Internal Report 2012–13 August 2012

Universiteit Leiden
Opleiding Informatica

On the scalability of evolution strategies on high-dimensional problems

Piet van Hekke

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

On the scalability of evolution strategies on high-dimensional problems

Piet van Hekke - 0725870
Supervisor: prof. dr. Thomas Bäck

August 18, 2012

Contents

1 Introduction

Evolutionary optimization has achieved great success on lots of numerical and combinatorial optimization problems. On problems that are known to be computationally hard, good result have been found. However, most types of evolutionary algorithms (EAs) suffer from the 'curse of dimensionality', meaning that their performance is good on low dimensional problems, but deteriorates as the dimensionality of the search space increases.

For every objective function there is a certain region around the global optimum which is considered the optimality region. Considering a general stochastic approach, the probablity of generating a sample in the optimality region is determined by the volume of the optimality region divided by the volume of the whole search space. This probablity decreases exponentially as the dimensionality of the search space increases. Altough EAs don't use a fully stochastic (Monte Carlo) approach, they still suffer heavily from this dimensionality problem [1]. In this thesis we will focus on Evolution Strategies (ES) with respect to this problem.

To deal with the curse of dimensionality, various modifications of Evolution Strategies have been proposed, which each behave differently on high dimensional problems. The most important variable to describe the behaviour of an EA on a certain type of problems is the convergence speed, the number of function evaluations it takes to find a good solution. In this study, a comparative analysis of various ES with respect to the dimensionality, is done. The analysis is limited to a series of common adaptations of ES, the (1+1)-ES, Derandomized ES (type 1 - 3) and the Covariance Matrix Adaptation-ES.

The theoretical background of the ES adaptations is given in Section 2. Section 3 covers the test problems to be used for the analysis. In Section 4, the implementation, parameters and experimental setup are discussed. In Section 5, the results are covered and discussed, to be concluded in Section 6.

2 Evolution Strategies

Since the introduction of Evolution Strategies in the early 1960, many improvements and adaptations have been proposed, aiming at various properties of the algorithm, such as convergence speed, robustness and computational speed. Especially the mutative stepsize control was investigated intensively. Implementations using individual stepsizes or a full covariance matrix turned out to need larger populations to come up with a good solution. To deal with this problem, a new series of ES implementations was suggested: the Derandomized Evolution Strategies (DES). In this section the theory of these algorithms will be discussed. We will start with the classical implementation of the ES, also known as Schwefel's implementation. Then a very simple (1+1)-ES, three DES and the state-of-the-art, completely derandomized Covariance Matrix Adaptation-ES (CMA-ES) will be discussed.

2.1 Schwefel's implementation

The classical ES implementation, as proposed by Rechenberg and Schwefel and extended by numerous others is given in its general form in Algorithm 1.

Algorithm 1 Schwefel's $(\mu +/, \lambda)$-ES

1: $t \leftarrow 0$
2: $P(t) \leftarrow \text{Init}()$ $\{P(t) \in S_\mu : \text{Set of solutions}\}$
3: Evaluate$(P(t))$
4: **while** $t < t_{max}$ **do**
5: Select ν mating parents from $P(t)$ {Marriage}
6: $\vec{a}'_k(t) := \text{Recombine}(P(t))\ \forall k \in \{1, ..., \lambda\}$ {Recombination}
7: $\vec{a}''_k(t) := \text{Mutate}(\vec{a}'_k(t))\ \forall k \in \{1, ..., \lambda\}$ {Mutation}
8: Evaluate $(P'(t) := \{\vec{a}''_1(t), ..., \vec{a}''_\lambda(t)\})(\{f(\vec{x}''_1(t)), ..., f(\vec{x}''_\lambda(t))\})$
9: **if** (μ, λ)-ES **then**
10: Select$(P'(t))$
11: **else if** $(\mu + \lambda)$-ES **then**
12: Select$(P'(t) \cup P(t))$
13: **end if**
14: $t \leftarrow t + 1$
15: **end while**

The algorithm starts with randomly generating μ initial solutions. A solution consists of n object variables and some strategy parameters to determine the step-size, which can either be one global step-size σ, a vector $\sigma(n)$, containing a step-size parameter for each of the n parameters, or a matrix $\mathbf{C}$, containing $\frac{1}{2}n(n+1)$ free strategy parameters. In each generation, an offspring population with size λ is generated, using recombination.

Recombination can be done in two ways:

1. For each variable choose ν parents and randomly pick one of the parents.
2. For each variable choose ν parents and take the average.

The mutation operator is the dominant variation operator in an ES. The object variables as well as the step-size parameters are mutated, based on a normal distribution. The step-size parameters are updated first, with

$$\sigma'_i = \sigma_i \cdot e^{\tau' \cdot \mathcal{N}(0,1) + \tau \cdot \mathcal{N}_i(0,1)}$$

where the individual step-size mutation can be excluded in an implementation with only a global step-size. The object variables are updated with

$$x'_i = x_i + \vec{\mathcal{N}}(\vec{0}, \mathbf{C}')$$

where $\mathbf{C}'$ can be replaced with 1 in an implementation without a covariance matrix.

Selection can be done in two ways. In a (μ, λ)-ES, the μ best individuals of the offspring are selected as new parent population. In a $(\mu + \lambda)$-ES, the μ best individuals of parents and offspring combined are selected as new parents. This leads to a more elitist strategy, allowing parents to survive for a large number of generations.

This algorithm forms a basis of the adaptations used in this study. In the next sections, the algorithms that are compared in this study, are presented with a short description, pseudocode and — most importantly — the way mutation is handled.

2.2 (1+1)-ES

As a tool for mathematical analysis, Rechenberg considered a very simple (1+1) Evolution Strategy without self-adaptation and recombination. Because of the small population, the algorithm can easily be analyzed. The result of his analysis was the famous 1/5-success rule: only one in five mutations should be succesfull, e.g. should lead to a better fitness value. If the success rate is higher, increase the mutation step-size, if it's smaller, decrease the step-size. The pseudocode of the (1+1)-ES with the 1/5-rule is given in Algorithm 2 [2].

Algorithm 2 $(1 + 1)$-ES

$t \leftarrow 0$
$P(t) \leftarrow \text{Init}()$ $\{P(t) \in S_\mu : \text{Set of solutions}\}$
Evaluate$(P(t))$
while $t < t_{max}$ **do**
  $\vec{x}(t) := \text{Mutate}\{\vec{x}(t-1)\}$ with step-size σ
  Evaluate $(P'(t) := \{\vec{x}(t)\}) : \{f(\vec{x}(t))\}$
  Select$(P'(t) \cup P(t))$
  if $t \bmod n = 0$ **then**

$$\sigma = \begin{cases} \sigma(t-n)/c & \text{if } p_s > 1/5 \\ \sigma(t-n) \cdot c & \text{if } p_s < 1/5 \\ \sigma(t-n) & \text{if } p_s = 1/5 \end{cases}$$

  else
    $\sigma(t) = \sigma(t-1)$
  end if
  $t \leftarrow t + 1$
end while

2.3 DR1

In order to increase the convergence speed of ES, the first derandomization attempt [3] couples the succesfull mutations to the selection of decision parameters. Offspring is created with

$$\vec{x}_k'^g = \vec{x}_k^g + \xi_k \delta^g \vec{z}_k, \quad \forall k \in \{1, ..., \lambda\}$$

as the mutation function and the parameter δ is updated with

$$\delta^{g+1} = \delta^g \cdot (\xi_{sel})^\beta$$

where $\xi_k \in \{\frac{5}{7}, \frac{7}{5}\}$ are constants with $\mathcal{P}(\xi_k = \frac{5}{7}) = \mathcal{P}(\xi_k = \frac{7}{5}) = \frac{1}{2}$ and *sel* refers to the selected individual. This approach shows a significantly faster convergence.

2.4 DR2

The second derandomized ES variant, as proposed in [4], is an extension of DR1. It adapts both the global step-size as well as the individual step-size. The mutation step for this approach reads:

$$\vec{x}'^g = \vec{x}^g + \delta^g \vec{\delta}^g_{scal} \vec{z}_k$$

A quasi-memory vector $\vec{Z}$ is introduced:

$$\vec{Z}^{g'} = (1-c)\vec{Z}^g + c\vec{z}_{sel}$$

The adaptation of the strategy parameters:

$$\delta^{g'} = \delta^g \cdot \left(exp\left(\frac{\|\vec{Z}^g\|}{\sqrt{n}\sqrt{\frac{c}{2-c}}} - 1 + \frac{1}{5n} \right) \right)^\beta$$

$$\vec{\delta}^{g'} = \vec{\delta}^g \cdot \left(\frac{|\vec{Z}^g|}{\sqrt{\frac{c}{2-c}}} + b \right)^{\beta_{scal}} \qquad |\vec{Z}^g| = (|Z^g_1|, |Z^g_2|, ..., |Z^g_n|)$$

The memory vector allows the omitting of the stochastic element in de adaptation of the strategy parameters. They are updated by means of succesfull variations, instead of random steps.

2.5 DR3

The third variant, also known as the Generation Set Adapatation [5]. This variant expands the quasi-memory vector to a full matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$, in order to achieve invariance with respect to the coordinate system and the scaling of variables. The mutation step is formulated as

$$\vec{x}^{g'} = \vec{x}^g + \delta^g \xi_k \vec{u}_k$$

$$\vec{z}_k = c_m \mathbf{B}^g \cdot \vec{z}_k$$

The memory matrix is updated with

$$\mathbf{B}^g = \left(\vec{b}^g_1, ..., \vec{b}^g_m \right)$$

$$\vec{b}^{g'}_1 = (1-c) \cdot \vec{b}^g_1 + c \cdot (c_u \xi_{sel} \vec{y}_{sel})$$

$$\vec{b}_{i+1}^{g'} = \vec{b}^{g}$$

and the step-size is updated with

$$\delta^{g'} = \delta^{g}(\xi_{sel})^{\beta}$$

where $\mathcal{P}(\xi_k = \frac{2}{3}) = \mathcal{P}(\xi_k = \frac{3}{2}) = \frac{1}{2}$.

2.6 CMA-ES

The state-of-the-art ES nowadays is the Covariance Matrix Adaptation. The original version is proposed in [6]. Various modifications have led to the methods as stated below for a $(1, 10)$-CMA-ES, explained in general and in more detail in [7].

The mutation step for each individual is defined as

$$\vec{x}_k^{g+1} = \vec{x}^{g} + \sigma^{g}\mathbf{B}^{g}\mathbf{C}^{g}\vec{z}_k^{g+1}$$

with $\vec{z}_k \sim \vec{\mathcal{N}}(\vec{0}, \mathbf{I})$ and with $\mathbf{B}$ and $\mathbf{D}$ as results of the eigendecomposition of $\mathbf{C}$.

The covariance matrix is updated using a so called *evolution path*, which contains the history of the steps taken using an exponentially weighted moving average:

$$\vec{p}_c' = (1 - c_c)\vec{p}_c + \sqrt{c_c(2 - c_c)} \cdot \mathbf{B} \cdot \mathbf{D}\vec{z}_{sel}$$

This is used to update the covariance matrix:

$$\mathbf{C}' = (1 - c_{cov}) \cdot \mathbf{C} + c_{cov} \cdot \vec{p}_c\vec{p}_c^{T}$$

To guarantee an increasing variance in all directions when necessary, global step-size control is needed. This is done by measuring the length of the evolution path. If the evolution path is longer than expected, the steps are highly parallel, so the step-size should be increased. If it is shorter than expected, evolution steps are done in various directions, with low convergence, so the step-size should be decreased. To do so, a so called *conjugate evolution path* is used, updated by:

$$\vec{p}_\sigma' = (1 - c_\sigma)\vec{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)} \cdot \mathbf{B} \cdot \vec{z}_{sel}$$

Using this information, the global step-size is updated with:

$$\sigma' = \sigma \times exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\vec{p}_\sigma\|}{E[\|\mathcal{N}(0, \mathbf{I}\|]} - 1\right)\right)$$

3 Test problems

There is a wide range of easy scalable continuous functions imaginable, which can be used as test problems for ES. For example the CEC'2010 Special Session on Large Scale Global Optimization provides a test suite with 20 benchmark functions [8]. Complexity

can differ greatly among these functions. Some functions can be solved by an ES relatively easy, others take a lot more iterations. There are two important properties that determine solvability.

First, the modality of the function has a strong influence. A unimodal function, that is, a function with only one, global optimum or minimum, is easier solvable than a multimodal function, that is, a function with more than one local optimum and (usually) one global optimum. On a unimodal function, the ES, once it is converging in the right direction, will easily find the optimum, because there is no chance of getting stuck in local optima. On multimodal functions however, the ES can get stuck in local optima, which can greatly increase the number of iterations needed to reach the global optimum.

Second, the separability of the objective function has an influence on the solvability. Functions are separable if they can be expressed as a product of two or more functions, which means that the input variable for the different dimensions are fully independent of each other. Using different mutation rates for each dimension, these functions can be optimized a lot faster than nonseparable functions.

For this project, we use the sphere function [9] and Rosenbrock's function as proposed in [10]. The sphere function is unimodal and separable, and therefore easily optimizable. Rosenbrock's function is multimodal and nonseparable, so it should be a lot harder to solve by ES.

3.1 Sphere function

$$f(\vec{x}) = \sum_{i=1}^{N} x_i^2$$

The sphere function was proposed by K.A. de Jong in [9] as a simple test function for optimization algorithms. It is a continuous, convex, unimodal, easy scalable quadratic function with a minimum of zero at the origin. Because it is simple and symmetric, the sphere function is a perfect first test function for optimization algorithms. The main reasons the sphere function is used in this study are its simplicity and scalability. The function behaves the same on higher dimensional outputs and converges within a relatively low number of iterations. The behaviour of the function in two dimensions is shown in figure 1.

3.2 Rosenbrock's function

$$f(\vec{x}) = \sum_{i=1}^{N-1} 100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2$$

This function was proposed by Rosenbrock in [10] in a series of benchmark functions. The function is easy scalable to higher dimensions, it is continuous and multi-modal, so in general it has more than one global minimum. For $N = 3$, there is an analytical solution at $(1, 1, 1)$ [11]. The general solution for $N \geq 2$ is $x^* = (1, ..., 1)$ with $f(\vec{x}) = 0$. A plot of the function in two dimensions is shown in figure 2

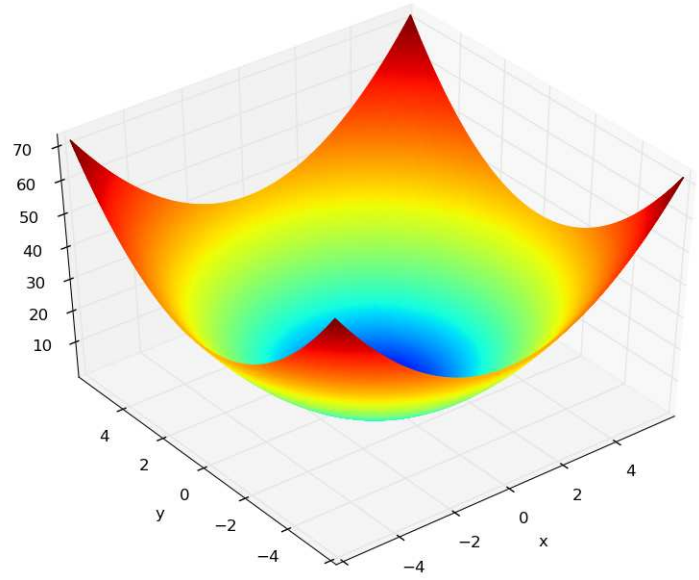

Figure 1: Plot of the sphere function on two dimensions.

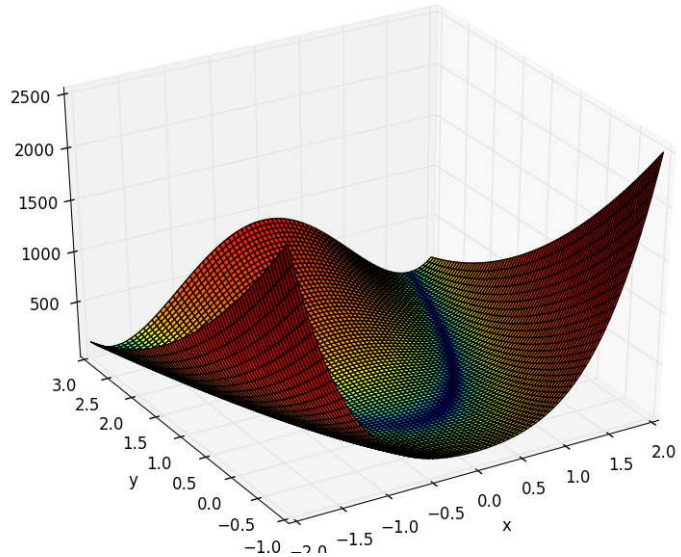

Figure 2: Plot of the Rosenbrock function on two dimensions.

4 Implementation

The implementations of the algorithms described in the previous section in Octave/ MATLAB code were kindly provided by Cristoph Foussette [12]. They provide a general

and easy customizable implementation of the (1+1)-ES, DES1-3 and the CORR-ES. The CORR-ES is a simpler version of the CMA-ES. The algorithm uses a Covariance Matrix, but is mainly focussed on rotation angle adaptation and doesn't use an evolution path as the CMA does. To do a large number of runs of all of these Evolution Strategies, a comparison function was build. All implementations are given in Appendix A.

4.1 Parameters

Each algorithm has quite a number of parameters. These have to be constant troughout the experiment, in order to be able to compare results. The most important ones are listed below.

The population sizes for parents and offspring, μ and λ, have a strong influence on the convergence speed of genetic algorithms. However, the derandomized evolution strategies are designed for small populations. The usual minimal configuration is one parent and an offpsring of size 10, so $(\mu, \lambda) = (1, 10)$. These settings are kept constant in the experiment. A small test on a $(4, 40)$ was done, but it provided only a small increase of the convergence speed, so the standard settings were kept. Of course these settings don't apply to the (1+1)-ES, as it has a $(1 + 1)$ setting by definition.

The target fitness value was set to $1E - 8$, forcing the ES to be really close to $\vec{0}$ for the sphere function, or $\vec{1}$ for Rosenbrock, before acceptance. This was kept constant for all Evolution Strategies and all dimensionalities. The algorithms demand an upper and lower bound for the candidate solutions. For both the sphere function and Rosenbrock's function, these were set to $[-100, 100]^n$ as suggested in [8]. The initial solution is a random vector $\vec{x} \in [-100, 100]^n$.

4.2 Environment

The experiments were executed on a large number of PCs in the LIACS computer rooms. In order to run the large number of executions as described in the next section, each pc ran a preset number of executions on one of the ES. Two types of PCs were involved, namely the quad-core Intel(R) Core(TM) i5 3.20GHz with 4GB memory, and the dual-core Intel(R) Core(TM)2 Duo 3.16GHz with 2GB memory. Processes were executed with a low priority, in order to prevent inconvenience for other users using the PCs.

4.3 Setup

The initial setup for the experiment was to run the five described Evolution Strategies on the sphere function and Rosenbrock's function on the range of dimensions from $n = 10$ to $n = 1000$, with stepsize $\Delta n = 10$. To compensate for the stochastic element in evolutionary algorithms, each step was run 10 times and averaged. However, Rosenbrock's function turned out to be a lot harder to solve for the ES. The number of function evaluations needed to solve Rosenbrock's function was several orders of magnitudes larger than the number needed for the sphere function on the same number of dimensions. For practical reasons, it was decided to change the range for Rosenbrock to $n = 2$ to $n = 100$,

with stepsize $\Delta n = 1$. Altough the range is different, the results should be relatively comparable, because both the range and the stepsize are one order of magnitude smaller. Note that $n = 2$ is the minimum dimension required for Rosenbrock's function, so $n = 1$ is excluded.

5 Results

In this section the results of the experiment are covered. It contains graphs of the results for both test functions, and the discussion of these results.

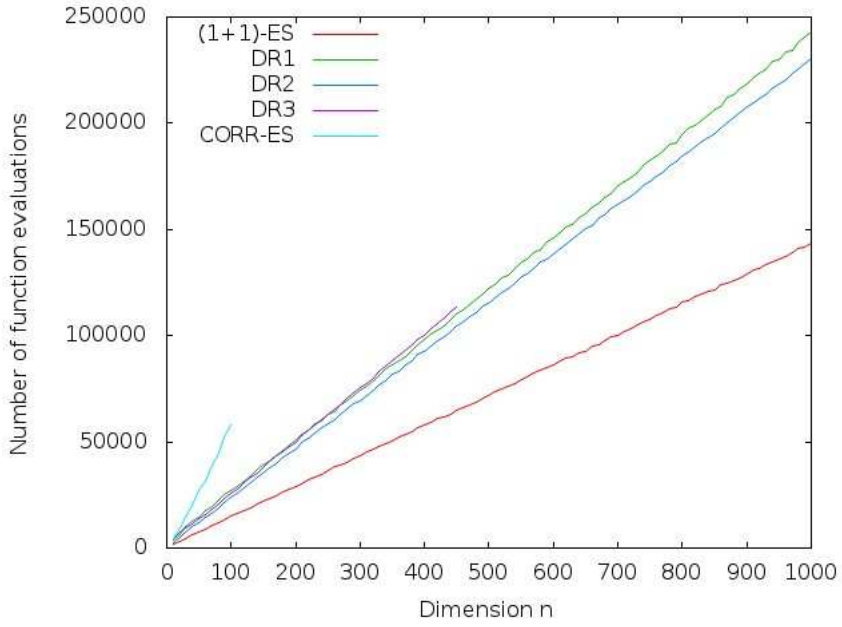

Figure 3: Plot of the experiment on the sphere function

5.1 Sphere function

The results of the experiment on the sphere function are shown in Figure 3. The relation between the number of function evaluations and the dimension of the objective function is clearly linear. The increase of the number of function evaluations per dimension is shown in Table 1. The (1+1)-ES, DR1 and DR2 are executed on the full range of $[10, 1000]$. The others however, are not. Derandimized ES 3 was run up to $n = 450$ because of time limitations. A single execution of DR3 with $n = 450$ in te described setup takes about 11 hours. The CORR-ES was only run up to $n = 100$ due to both time and space limitations. A run of the CORR-ES with $n = 100$ takes about 18 hours, but more importantly, it consumes more than the available 4GB memory on a higher dimensional sphere function, resulting in a crash of the process.

ES	#function evaluations
(1+1)-ES	142.954
DR1	239.726
DR2	229.315
DR3	248.748
CORR-ES	604.024

Table 1: Average increase of function evaluations per dimension on the sphere function

The results are overall linear as expected, but the relative differences between the ES are surprising at first hand. The (1+1)-ES is a lot faster than the other algorithms. This can be explained in relation to the complexity of the objective function. The sphere function is basically a large sink, so every mutation has quite a large chance to be in the right direction. The (1+1)-ES has one function evaluation in every iteration, where the other ES have 10 function evaluations. Apparently, the progress-per-function-evaluation rate is higher for the (1+1)-ES than for the more sophisticated evolution strategies, and this effect gets stronger on higher dimensions. Figure 4 shows the number of generations for each dimension and shows clearly that the (1+1)-ES is actually a lot slower from that point of view.

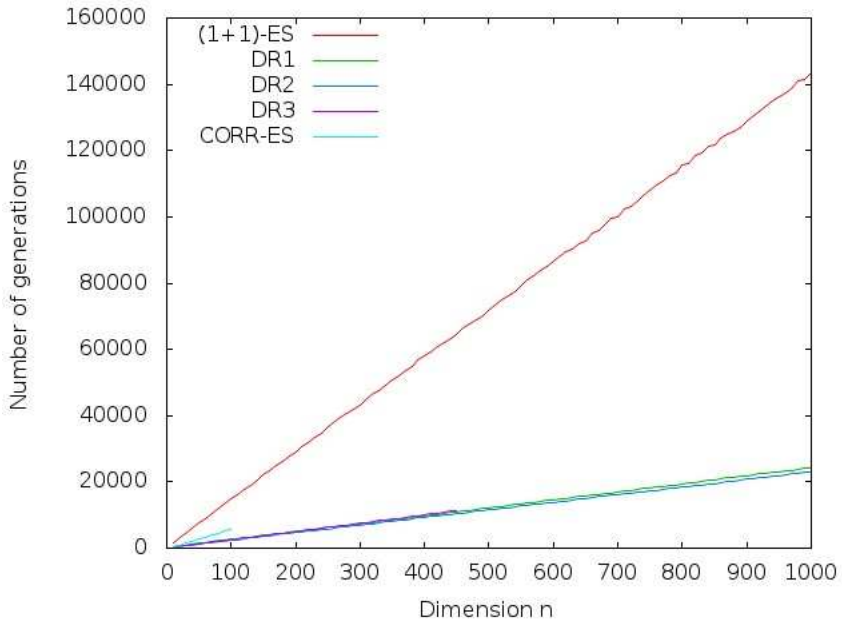

Figure 4: Plot of the experiment on the sphere function with the number of generations for each dimension

ES	#function evaluations	Polynomial fit
(1+1)-ES	69365.3	$f(x) = 425.716x^2 + 25986.3x + 440922$
DR1	10400.9	$f(x) = 77.8613x^2 + 2653.84x + 15754.6$
DR2	13651.5	- not enough results -
DR3	12532.0	$f(x) = 116.682x^2 + 693.22x + -1177.49$
CORR-ES	18921.5	$f(x) = -101.712x^2 + 23358.8x + 179.336$

Table 2: Average increase of function evaluations per dimension on Rosenbrock's function

Secondly, the number of function evaluations for the CORR-ES increases a lot faster than for the other evolution strategies. According to the theory, the CMA-ES should be at least as fast as the other ES and probably even faster. It turns out that the CORR-ES without an evolution path is a lot slower than the general CMA-ES. This result matches the literature, as the CORR-ES is proven to be a lot slower than the CMA [13].

5.2 Rosenbrock function

The results of the experiment on Rosenbrock's function are shown in Figure 5. The (1+1)-ES, DR1 and DR3 are run on the full range of $[2, 100]$. The CORR-ES is tested on the range of $[2, 42]$, due to time limitations, since one run with $n = 40$ takes about 11 hours. The DR2 algorithm has very limited results, on the range of $[2, 4]$. The algorithm doesn't converge when the dimension of Rosenbrock's function is 5 or higher. After investigation of this problem, no reason can be found. The implementation of DR2 seems to be correct, it gives good results on the sphere function, and the implementation of Rosenbrock's function gives good results on the other implemented ES. The average increase of the number of function evaluations every time the dimension increases one step is given in Table 2. The results for the (1+1)-ES, DR1 and DR3 are not fully linear, so a polynomial curve fit is provided in the same table.

The results are clearly different from the results of the experiment on the sphere funtion. The plot shows the same linearity with respect to the dimensionality of the function, but the relative speed of the evolution strategies is totally different. On the sphere function, the (1+1)-ES was the fastest algorithm, but on a harder problem, the more robust and sophisticated algorithms perform a lot better. As expected, DR1 and DR3 have a good performance, DR3 being a bit better than DR1. The CORR-ES however, performs better than the (1+1)-ES, but is slower than the Derandomized Evolution Strategies. The reason for it, is the same as for the experiment on the sphere function. The CORR-ES converges slower than a general CMA-ES, and the complexity of the objective function is not a large factor in this.

Altough the results are overall linear, there are a lot of fluctuation in the results, especially in the results of the (1+1)-ES. This has to do with the stochastic element in Evolution Strategies. Because of that, some instances of an algorithm might be a lot faster than others. To compensate this effect, each setup was run 10 times, and

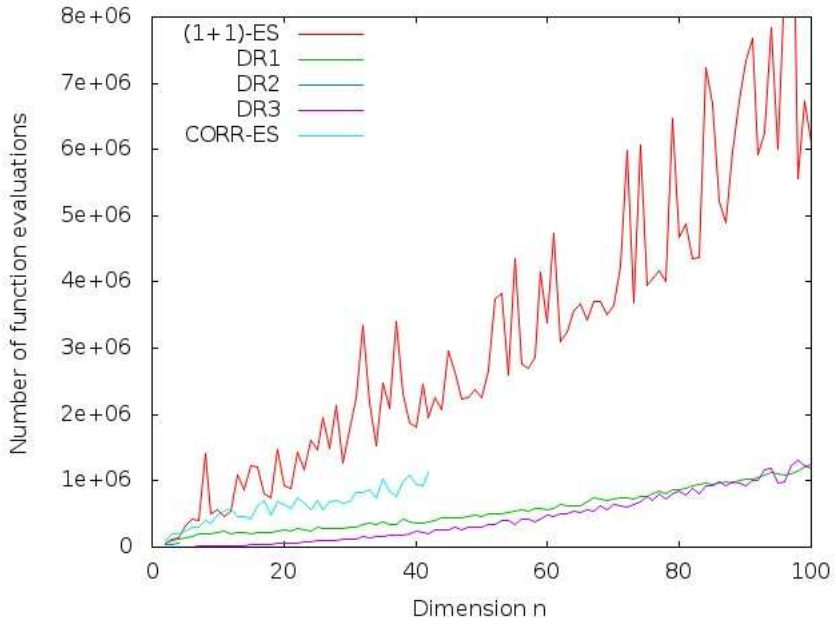

Figure 5: Plot of the experiment on Rosenbrock's function

the average was used as result. It seems that 10 runs is not enough to cancel out the randomness for each algorithm. Especially the algorithm that relies heavy on random steps, the (1+1)-ES, shows very large fluctuations of the results. The average on 20 or 30 runs might compare for it, but due to time limitations, and because the overall linear tendency is clearly visible, no more experiments were done. Figure 6 shows the results of the (1+1)-ES without averaging. The trend of the individual results is clearly visible.

6 Conclusion

Altough the results of the experiments are not as complete as expected, mostly because of the strongly increasing execution time, the obtained result is complete enough to draw some conclusions.

The differences between the algorithms on one objective function are as expected. Especially on Rosenbrock's function, the more sophisticated algorithms win over the simpler and more stochastic (1+1) algorithm. On the sphere function, this effect is not as strong, as the (1+1)-ES is faster than the others in terms of function evaluations.

The difference in results between the two objective functions are quite large. For all algorithms, Rosenbrock's function is a lot harder to solve than the sphere function. Even for low-dimensional experiments it takes several orders of magnitude more function evaluations to converge. Besides that, the variation of the results is a lot higher on

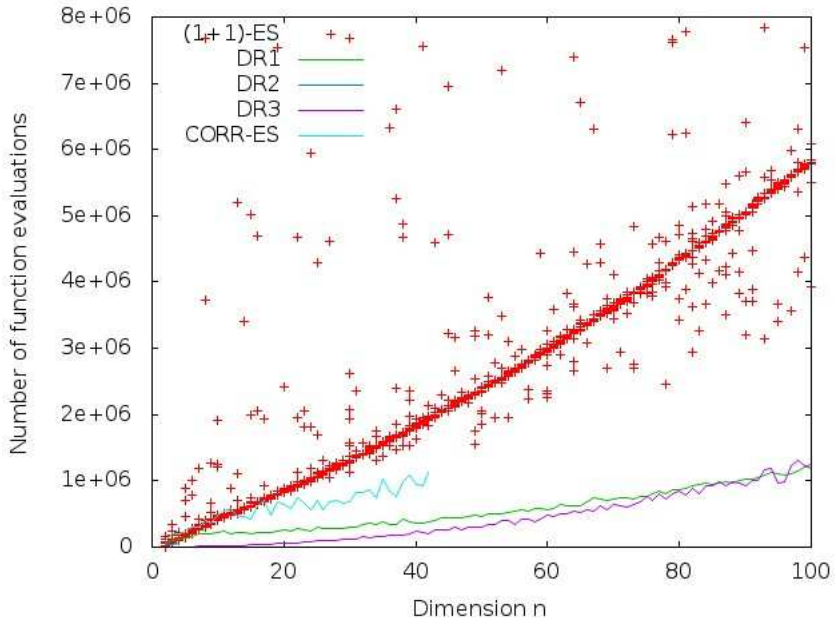

Figure 6: Plot of the experiment on Rosenbrock's function with unique results on (1+1)-ES

Rosenbrock's function than on the sphere function, which is presumably the direct effect of the multimodality of Rosenbrock's function.

The effect of dimensionality on the algorithms in general is different for the two objective functions. For the sphere function, the plots for all algorithms are very linear, but the effect of increasing dimensionality is the least on the (1+1)-ES, so for very simple high dimensional problems, this algorithm is preferrable. For Rosenbrock's function, the picture is less clear. The linear average increase of function evaluations is the lowest for DR1, but the results don't definitively answer whether the increase is linear or not. The polynomial fits imply that the (1+1)-ES, DR1 and DR3 are polynomially increasing. The CORR-ES however, increases slower on higher dimensions in this fit, but this can be a result of the variation in the results for Rosenbrock's function and the smaller range on which the CORR-ES is evaluated.

6.1 Future research

To be able to make a good comparison, it would be nice if more of this systematical research was done on other objective functions and on other algorithms. Especially the CMA-ES would be interesting, as the simplified CORR-ES already shows some potential, but is overall slower than the Derandomized Evolution Strategies.

As a remark on the comparison of Evolution Strategies in general, the choice for objective function evaluations as comparison parameter might be debatable in some

cases. A simple Evolution Strategy such as the (1+1)-ES needs a lot more function evaluations to converge, but in some cases it still converges faster than a CORR-ES or a Derandomized ES when it comes to execution time. Of course this has to do with the complexity of the algorithm. A (1+1)-ES only generates a single new solution, where a CORR-ES or CMA-ES evaluates and updates a full covariance matrix for each generation. This is obviously a lot more time consuming.

References

[1] F. van den Bergh, A.P. Engelbrecht, "A cooperative approach to particle swarm optimization", in *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[2] Ofer M. Shir, "Niching in Derandomized Evolution Strategies and its Applications in Quantum Control; A Journey from Organic Diversity to Conceptual Quantum Designs", PhD Thesis Universiteit Leiden. ISBN: 978-90-6464-256-2, 2008.

[3] A. Ostermeier, A. Gawelczyk, N. Hansen, "A Derandomized Approach to Self Adaptation of Evolution Strategies", TU Berlin, Tech. Rep. TR-93-003, 1993.

[4] A. Ostermeier, A. Gawelczyk, N. Hansen, "Step-Size Adaptation Based on Non-Local Use of Selection Information," in *Parallel Problem Solving from Nature - PPSN III*, ser. Lecture Notes in Computer Science, vol. 866, pp. 189-198, Springer, 1994,

[5] N. Hansen, A. Ostermeier, and A. Gawelczyk, "On the Adaptation of Arbitrary Normal Mutation Distributions in Evolution Strategies: The Generating Set Adaptation," in *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6)*. San Francisco, CA: Morgan Kaufmann, pp. 57-64, 1995.

[6] N. Hansen and A. Ostermeier, "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: the Covariance Matrix Adaptation," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. Piscataway, NJ: IEEE, pp. 312-317, 1996.

[7] N. Hansen. *The CMA Evolution Strategy: A Tutorial.* 2011.

[8] K. Tang, Xiaodong Li, P. N. Suganthan, Z. Yang and T. Weise, "Benchmark Functions for the CEC'2010 Special Session and Competition on Large Scale Global Optimization," Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, http://nical.ustc.edu.cn/cec10ss.php, 2009.

[9] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems", PhD thesis, University of Michigan, 1975.

[10] H. H. Rosenbrock, "An Automatic Method for Finding the Greatest or Least Value of a Function", in *The Computer Journal*, vol. 3, no. 3, pp. 175-184, 1960.

[11] S. Kok, C. Sandrock, "Locating and Characterizing the Stationary Points of the Extended Rosenbrock Function", in *Evolutionary Computation*, vol. 17, no. 3, pp. 437-453, 2009.

[12] Christophe Foussette (personal correspondence)

[13] N. Hansen, "Invariance, Self-Adaptation and Correlated Mutations in Evolution Strategies," in *Parallel Problem Solving from Nature PPSN VI*, pp. 355-364, 2000.

Appendices

A Source code

Main comparison function

```
1   function [] = compareFunctions(ESname, funName, runs, nStart, nEnd, nStep, maxFunEvals, mu,
        lambda)
2   % Input parameters
3   % - ESname               name of ES to run
4   % - funName              name of objective function
5   % - runs                 number of runs for each n (to get mean value)
6   % - nStart               start dimension
7   % - nEnd                 end dimension
8   % - nStep                increase of dimension in each step
9   % - maxFunEvals maximum number of function evaluations to be executed
10  % - mu, lambda  parent and offspring population size
11  %
12  % Author: Piet van Hekke
13
14          if (isdeployed)
15                  runs = str2num(runs);
16                  nStart = str2num(nStart);
17                  nEnd = str2num(nEnd);
18                  nStep = str2num(nStep);
19                  maxFunEvals = str2num(maxFunEvals);
20                  mu = str2num(mu);
21                  lambda = str2num(lambda);
22          end
23
24          matlabpool open
25          results_dir = 'Results/';
26
27          if (exist([results_dir]) == 0) mkdir([results_dir]); end
28          resultsfile = fopen([results_dir,ESname,funName,'_',num2str(nStart),'_',num2str(nEnd),'
                _',num2str(nStep)],'w');
29          resultsmean = fopen([results_dir,'M_',ESname,funName,'_',num2str(nStart),'_',num2str(
                nEnd),'_',num2str(nStep),'runs',num2str(runs)],'w');
30
31          optsIn = struct();
32          optsIn.mu = mu;
33          optsIn.lambda = lambda;
34          % Specify upper and lower bound for objective function
35          if strcmp(funName,'sphereFunction')
36                  ub = 100;
37                  lb = -100;
38          elseif strcmp(funName,'rosenbrock')
39                  ub = 100;
40                  lb = -100;
41          elseif strcmp(funName,'griewank')
42                  ub = 600;
43                  lb = -600;
44          elseif strcmp(funName,'bohachevsky')
45                  ub = 15;
46                  lb = -15;
47          end
48
49          i = nStart;
50          tempruns = runs;
51          extraruns = 0;
52          hist_best_so_far = NaN(runs,1);
53          timeElapsed = NaN(runs,1);
```

```
54          while(i <= nEnd)
55                  if tempruns ~= runs || extraruns > 0
56                          disp(['--Doing rerun of length ',num2str(tempruns), ' now, ',num2str(
                                extraruns-tempruns),' reruns done.'])
57                  else
58                          disp(['--- n = ',num2str(i),' ---'])
59                  end
60
61                  if tempruns == runs
62                          hist_best_so_far = NaN(tempruns,1);
63                          timeElapsed = NaN(tempruns,1);
64                  end
65                  optsIn.ub = ub.*ones(i,1);
66                  optsIn.lb = lb.*ones(i,1);
67                  optsIn.maxFunEvals = i*maxFunEvals;
68
69                  %Run specified ES for 'runs' runs
70                  parfor j = 1:tempruns
71                          t = fix(clock);
72                          disp([ESname,' run ',num2str(j),'/',num2str(tempruns),' dim ',num2str(i
                                ),'(max ', num2str(nEnd),') - ', num2str(t(2)),'-',num2str(t(3)),'
                                ', num2str(t(4)),':',num2str(t(5)),':',num2str(t(6))])
73                          randNr = lb + (ub-lb).*rand(1,i);
74                          clear functions;
75                          tic;
76                          [xopt, fopt, funevals, resulthistf] = feval(ESname,funName,i,randNr,
                                optsIn);
77                          timeElapsed(j) = toc;
78                          t = fix(clock);
79                          if funevals >= optsIn.maxFunEvals
80                                  disp(['Results (',num2str(fopt),') of ',num2str(j),': ',
                                        num2str(funevals), ' in ', num2str(timeElapsed(j)),' - ',
                                        num2str(t(2)),'-',num2str(t(3)),' ',num2str(t(4)),':',
                                        num2str(t(5)),':',num2str(t(6)),' - Fail!!'])
81                          elseif fopt <= 1e-8
82                                  disp(['Results (',num2str(fopt),') of ',num2str(j),': ',
                                        num2str(funevals), ' in ', num2str(timeElapsed(j)),' - ',
                                        num2str(t(2)),'-',num2str(t(3)),' ', num2str(t(4)),':',
                                        num2str(t(5)),':',num2str(t(6))])
83                          else
84                                  disp(['Results (',num2str(fopt),') of ',num2str(j),': ',
                                        num2str(funevals), ' in ', num2str(timeElapsed(j)),' - ',
                                        num2str(t(2)),'-',num2str(t(3)),' ', num2str(t(4)),':',
                                        num2str(t(5)),':',num2str(t(6)),' - Fail!! - In time, but
                                        no solution'])
85                                  funevals = optsIn.maxFunEvals+1;
86                          end
87                          hist_best_so_far(j) = funevals;
88                  end
89                  tempruns = runs;
90                  rerun = 0;
91                  for j = 1:runs
92                          if hist_best_so_far(j) >= optsIn.maxFunEvals
93                                  rerun = rerun+1;
94                          end
95                  end
96                  if rerun == 0
97                          for j = 1:runs
98                                  fprintf(resultsfile, '%d %d %d\n', i, hist_best_so_far(j),
                                        timeElapsed(j));
99                          end
100                         opl = mean2(hist_best_so_far);
101                         fprintf(resultsmean, '%d %d %d\n', i,opl,extraruns);
102                         extraruns = 0;
103                         i = i + nStep;
104                 else
105                         tempruns = rerun;
106                         extraruns = extraruns + tempruns;
107                         for j = 1:tempruns
108                                 % Empty first places in results-array for rerun
109                                 tempRes = hist_best_so_far(j);
110                                 tempTime = timeElapsed(j);
111                                 done = false;
112                                 k = runs;
113                                 while ~done
114                                         if hist_best_so_far(k) >= optsIn.maxFunEvals
115                                                 hist_best_so_far(k) = tempRes;
116                                                 timeElapsed(k) = tempTime;
117                                                 done = true;
118                                         end
119                                         k = k-1;
120                                 end
121                         end
```

```
122                     end
123             end
124
125     fclose(resultsfile);
126     fclose(resultsmean);
127     close all;
128     matlabpool close
129     end
```

(1+1)-ES

Not published online, because I am not the owner of this implementation.

DR1

Idem

DR2

Idem

DR3

Idem

CORR-ES

Idem

Parameter function

Idem

Boundary function

Idem

Objective functions

Sphere function - Idem

```
1   function f=rosenbrock(x)
2       %if size(x,1) < 2 error('dimension must be greater one'); end
3       f = 100*sum((x(1:end-1).^2 - x(2:end)).^2) + sum((x(1:end-1)-1).^2);
4   end
```