{N}(0, I) \quad i \in [1, \lambda] \\
&(2) \quad x_i &&= s + \sigma BDz_i \\
&(3) \quad s &&= s + \sigma BD \langle z \rangle_\mu \\
&(4) \quad p_c &&= (1 - c_c)p_c + c_c^u \frac{c_w}{\sigma} \left(\langle x \rangle_w^{(g+1)} - \langle x \rangle_w^{(g)} \right) \\
&(5) \quad C &&= (1 - c_{\text{cov}})C + c_{\text{cov}} p_c p_c^T \\
&(6) \quad [B, D^2] &&= \text{EVD}(C) \\
&(7) \quad p_\sigma &&= (1 - c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_{eff}} \langle z \rangle_\mu \\
&(8) \quad \sigma &&= \sigma \exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{||p_\sigma||}{\chi_N} - 1 \right) \right)
\end{aligned}$$

Figure 1: The CMA-ES algorithm. The parameters c_c, c_c^u, c_{cov}, c_σ, d_σ, and μ_{eff} are described in [7].

(1) $q = B^T p_c$
(2) $p = p_c - Bq$
(3) $R_a = ||p||$
(4) $P = \frac{1}{R_a}\ p$
(5) $\beta = (1 - c_{\text{cov}})$
(6) $K = \begin{bmatrix} \beta D^2 & 0 \\ 0 & 0 \end{bmatrix} + c_{\text{cov}} \begin{bmatrix} q \\ R_a \end{bmatrix} \begin{bmatrix} q \\ R_a \end{bmatrix}^T$
(7) $[U, S^2] = \text{EVD}(K)$
(8) $\tilde{B} = \begin{bmatrix} B & P \end{bmatrix} U$
(9) $B = \tilde{B}(:, 1:n)$
(10) $D = S(1:n, 1:n)$

Figure 2: An algorithm for computing the EVD of a matrix plus rank-1 update. Using a special eigen-decomposition, Step 6 can be avoided. Matlab notation is used in Lines 9 and 10 to denote selection of submatrices.

matrix multiplication in Line 8. Empirical tests suggest that the updating algorithm runs slightly faster than the original implementation.

3. THE L-CMA-ES

The reformulation of CMA-ES as an eigen-decomposition updating problem does not immediately improve its asymptotic computational complexity. Here, we show how the asymptotic space and time complexity of the CMA-ES computations can be reduced by a modification to the updating algorithm. The L-CMA-ES is a modified version of CMA-ES that uses a reduced dimensional representation of the mutation distribution. The dimensionality of the representation is controlled by the parameter, m. In the L-CMA-ES, only the m dominant eigen-pairs of C are computed. The updating approach described in the previous section is used to avoid the explicit computation and storage of the covariance matrix C. When $m < n$ both computational and space savings are achieved. Note that this does not necessarily imply a reduction in the overall runtime of the optimization as this is also a function of the data efficiency of the algorithm. These issues are addressed in the next section.

The L-CMA-ES is derived from the algorithm in Figure 1 with the key modifications described presently. To begin, define the matrices $\hat{B} \in \mathbb{R}^{n \times m}$ and $\hat{D} \in \mathbb{R}^{m \times m}$, and initialize them by

$$\begin{aligned} \hat{B} &= \begin{bmatrix} I \\ 0 \end{bmatrix} \\ \hat{D} &= I. \end{aligned}$$

The generation of a new population from $\hat{B}$ and $\hat{D}$ requires a modification to Equation 1. The equation is rewritten as

$$\begin{aligned} z_i^{(g+1)} &\sim \mathcal{N}(0, I_{m \times m}) \\ x_i^{(g+1)} &= s^{(g)} + \sigma \hat{B}^{(g)} \hat{D}^{(g)} z_i^{(g+1)} \end{aligned} \tag{4}$$

for L-CMA-ES. The generation of a population according to Equation 4, however, is flawed in the following way: each new population will span only an m-dimensional subspace of the optimization problem's domain. Unless the optimization problem is rank deficient, this flaw will limit optimization to a subspace of the domain. For example, in a two dimensional problem with $m = 1$, the algorithm would search only in a line and thus could, in general, only find the optimum if the function was constant in one direction. The direction in which the population adapts would be fixed for the entire run of the algorithm.

To correct for this problem we would like to approximate the remaining eigenvalues by the smallest known eigenvalue, $\hat{d} = \hat{D}(m, m)$,

$$D \approx \begin{bmatrix} \hat{D} & 0 & \dots & 0 \\ 0 & \hat{d} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \hat{d} \end{bmatrix},$$

and use a set of vectors spanning the subspace orthogonal to the range of $\hat{B}$. On the other hand, it is essential to avoid computing this remaining set of vectors as this adds undesirable, additional computation. Instead, an n-dimensional isotropic distribution with variance $\hat{d}^2$ is used in conjunction with the m-dimensional distribution defined by $\hat{B}$ and $\hat{D}$. The mutation distribution in L-CMA-ES is thus described by

$$\begin{aligned} z_i^{(g+1)} &\sim \mathcal{N}(0, I_{m \times m}) \quad i \in [1, \lambda] \\ w_i^{(g+1)} &\sim \mathcal{N}(0, I_{n \times n}) \\ x_i^{(g+1)} &= s^{(g)} + \sigma \hat{B}^{(g)} \hat{D}^{(g)} z_i^{(g+1)} + \sigma \hat{d} w_i^{(g+1)}. \end{aligned} \tag{5}$$

The use of the scaled isotropic distribution results in a regularization of the low rank approximation to C given by $\hat{B}\hat{D}^2\hat{B}^T$. Consider that

$$\begin{aligned} \text{E}\{xx^T\} &= \text{E}\{(\sigma \hat{B}\hat{D}z + \sigma \hat{d}w)(\sigma \hat{B}\hat{D}z + \sigma \hat{d}w)^T\} \\ &= \text{E}\{\sigma^2(\hat{B}\hat{D}zz^T\hat{D}^T\hat{B}^T + \hat{d}^2 ww^T + \\ &\qquad \hat{d}\hat{B}\hat{D}zw^T + \hat{d}wz^T\hat{D}^T\hat{B}^T)\} \\ &= \sigma^2(\hat{B}\hat{D}\text{E}\{zz^T\}\hat{D}^T\hat{B}^T + \hat{d}^2\text{E}\{ww^T\} + \\ &\qquad \hat{d}\hat{B}\hat{D}\text{E}\{zw^T\} + \hat{d}\text{E}\{wz^T\}\hat{D}^T\hat{B}^T) \\ &= \sigma^2(\hat{B}\hat{D}^2\hat{B}^T + \hat{d}^2 I) \end{aligned}$$

which illustrates the connection between the mutation distributions of L-CMA-ES and CMA-ES.

Once a population has been generated, the variables p_c and p_σ are computed as shown in Figure 1. The matrices, $\hat{B}$ and $\hat{D}$, are then updated using a modified version of the algorithm in Figure 2. The algorithm is changed slightly by using $\hat{B}$ and $\hat{D}$ in place of B and D and using m in place of n in the last two lines. The matrix $\hat{B}$ will contain the m eigenvectors of C corresponding to the m largest eigenvalues. The matrix $\hat{D}$ has the m largest eigenvalues of C on the diagonal. Computing the EVD of K has a complexity of $\mathcal{O}(m^2)$ as discussed previously. The matrix multiplication in Line 8 of Figure 2 dominates the asymptotic complexity with order $\mathcal{O}(nm^2)$.

The final development in L-CMA-ES is a change in the computation of χ_N. This parameter influences the adaptation of σ as shown in line 7 of Figure 1. The parameter measures the expected magnitude of z (see [7] for a discus-

sion). For the CMA-ES this quantity is given by

$$\chi_N = \mathrm{E}\{||z||\} = \sqrt{2}\,\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)}.$$

If z is generated as in L-CMA-ES by

$$z \sim \mathcal{N}(0, I_{m\times m}) + \mathcal{N}(0, I_{n\times n})$$

then

$$\chi_N = \mathrm{E}\{||z||\} = \sqrt{2}\,\frac{\Gamma\left(\frac{n+m+1}{2}\right)}{\Gamma\left(\frac{n+m}{2}\right)}.$$

4. EMPIRICAL EVALUATION

The time (and space) complexity of L-CMA-ES can be directly controlled. When $m < n$, L-CMA-ES does less work per generation than CMA-ES. This decrease in complexity does not come for free; L-CMA-ES uses less information to update the mutation distribution. Intuitively, we would expect L-CMA-ES to require more generations to optimize a function, and it does. One of the goals in this section is to show that as m decreases, L-CMA-ES degrades "gracefully" in terms of the number of evaluations needed to reach the global optima.

On higher dimensional problems, the complexity of CMA-ES becomes a liability that is difficult to ignore, but this is hidden in a comparison that only considers the number of evaluations. There is an observable time difference between CMA-ES and L-CMA-ES and in order to understand the actual difference in overall efficiency—the main reason to prefer L-CMA-ES—we must measure CPU time. The second goal in this section is to show that L-CMA-ES, with $m < n$, is more efficient in high dimensions than CMA-ES.

Normally, optimization algorithms are evaluated and compared based on how many evaluation calls they make to the objective function. This is the most accepted, and generally the most fair, way of comparing algorithm efficiency. An empirical comparison using CPU time generally says more about the implementation details of an algorithm than it does about how well a particular algorithm optimizes a given problem. Needless to say, great care must be taken when comparing clock cycles. In order to mitigate the undesirable affects of comparing CPU times, we implemented both algorithms in C++ using the same base code. The only implementation difference in the algorithms is the way in which they update the distribution. Isolating the distribution update as the only difference in implementation results in a fair comparison in terms of time complexity. In the standard CMA-ES implementation a call to the LAPACK routine syevr [4] is made to compute the eigen-decomposition of the matrix C. The rank one update is performed using the BLAS routine dsyr. In the L-CMA-ES implementation the eigen-decomposition of K was done using a slightly modified version of the LAPACK routines laed7 and laed8[1]. All other matrix operations were performed by the BLAS library. All experiments were run on Dell Precision 530s with 1.5Ghz Xeon processors and 512Mb of RAM.

4.1 Evaluation of Data Efficiency

We compare each algorithm on three standard unimodal test functions in 30 dimensions: the sphere, the ellipsoid, and Rosenbrock's function. Table 1 lists the exact details of each problem. Although the sphere is separable, symmetric, equally-scaled, and unimodal, Bäck points out the importance of simple functions like the sphere: "..., before we can expect an algorithm to be successful in the case of hard problems, it has to demonstrate that it does not fail to work on simple problems" [1]. The ellipsoid is similar to the sphere in all its properties except that it is scaled differently in each dimension, which creates ridges in the landscape. Rosenbrock's function is the "classic" ridge problem. It is both non-symmetric and non-separable.

We ran each algorithm with the population size of $\lambda = 4 + \lfloor 3\log(n) \rfloor$, which is the default suggested in [7]. For the thirty dimension problems $\lambda = 14$. The initial step-size was set to $\sigma = 0.5$ times the length of a side of the bounding box on the functions domain. The other algorithm parameters were set as suggested in [7]. Three different values of m were considered: $\lfloor\sqrt{n}\rfloor, \lfloor n/2 \rfloor, n$. With $n = 30$, this corresponds to $m = 5, 15$, and 30. Figures 3, 4, and 5 show the behavior of each algorithm on the sphere, ellipsoid, and Rosenbrock's function, respectively, as best fitness achieved versus the number of evaluations.

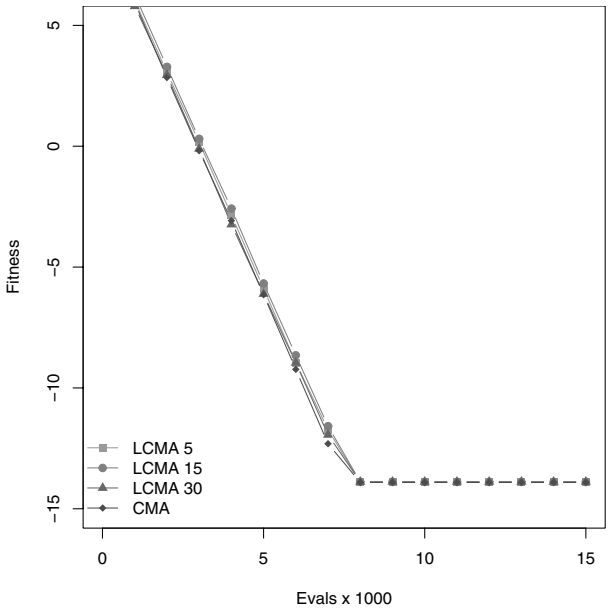

Figure 3: L-CMA-ES and CMA-ES on the 30 dimensional sphere function. There is no significant difference in performance between the three algorithms.

On the sphere function there is no significant difference in the performance of the algorithms, including CMA-ES. Since the initial mutation distribution of all algorithms, an isotropic normal distribution, is optimal for this function, this experiment tests the modified step-length adaptation used in L-CMA-ES. In Section 3 we described a modification to the value of χ_N, necessary because of the change in how the population is generated, that affected the adaptation of σ. The results of this experiment show that this modification did not negatively affect the algorithm.

On the ellipsoid function the story is different. When $m = n$, the number of evaluations required to reach the optimal solutions is indistinguishable when compared with

[1]Implementations of the two algorithms are available for download at http://www.cs.colostate.edu/∼nate/lcmaes.

Name	Function	Domain
Sphere	$f(x_i\mid_{i=1,n}) = \sum_{i=1}^{n} x_i^2$	$[-5.0, 5.0]$
Ellipsoid	$f(x_i\mid_{i=1,n}) = \sum_{i=1}^{n} (1000^{\frac{i-1}{n-1}} x_i)^2$	$[-5.0, 5.0]$
Rosenbrock	$f(x_i,\mid_{i=1,n-1}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})2 + (1 - x_i)^2$	$[-2.0, 2.0]$

Table 1: The test functions used in this paper.

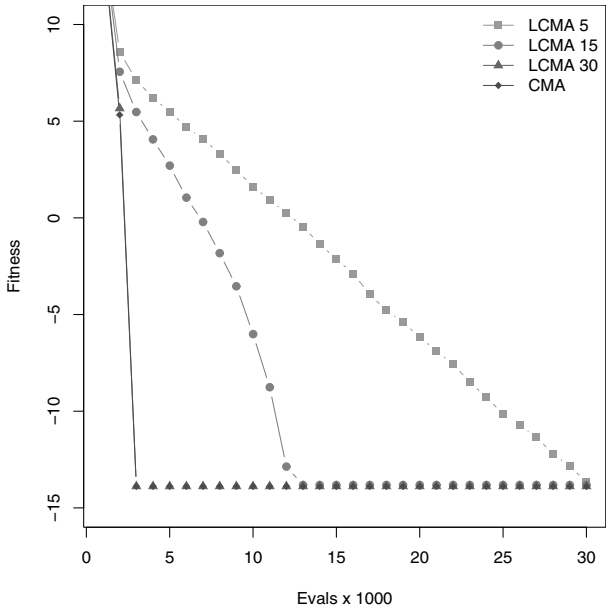

Figure 4: L-CMA-ES and CMA-ES on the 30 dimensional ellipsoid function. Notice that as m decreases, L-CMA-ES requires significantly more evaluations to converge.

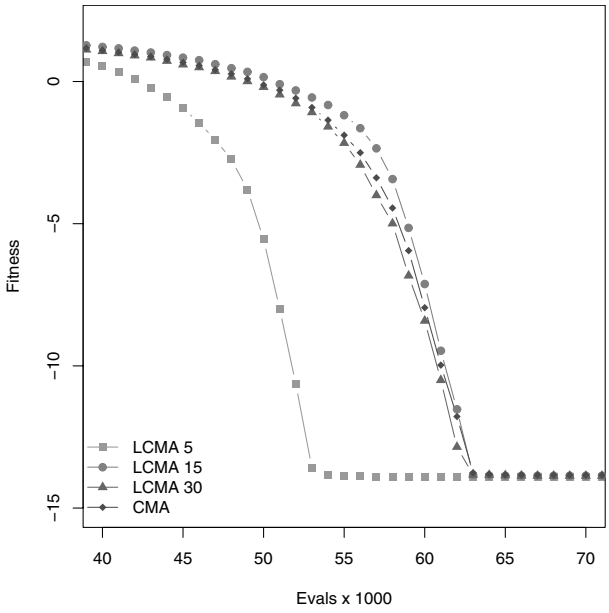

Figure 5: L-CMA-ES and CMA-ES on the 30 dimensional Rosenbrock function. Here the most data efficient algorithm is L-CMA-ES using $m = 5$. It reaches the solution in the fewest number of evaluations, and is also the most computationally efficient algorithm.

CMA-ES. This is expected since both algorithms are using the same information when updating the mutation distribution. As m decreases, however, L-CMA-ES becomes less efficient in terms of the number of evaluation calls. Note, however, that it never fails to reach the optimal solution. The decrease in data efficiency appears to be smooth with respect to changes in the parameter m. As m decreases the number of evaluations needed increases. In fact, further experimentation with other values of m shows a gradual increase in the number of necessary evaluations as m goes from 1 to n.

In the third experiment CMA-ES and the L-CMA-ES are compared on Rosenbrock's function. We observe that the smallest m value is actually the most data efficient on Rosenbrock's function. This result is unexpected given the behavior of L-CMA-ES on the ellipsoid function. Certain features of the function contribute to this behavior. Recently, Voigt has shown that the curvature of the ridge, not the scale, is the primary feature that makes the high dimensional Rosenbrock problem difficult [10]. In light of this, we conjecture that the accuracy of the mutation distribution is less important than the adaptability of the search direction. Although small values of m may create a less accurate model of the "true" CMA-ES mutation distribution, the distribution itself has less parameters and may adapt more quickly to the curvature in Rosenbrock's function.

Also, we note that the definition of Rosenbrock's function is such that there is only local parameter interaction as opposed to the global interaction in functions like the ellipsoid. An example run of the CMA-ES algorithm on Rosenbrock's in 30 dimensions is shown in Figure 6. The structure of Rosenbrock's function contributes to a particular behavior in CMA-ES. After an initial phase, the algorithm proceeds to optimize overlapping subsets of parameters. The L-CMA-ES behaves in essentially the same way. It appears that only information about the interaction between a small set of parameters is needed at any particular time. The L-CMA-ES with $m = 5$ does not need to update and adapt unnecessary parameters and thus is more efficient on this function.

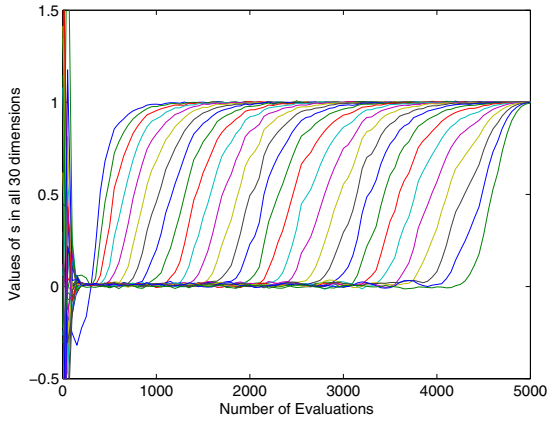

Figure 6: The structure of Rosenbrock's function contributes to a particular behavior in CMA-ES. After an initial phase, the algorithm proceeds to optimize an overlapping set of parameters.

4.2 Measuring CPU-time

The ellipsoid presents the greatest challenge to L-CMA-ES in terms of the necessary number of function calls, especially with small values of m. Referring back to Figure 4, notice that the number of evaluations for $m = 5$ is substantial. With the following experiment we answer the question: how does the complexity of each algorithm, measured in terms of CPU time, scale with dimensionality.

We measured the amount of time and the number of evaluations needed for the algorithms to reach 1×10^{-6} on the ellipsoid in 50, 100, 150, and 200 dimensions. The average number of evaluations, based on twenty-four trials, are displayed in Figure 7. Notice that as dimensionality increases, L-CMA-ES continues to require the greatest number of evaluations, which increases with a decreasing m. Again, this is not surprising because lower m values are considering less information and require more generations to converge. Note also that the rate at which the number of evaluations grows appears to be linear for all three algorithms.

Figure 8 shows the averaged CPU times for the twenty-four trials. The opposite is true here; as dimensionality increases, the most efficient algorithm is L-CMA-ES. The configuration with $m = \sqrt{n}$ performs the best. Here, the rate at which the computation time grows for each algorithm is different. At some value of n the computational complexity will always come to dominate the total run time of the CMA-ES. For $m = \sqrt{n}$ the computational complexity of L-CMA-ES grows at a rate of n^2 as opposed to the n^3 of the original algorithm. The difference in the rate of growths of the computational complexity, coupled with the relatively constant rates of growth in the number of evaluations, leads to the improvement in overall performance for the L-CMA-ES.

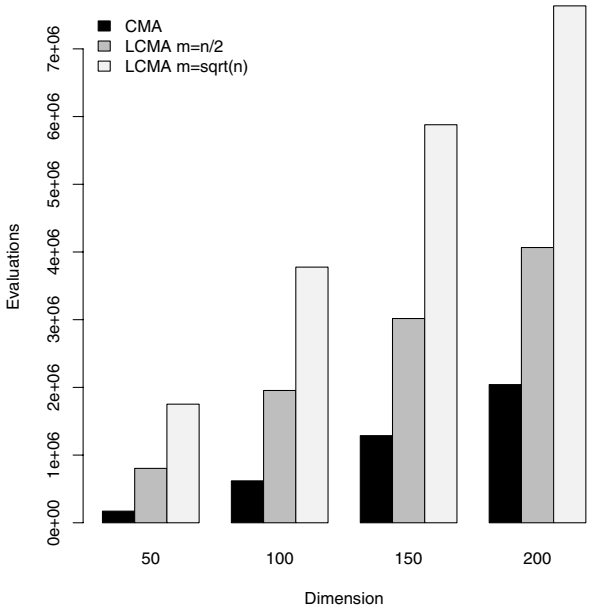

Figure 7: The average number of evaluations required for L-CMA-ES and CMA-ES on the ellipsoid function in high dimensions.

The point here is not to suggest that L-CMA-ES is a better algorithm than CMA-ES. Instead, we are documenting that there are high-dimensional, relatively fast objec-

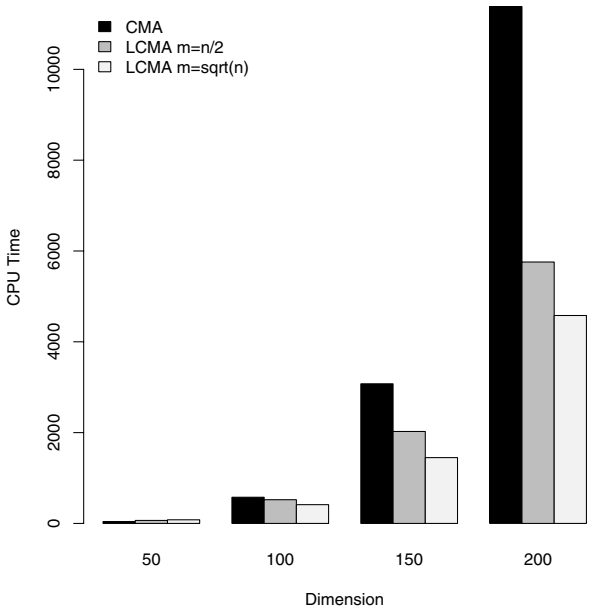

Figure 8: The average CPU time required for L-CMA-ES and CMA-ES on the ellipsoid function in high dimensions.

tive functions where the complexity of CMA-ES hinders its ability to efficiently solve these problems. This is where L-CMA-ES is most appropriately used. For problems with more expensive evaluation functions, the cross over point at which it becomes more efficient to use $m < n$ will change. Unless these problems are of very high dimension it may be more efficient to keep $m = n$.

5. FUTURE WORK

We began the development of the L-CMA-ES algorithm by noting the connection between adaptation of the CMA-ES mutation distribution and subspace tracking problems. In this development we chose to use an algorithm capable of tracking the m dominant eigen-values and eigen-vectors of C exactly. A further improvement in the computational complexity of the algorithm can be achieved by only approximately tracking the subspace of interest. In particular we are currently exploring use of the YAST algorithm [2], that has a computational complexity of $\mathcal{O}(nm)$. The effect of the approximation on the data efficiency of the L-CMA-ES has yet to be explored.

In [6], Hansen *et al.* modify the CMA-ES mutation distribution update by adding in a rank-μ update to C, built from the μ best members of the current population. Equation 2 is modified such that

$$
\begin{aligned}
Z^{(g+1)} &= \frac{1}{\mu} \sum_{i \in I_{\text{sel}}} x_i^{(g+1)} \left(x_i^{(g+1)} \right)^T \\
C^{(g+1)} &= (1 - c_{\text{cov}}) C^{(g)} + \\
&\quad c_{\text{cov}} \left(\alpha_{\text{cov}} p_c^{(g+1)} (p_c^{(g+1)})^T + (1 - \alpha_{\text{cov}}) Z^{(g+1)} \right)
\end{aligned}
$$

where I_{sel} contains the indices of the μ best individuals. The modification allows the information contained in large populations to be exploited and leads, on a set of test functions, to a decrease in the number of function evaluations necessary to reach a fixed level of fitness. It is also demonstrated that larger population sizes improve the performance of CMA-ES on multi-modal functions. In this paper we have restricted our attention to unimodal functions and to the original implementation of the CMA-ES for the initial exploration of the proposed technique. Nevertheless, the rank-μ update proposed in [6] can be incorporated into the L-CMA-ES. The addition of this update term will affect the number of computations by requiring a more general EVD procedure to be used on Line 7 of the update algorithm in Figure 2. Currently an efficient algorithm with complexity $\mathcal{O}(m^2)$ can be used, but the more general procedure would have complexity $\mathcal{O}(m^3)$. The overall asymptotic complexity of $\mathcal{O}(nm^2)$ would be preserved, but the trade-off analyzed in the previous sections would be affected. An analysis of the effect of these changes is left for future work.

6. SUMMARY

We have presented an algorithm that reduces the space-time computational complexity of the CMA-ES algorithm. The reduction in complexity can cause a decrease in the data efficiency of the algorithm. This trade-off between computational complexity and data efficiency is controlled by an adjustable parameter. The overall complexity of the algorithm can be improved for certain values of this parameter that are problem dependent. While there is yet no way to determine the optimal setting of this parameter *a priori*, understanding the construction of the algorithm can guide its selection. As the parameter m decreases the amount of parameter interaction that can be modeled decreases. For functions with strong coupling between the parameters, values of m near n will tend to be best. On the other hand, for functions with weaker coupling, such as Rosenbrock's function, smaller values of m can improve overall performance as was seen in the experiments. Further exploration into the relationship between the coupling of parameters and the value of m will give insight on how to set this parameter. Also, as was suggested in [3], we believe it might be possible to adapt this parameter during optimization.

The algorithm we have presented is connected to the whole spectrum of derandomized adaptation evolution strategies. It is obvious that for $m = n$ the algorithm is equivalent to the original CMA-ES formulation. When the parameter m equals zero, the algorithm is a cumulative step-size adaptation (CSA) style algorithm with an isotropic mutation distribution [8]. When $m = 1$ the algorithm is quite similar, in behavior and complexity, to the main vector adaptation (MVA) algorithm described in [9]. While a direct comparison was not performed, the results on the ellipsoid and Rosenbrock's function presented in [9] agree with results produced by L-CMA-ES using $m = 1$. Specifically, the MVA, like L-CMA-ES, performs poorly on the ellipsoid and similar to CMA on Rosenbrock's function. The L-CMA-ES algorithm fills the gap between MVA-ES and CMA-ES by allowing the number of adaptation directions to vary from one to n. The algorithm is a flexible tool for optimizing functions with various levels of parameter interaction.

7. REFERENCES

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[2] R. Badeau, B. David, and G. Richard. Yet another subspace tracker. In *Proceedings of the IEEE*

International Conference on Acoustics, Speech, and Signal Processing, 2005.

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

[4] I. Dhillon, B. Parlett, and C. Vömel. Lapack working note 162: The design and implementation of the mrrr algorithm. Technical Report UT-CS-04-541, 2004.

[5] M. Gu and S. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal on Matrix Analysis and Its Applications*, 15(4):1266–1276, 1994.

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

[8] S. Kern, S. Müller, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3(1):77–112.

[9] J. Poland and A. Zell. Main vector adaptation: A CMA variant with linear time and space complexity. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1050–1055, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[10] H.-M. Voigt. On some difficulties in local evolutionary search. In *IEEE*, 1999.