# Universiteit Leiden
# Opleiding Informatica

Evolving the Structure of Evolution Strategies using a Genetic Algorithm


Name: Sander van Rijn

Date: 19/02/2016

1st supervisor: Prof. dr. Thomas Bäck
2nd supervisor: Dr. Michael Emmerich


MASTER'S THESIS


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

LEIDEN UNIVERSITY

# Evolving the Structure of Evolution Strategies using a Genetic Algorithm

Sander van Rijn

M.Sc. Thesis

Natural Computing Group
Leiden Institute of Advanced Computer Science (LIACS)

February 2016

# *Abstract*

Evolution strategies are one of the most successful classes of stochastic optimization algorithms for solving real world problems, which involves discontinuous, discrete or mixed-integer search space with nonlinear constraints. Since the creation of evolution strategies back in the 1960s, a variety of improvements and modifications have been suggested to enhance its performance.

The covariance matrix adaptation evolution strategy (CMA-ES) is the state-of-the-art development in this category. Many variants have been proposed recently to accelerate its convergence speed. However, how to choose those variants optimally in practice is still an open question. In this thesis, based on the well-known No Free Lunch Theorem, we state that the optimal choice of variants should be related to the type of objective function landscape. For example, a certain variant favoring uni-modal landscape would perform worse on a multi-modal landscape.

In order to obtain the optimal variant setting in practice, it is proposed to consider all the variants as a search space such that 1) many new combinations of variants beyond the literature can be tested and 2) an optimization algorithm can be used to search for the optimal ES-structure. An ES framework is presented in this thesis, which allows the usage of all the possible combinations of ES-variations. In addition, a genetic algorithm (GA) is exploited to evolve the ES-structure for a given black-box optimization problem using this framework. An empirical study is also conducted to validate the proposed approach, in which the GA is shown to converge fast and consistently to the best possible ES-structure within the framework by comparison with a brute force search. Performance of the evolved ES-structures is finally compared to the results of the black-box optimization benchmark (BBOB) from 2009 by means of the fixed cost error measure.

Parts of this thesis are extracted from a paper by Van Rijn *et al.* [21].

# *Acknowledgements*

The author would like to extend special thanks to Hao Wang and Thomas Bäck for their scientific guidance during the research for this thesis.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Evolution Strategies are popular for solving a large variety of real-valued optimization problems. Many adaptations of the classic *evolution strategy* (ES) have been proposed and tested. Most adaptations still follow the same general structure of an evolution strategy (see Algorithm 1). None of these can be perfect for all fitness landscapes because of the No Free Lunch theorem [23]: An increase in performance for some cases causes a decrease in performance for some other cases. Instead, a realistic goal is to have an ES that finds a good solution in the least amount of function evaluations for a specific function or function class.

An all-round optimization method such as the *covariance matrix adaptation* ES (CMA-ES) by Hansen *et al.* [14] can be immediately used for new problems with decent expected results. This comes at the cost of more function evaluations to reach a similar solution, when compared to an optimization method tailored to the problem. Choosing an ES that is tailored to the problem results in faster convergence, but analysis of the fitness landscape is often required before such a choice can be made. For many black-box optimization problems, this is not feasible. Automated parameter tuning can provide some improvement in these cases, but is limited in effectiveness.

**Algorithm 1** Outline of a general $(\mu, \lambda)$ evolution strategy

1: **Initialization** of $\mu$ parents for the first generation
2: **repeat**
3: The $\mu$ parents **recombine** their information to create $\lambda$ offspring
4: Each offspring obtains random **mutations**
5: All individuals are **evaluated**
6: The $\mu$ best (fittest) individuals are **selected** as parents for the next generation
7: **until** Termination criterion fulfilled

Instead, we view the task of determining the *structure* of the fastest converging method as an optimization problem itself. We consider methods based on the CMA-ES, as they are the focus of many recent publications. A number of CMA-ES variants recently described in literature including *active update* [15], *increasing population* (IPOP) [2] and several others, are described in [5].

This research aims at finding an ES whose structure is determined by combining one or more of these variants into the CMA-ES. A slightly similar approach has been taken by Bäck [4], by using an evolutionary algorithm to optimize parameters and the operator for selection and crossover of a simple genetic algorithm (GA). In more recent research, Martin *et al.* [18] used Genetic Programming to create tree-based structures of GAs. We choose not to use a tree-based structure, because more components in which the ES can vary are examined other than selection or recombination, and any parameters values are excluded in our optimization. This results in a representation by means of a list of integers, so a GA is chosen here for the optimization.

A complete definition of the problem can be found in chapter 2. Our approach is explained in more detail in chapter 3. Section 3.2 lists all used variants of the CMA-ES, followed in Section 3.3 by the problems given by some combinations and our solutions for them. We give an overview of the ES-framework created and the GA that uses it in Section 3.4 and Section 3.6, respectively. The experimental setup is discussed in chapter 4. The results of these experiments are listed in chapter 5. Finally, conclusions from these results and suggestions for future work are discussed in chapter 6.

# Chapter 2

# Problem Definition

We examine the set of real-valued minimization problems $\mathbb{F} = \{f : \mathbb{R}^n \to \mathbb{R}\}$ in $n$ dimensions. The goal of an optimization method is to find some $\mathbf{x}_{\text{opt}} \in \mathbb{R}^n$ such that

$$\forall \mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}_{\text{opt}}) \leq f(\mathbf{x}).$$

Although optimization methods have the *goal* of finding the *true optimum* $f(\mathbf{x}_{\text{opt}})$, it can only be approximated. Assuming that all algorithms can reach a certain threshold value $f' > f(\mathbf{x}_{\text{opt}})$, the key difference between methods is the number of function evaluations $f(\mathbf{x})$ they require to reach that threshold. This is the idea behind the *expected runtime* (ERT) measure for comparing optimization methods with respect to a fixed (black-box) optimization problem: Set a target threshold $f'$ and run each method until this target has been reached. The method that uses the least function evaluations to reach the target must have *converged* faster, and is therefore better suited to that particular fitness landscape. The aim of this research is to find, for a given optimization problem, the ES-structure that achieves the *highest convergence speed*.

A threshold value $f'$ must be reached before a comparison can be made when using the ERT measure for comparing convergence speed of different optimization methods. An upper limit of the allowed number of function evaluations ensures that optimization methods which are highly unsuited for the given optimization problem will terminate.

The difference in required number of function evaluations will be very large in these cases. However, an indication of convergence speed can already be obtained before the threshold is reached. When one method converges faster than another, there will be a difference in the lowest value found after both methods have been allowed the same number of function evaluations. Comparing the best found values within a relatively low

and fixed evaluation budget, also known as the *fixed-cost error* (FCE) measure, can be used to approximate the convergence speed of an optimization method to some extent.

The major advantage of using the FCE measure is that the less runtime is required before a comparison can be made. A drawback of using FCE is that information on the performace of the optimization method in a later stadium is lost. There may exist methods that initially converge slower, but speed up at a later point in the optimization process. Such methods will be at a disadvantage when the allowed budget does not reach into the second stage of their optimization progress. It is impossible to detect when such a case occurs without allowing more function evaluations.

# Chapter 3

# Approach

The CMA-ES [14] and its variants are the preferred optimization method for many real-valued black-box problems. All these variants form a search space. Therefore, it is worth conducting structural optimization for CMA-ES. Many of these variations have been separately discussed in the literature, but only few combinations have been tested. This leaves a large part of the potential search space for CMA-ES-like optimization methods undiscovered. We search this space using a Genetic Algorithm (GA).

In Section 3.1 the default CMA-ES used in this research is defined. Section 3.2 lists and discusses all variants of the CMA-ES included in our experiments. Most combinations of these variants work without any major problems, but some require extra attention in order to work. The problems encountered and our solutions are discussed in Section 3.3. An overview of the ES-framework we implemented, and the representation that is used to define an ES-structure in this framework are given in Sections 3.4 and 3.5 respectively. Finally, the GA used for the optimization process is discussed in Section 3.6.

## 3.1 CMA-ES

The presented framework is built on a default $(\mu_W, \lambda)$-CMA-ES from [5] (see Algorithm 2). This implementation follows the general structure of an ES as outlined in Algorithm 1.

Lines 1 – 5 are the initialization phase of the algorithm. The initial parent $\langle \mathbf{x} \rangle$ is determined in line 1, while the evolution paths $\mathbf{p}_c$ and $\mathbf{p}_\sigma$ are initialized as zero vectors $\mathbf{0}^n$ in lines 2 and 3. The covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ is initially set to the identity matrix $\mathbf{I}^{n \times n}$ (line 4), and the generation counter is started at 0 (line 5).

Mutation is performed in lines 8 – 12. First the eigendecomposition of $\mathbf{C}$ is stored in $\mathbf{B}$ and $\mathbf{D}$ (line 8). For all $\lambda$ individuals, a new random vector $\mathbf{z}_i$ is drawn from the normal distribution (line 10), and is then used to create a new mutation vector $\mathbf{y}_i$ that is adapted according to the covariance matrix by multiplication with $\mathbf{BD}$ (line 11). Addition of the parent $\langle \mathbf{x} \rangle$ to the mutation vector $\mathbf{y}_i$ scaled by the step-size $\sigma$ results in the new individual $\mathbf{x}_i$ (line 12). This individual is then evaluated in line 13.

Selection is implicit before line 15, as the mutation vectors $\mathbf{y}_i$ are sorted according to fitness value $f_i$ of the corresponding individual $\mathbf{x}_i$. Finally, weighted recombination of the mutation vectors is performed in line 15, where the weights $w_i$ are defined by

$$w_i = \log(\mu + \frac{1}{2}) - \frac{\log(i)}{\sum_j w_j}. \tag{3.1}$$

This combined mutation vector $\langle \mathbf{y} \rangle$ is then used to update the parent $\langle \mathbf{x} \rangle$ in preparation for the next generation.

The remaining lines 17 – 21 describe the adaptation of the strategy parameters. Evolution paths $\mathbf{p}_c$ and $\mathbf{p}_\sigma$ are first updated (lines 17 – 18) using the recombined mutation vector. An additional parameter $h_\sigma$ prevents the addition of information from this generation when $\|\mathbf{p}_\sigma\|$ becomes too large. It is defined as

$$h_\sigma = \begin{cases} 1 & \text{if } \frac{\|\mathbf{p}_\sigma\|}{\sqrt{1-(1-c_\sigma)^{2(t+1)}}} < (\frac{7}{5} + \frac{2}{n+1}) E \|N(\mathbf{0}, \mathbf{I})\| \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where

$$E \|N(\mathbf{0}, \mathbf{I})\| \approx \sqrt{n} \left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right).$$

The step-size $\sigma$ is updated next, using the information of the evolution path $\mathbf{p}_\sigma$. Finally, the covariance matrix $\mathbf{C}$ is updated in lines 20–21. The first term in line 21 denotes the contribution of the current covariance matrix. Next is the *rank-one-update*, which uses the information from the evolution path $\mathbf{p}_c$. The *rank-$\mu$-update* applies $\mathbf{Z}$, which contains information of the most successful mutations.

Default settings for the exogeneous parameters are used from [10]:

$$\begin{aligned} \lambda &= 4 + \lfloor 3 \ln n \rfloor \\ \mu &= \left\lfloor \frac{\lambda}{2} \right\rfloor \\ \mu_{\text{eff}} &= \left( \sum_{i=1}^{\mu} w_i^2 \right)^{-1} \end{aligned}$$

$$
\begin{aligned}
c_\sigma &= \frac{\mu_{\text{eff}} + 2}{\mu_{\text{eff}} + n + 5} \\
d_\sigma &= 1 + 2\max(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{n+1}}) \\
c_c &= \frac{4 + \mu_{\text{eff}}/n}{n + 4 + 2\mu_{\text{eff}}/n} \\
c_1 &= \frac{2}{(n+1.3)^2 + \mu_{\text{eff}}} \\
c_\mu &= \min\left(1 - c_1, \alpha_\mu \frac{\mu_{\text{eff}} - 2 + 1/\mu_{\text{eff}}}{(n+2)^2 + \alpha_\mu \mu_{\text{eff}}/2}\right) \text{ with } \alpha_\mu = 2
\end{aligned}
$$

**Algorithm 2** Default $(\mu_W, \lambda)$-CMA-ES

initialize $\langle \mathbf{x} \rangle$
$\mathbf{p}_c \leftarrow \mathbf{0}$
$\mathbf{p}_\sigma \leftarrow \mathbf{0}$
$\mathbf{C} \leftarrow \mathbf{I}$
$t \leftarrow 0$
**repeat**
  $t \leftarrow t + 1$
  $\mathbf{B}, \mathbf{D} \leftarrow$ eigendecomposition of $\mathbf{C}$
  **for** $i = 1$ to $\lambda$ **do**
    $\mathbf{z}_i \leftarrow N(\mathbf{0}, \mathbf{I})$
    $\mathbf{y}_i \leftarrow \mathbf{BDz}_i$
    $\mathbf{x}_i \leftarrow \langle \mathbf{x} \rangle + \sigma \mathbf{y}_i$
    $f_i \leftarrow f(\mathbf{x}_i)$
  **end for**
  $\langle \mathbf{y} \rangle \leftarrow \sum_{i=1}^{\mu} \mathbf{y}_{i:\lambda} w_i$
  $\langle \mathbf{x} \rangle \leftarrow \langle \mathbf{x} \rangle + \sigma \langle \mathbf{y} \rangle$
  $\mathbf{p}_\sigma \leftarrow (1 - c_s)\mathbf{p}_\sigma + \sqrt{c_s(2 - c_s)\mu_{\text{eff}}}\mathbf{BD}^{-1}\mathbf{B}^T \langle \mathbf{y} \rangle$
  $\mathbf{p}_c \leftarrow (1 - c_c)\mathbf{p}_c + h_\sigma \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \langle \mathbf{y} \rangle$
  $\sigma \leftarrow \sigma \cdot \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{E\|N(\mathbf{0},\mathbf{I})\|} - 1\right)\right)$
  $\mathbf{Z} \leftarrow \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}^T$
  $\mathbf{C} \leftarrow (1 - c_1 - c_\mu)\mathbf{C} + c_1(\mathbf{p}_c, \mathbf{p}_c^T + (1 - h_\sigma)c_c(2 - c_c)\mathbf{C}) + c_\mu \mathbf{Z}$
**until** termination criterion fulfilled

## 3.2 ES Variations

Every ES we consider can be described as a sequence of discrete building blocks, each originally introduced as a separate ES-variant. Eleven variants are considered in total, nine of which have two available choices, and the remaining two have three choices. For each variant, a brief description is given below. This results in a search space of $2^9 \cdot 3^2 = \mathbf{4,608}$ different ES-structures.

### 3.2.1 Active Update

The update of covariance matrix $C$ is normally only done by taking the most successful mutations into account. With Active Update as introduced by Jastrebski *et al.* [15], an additional negative factor based on the $\mu$ least successful individuals is added. This is done by replacing line 20 of Algorithm 2 with

$$\mathbf{Z} = \sum_{k=1}^{\mu} w_i \mathbf{y}_{k:\lambda} \mathbf{y}_{k:\lambda}^T - \sum_{k=\lambda-\mu+1}^{\lambda} w_i \mathbf{y}_{k:\lambda} \mathbf{y}_{k:\lambda}^T. \tag{3.3}$$

Following [5], the parameter $c_c$ is modified to

$$c_c = \frac{2}{(n+\sqrt{2})^2}. \tag{3.4}$$

When $\lambda < 2\mu$, there will be an overlap between the $\mu$ most and least successful individuals. The mutations of these overlapping individuals will cancel out in Eq. 3.3, effectively reducing the number of individuals that affect the covariance matrix $\mathbf{C}$. Because of this, we only allow the application of this variant when $\lambda \geq 2\mu$.

### 3.2.2 Elitism

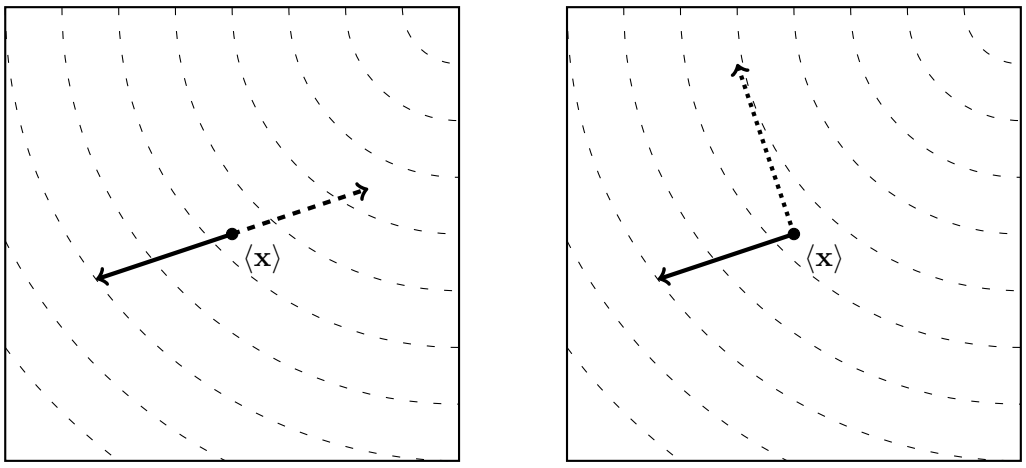
CMA-ES uses a $(\mu, \lambda)$-strategy by default, whereby the selection of individuals is only done using the current generation of offspring. When elitism is active, the parent individuals are also considered during selection. This is denoted as a $(\mu + \lambda)$-strategy.

### 3.2.3 Mirrored Sampling

A technique to ensure more evenly spaced sampling of the search space is Mirrored Sampling by Auger *et al.* [8]. Half of the mutation vectors are still sampled from the normal distribution, but every other mutation vector is the mirror image of the previous random vector. An example of this can be seen in Fig. 3.1a.

$$\mathbf{z}_i \leftarrow \begin{cases} N(\mathbf{0}, \mathbf{I}) & i \text{ is odd} \\ -\mathbf{z}_{i-1} & i \text{ is even} \end{cases} \tag{3.5}$$

Mirroring the previously generated random vector may even cross generations when $\lambda$ is odd. In such cases $\mathbf{z}_1^{t+1} \leftarrow -\mathbf{z}_\lambda^t$.




(A) Mirrored Sampling (B) Orthogonal Sampling

FIGURE 3.1: Examples of Mirrored (left) and Orthogonal (right) sampling. The dashed arcs represent the fitness landscape of the sphere function. Mutation vectors are represented by arrows originating from the parent $\langle \mathbf{x} \rangle$. The solid line indicates the original random mutation, while the dashed and dotted lines represent the mirrored and orthogonally sampled mutation vectors respectively. Note that the mirrored vector's length is equal to that of the original, while the orthogonal vector can be of different length.

### 3.2.4 Orthogonal Sampling

Wang *et al.* [22] later proposed Mirrored Orthogonal Sampling as an improvement to Mirrored Sampling in an attempt to further ensure more evenly spaced sampling by using mutation vectors that are orthogonal to each other. In this research, Orthogonal Sampling is considered separately from Mirrored Sampling. However, when both are active simultaneously, the initial mutation vectors are sampled using this method and mirrored afterwards.

For Orthogonal Sampling, the desired number of samples $\lambda'$ is first drawn from the normal distribution. The Gram-Schmidt process [6] is used to orthonormalize the set of vectors. If this number $\lambda'$ is greater than the dimensionality $n$ of the problem, only $n$ vectors are orthonormalized because only $n$ orthogonal vectors can exist in $n$-dimensional space. The remaining vectors sampled from the normal distribution are maintained as they are. Finally, the now orthonormalized vectors are restored to their original lenghts. Fig. 3.1b shows a possible result from this procedure.

### 3.2.5 Sequential Selection

The runtime duration of optimization by an ES depends on the function evaluation budget that is set beforehand. Every individual is evaluated in order before selection is performed using all $\lambda$ individuals. Not all of the $\lambda$ individuals will be an improvement

over the best result found up to that point in time, resulting in unnecessary function evaluations and therefore lost optimization time.

Sequential Selection, proposed by Auger *et al.* [8] for $(1\dagger,\lambda)$-selection, compares the function value of each individual to the best found so far immediately after it has been evaluated. The current generation is cut off to prevent further evaluations from occurring when an improvement has been found. The resulting evolution process is able to find more improving mutations because more generations are created.

For this method to work with $(\mu, \lambda)$-strategies, the required number of individuals for other calculations in the algorithm has to be taken into account. Allowing a cut-off after less than $\mu$ individuals causes issues with the recombination and update of the covariance matrix. A delay in the cut-off is introduced to ensure that at least $\mu$ individuals are evaluated when $\mu > 1$, preventing these issues. This represents a more robust solution than accepting less than $\mu$ individuals and adapting all following calculations.

### 3.2.6 Threshold Convergence

Getting stuck in a local optimum is a common issue when using an ES for solving multi-modal optimization problems. Piad *et al.* propose Threshold Convergence [19] for the standard $(\mu, \lambda)$-ES as a method of forcing the evolution to stay in an exploratory phase for longer, by imposing a minimum length threshold $T$ for mutation vectors. This threshold decays during the optimization process to transition the search from a global to local search.

A mutation vector $\mathbf{z}$ with length $\|\mathbf{z}\| < T$ is mirrored with respect to $T$ by

$$\mathbf{z} \leftarrow \mathbf{z} + 2 \cdot (T - \|\mathbf{z}\|) \cdot \mathbf{z}. \tag{3.6}$$

This threshold is calculated using

$$T_i = \alpha_{TC} \cdot d \cdot \left(\frac{n-i}{n}\right)^{\gamma}, \tag{3.7}$$

where $\alpha$ is the initial threshold, $d$ is the diagonal of the search space, $n$ is the total evaluation budget, $i$ is the number of evaluations used so far and parameter $\gamma$ controls the decay rate. Initial values of $\alpha_{TC} = 0.2$, $d = \sqrt{10^2 n}$ and $\gamma = 0.995$ are chosen, assuming a standard search space of $[-5, 5]^n$

In the original paper by Piad *et al.*, Threshold Convergence is used in a regular $(\mu, \lambda)$-ES. The threshold is applied to the mutation vector after it has been scaled by the step size $\sigma$. If this method is equally applied to a CMA-ES, any shape implied by the

covariance matrix $C$ is lost, because the threshold is a scaled (hyperdimensional) unit circle. Instead, the threshold is applied to the randomly sampled vector from line 10 in Algorithm 2, before it is used in any further calculations. This forces the mutation vector to have a minimal length, without losing the benefits obtained by scaling with the covariance matrix $C$.

### 3.2.7 Two-Point Step-Size Adaptation (TPA)

The step-size $\sigma$ of the CMA-ES using cumulative step size adaptation (CSA) is adapted after every generation according to the evolution path, which incorporates the latest successful individuals. In order to alleviate issues of CSA, Hansen *et al.* propose TPA [11], which reserves two individuals from the $\lambda$ offspring. These are used to evaluate two additional individuals after selection and recombination has taken place:

$$f_+ = f(\mathbf{m} + \alpha'_{\text{TPA}}\sigma\langle\mathbf{y}\rangle) \tag{3.8}$$

$$f_- = f(\mathbf{m} - \alpha'_{\text{TPA}}\sigma\langle\mathbf{y}\rangle) \tag{3.9}$$

where $\mathbf{m}$ is the mean value after recombination and $\mathbf{y}$ is the weighted mutation vector of the previous generation. The step-size should decrease if $f_-$ is better than $f_+$, and increase otherwise. To do so, a multiplication factor $\alpha_{\text{TPA, s}}$ is calculated by

$$\alpha_{\text{TPA, s}} \leftarrow \alpha_{\text{TPA, s}} + c_\alpha(\alpha_{\text{TPA, act}} - \alpha_{\text{TPA, s}}) \tag{3.10}$$

where

$$\alpha_{\text{TPA, act}} \leftarrow \begin{cases} -\alpha & f_- < f_+ \\ \alpha & f_- \geq f_+ \end{cases} \tag{3.11}$$

The step-size $\sigma$ is finally updated using

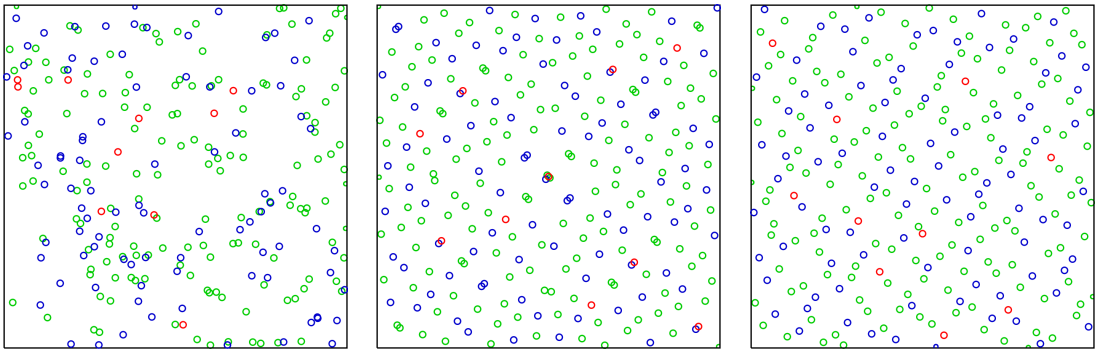
$$\sigma \leftarrow \sigma \cdot \exp(\alpha_{\text{TPA, s}}). \tag{3.12}$$

Default values for the parameters are $\alpha'_{\text{TPA}} = 0.5$, $\alpha_{\text{TPA}} = 0.5$ and $c_\alpha = 0.3$.

### 3.2.8 Pairwise Selection

Use of Mirrored Sampling in an ES with $\mu > 1$ can cause a bias in the length of mutation vectors, as two mirrored vectors will (partially) cancel each other out in recombination. Pairwise Selection is introduced in a later paper by Auger *et al.* [1] to prevent this.

During the Mirrored Sampling, mirrored pairs of individuals are created, and passed in paired order to the selection function. Instead of immediately sorting by fitness, the



(A) Pseudorandom sequence (B) Sobol sequence (C) Halton sequence

FIGURE 3.2: *Quasi-random sequences.* 256 points from a 2-dimensional pseudorandom number source (left); compared with the first 256 points from a 2-dimensional Sobol (center) and Halton sequence (right). The Sobol and Halton sequences cover the space more evenly. (red=1,..,10, blue=11,..,100, green=101,..,256). Source: [16]

lowest fitness of each mirrored pair $i$ is first selected by

$$\mathbf{x}_{\min,i} = \arg\min(f(\mathbf{x}_{2i}), f(\mathbf{x}_{2i+1})). \tag{3.13}$$

Only then are the selected individuals used by the regular selection method.

Although this selection scheme was devised specifically for use with Mirrored Sampling, there is no fundamental problem with using it when mutations are used according to the non-mirrored sampling methods. Therefore both are allowed to be activated independently and no dependency between this Pairwise Selection and Mirrored Sampling is enforced.

### 3.2.9 Recombination Weights

In the standard $(\mu/\mu_W + /, \lambda)$-CMA-ES, weighted recombination is performed with the following weights vector $\mathbf{w}$:

$$w_i = \log(\mu + \frac{1}{2}) - \frac{\log(i)}{\sum_j w_j}. \tag{3.14}$$

Alternative weights that were used in the original introduction of the CMA-ES are the arithmetic mean

$$w_i = \frac{1}{\mu}. \tag{3.15}$$

### 3.2.10 Quasi-Gaussian Sampling

Samples are not necessarily drawn *uniformly* from the normal distribution. Alternatively, the random mutation vectors can be drawn from a quasi-random uniform sequence

as proposed in [3]. Two such sequences are the *Sobol* [20] and *Halton* [7] sequences. Fig. 3.2 shows the distribution of points by a uniform pseudorandom number generator and the Sobol and Halton sequences.

As numbers generated by these sequences are uniformly distributed, a transformation must be applied before they can be used as uniform samples from the normal distribution. This transformation is performed with the inverse of the cumulative distribution function for normal distribution, also known as the percent point function

$$\mathbf{z}_i \leftarrow \text{ppf}(\mathbf{z}_i). \tag{3.16}$$

### 3.2.11 Increasing Population (IPOP)

Initiating a local restart of an ES can be done when the optimization process no longer seems to progress. The criteria listed below check multiple parameters and function values of the ES for degeneration or stagnation. Once such a check fails, the ES is stopped and restarted. All following criteria, including their default values, are used as proposed in [2, 12].

- *Equalfunvalhist*: Stop if the range of the best objective function values of the last $10 + \lceil 30n/\lambda \rceil$ generations is zero.
- *Tolfun*$= 10^{-12}$: Stop if the range of the best objective function values of the last $10 + \lceil 30n/\lambda \rceil$ generations and all function values of the recent generation is below Tolfun.
- *TolX*$= 10^{-12}\sigma^0$: Stop if the standard deviation of the normal distribution is smaller than TolX in all coordinates and $\sigma^t \mathbf{p}_c$ is smaller than TolX in all components.
- *NoEffectAxis*: Stop if adding a 0.1-standard deviation vector in a principal axis direction of $\mathbf{C}^t$ does not change $\langle \mathbf{x} \rangle^t$.
- *NoEffectCoord*: Stop if adding 0:2-standard deviation in each coordinate does change $\langle \mathbf{x} \rangle^t$.
- *MaxIter* $= 100 + 50(n+3)^2/\sqrt{\lambda}$: the maximal number of iterations in each run of CMA-ES
- *EqualFunVals*: in more than $1/3^{\text{rd}}$ of the last $n$ iterations the objective function value of the best and the $k$-th best solution are identical, that is $f(\mathbf{x}_{1:\lambda}) = f(\mathbf{x}_{k:\lambda})$, where $k = 1 + \lceil 0.1 + \lambda/4 \rceil$.

- *TolUpX* $= 10^{12}$: all components of $\mathbf{p}_c^t$ and all square roots of diagonal components of $\mathbf{C}^t$, multiplied by $\sigma^t/\sigma^0$, are smaller than TolUpX.
- *TolUpSigma* $= 10^{20}$: $\sigma^t/\sigma^0 >$ TolUpSigma $\sqrt{l^t}$, where $l^t$ is the largest eigenvalue of $\mathbf{C}^t$, indicates a mismatch between increase and decrease of all eigenvalues in $\mathbf{C}$. In this, rather untypical, case the progression of the strategy is usually very low and a restart is indicated.
- *Stagnation*: the median of the 20 newest values is not smaller than the median of the 20 oldest values, respectively, in the two arrays containg the best function values and the median function values of the last $0.2t + 120 + 30n/\lambda$ iterations.
- *ConditionCov*: the condition number of $\mathbf{C}^t$ exceeds $10^{14}$.

Auger *et al.* proposed an increasing population scheme IPOP [2] to use the remaining function evaluations more effectively after a restart. The population size $\lambda$ is increased with a constant factor between 1.5 and 5 after every restart, where 2 is chosen by default.

Later, Hansen *et al.* introduced the bi-population (BIPOP) [12] variation in which two interlaced regimes are started. Population size is increased by the same factor of two as with IPOP after every restart of the first regime. A small population size is used in the second regime, where $\lambda$ is set according to

$$\lambda_s = \left\lfloor \lambda_{\text{def}} \left( \frac{1}{2} \frac{\lambda_l}{\lambda_{\text{def}}} \right)^{U[0,1]^2} \right\rfloor . \tag{3.17}$$

$U[0,1]$ denotes a number sampled uniformly from the range $[0,1]$. The initial step-size for this small regime is set to $\sigma^0 = 2 \cdot 10^{-2U[0,1]}$, and a maximum budget of half that of the recent budget for the first regime is enforced.

## 3.3 Problematic Combinations

Our aim is to allow for combining any of the ES-variations listed in Section 3.2 such that the resulting ES will run with minimal need of checking dependences between variants, and without causing errors. Selection variations can cause problems when they require more than $\mu$ individuals for the selection process. Descriptions of the encountered issues and our solutions are given below.

### 3.3.1 Pairwise Selection and Sequential Selection

For pairwise selection to return $\mu$ individuals, a selection must be made from at least $2\mu$ individuals. This causes a problem when sequential selection is allowed to stop the generation after $\mu$ individuals, leaving only $\mu/2$ pairs. To solve this, the cut-off point for sequential selection is artificially increased to $2\mu$. If $\lambda < 2\mu$, $\lambda$ is also increased to $2\mu$.

### 3.3.2 Pairwise Selection and TPA

TPA reserves two individuals from the $\lambda$ offspring, preventing them from being used for selection and recombination. This leaves the ES with $\lambda_{\text{eff}} = \lambda - 2$ individuals. When pairwise selection is used and $\lambda = 2\mu$, we are one pair short of being able to select $\mu$ individuals. In this case, $\mu$ is set to $\lambda_{\text{eff}}/2$.

### 3.3.3 Pairwise Selection, Sequential Selection and TPA

When pairwise selection, sequential selection and TPA are all active, both of the issues mentioned above will occur. To remedy this, the cut-off point for sequential selection is based on $\lambda_{\text{eff}}$.

## 3.4 ES Framework

To easily allow the combination of all ES-variations listed in Section 3.2, we created a generic framework based on the CMA-ES (see Algorithm 3). It is designed such that an ES-variation can be activated, by replacing a function or passing an additional boolean variable. Any endogenous variables of the CMA-ES and its variations are abstracted into a single global *parameters* (see line 26) object that is accessible from all other functions.

Only the *structural* variations of the CMA-ES are considered in this research, each using their own parameters. These parameter values are not included in the optimization. Instead, the default values and formulas were used directly from literature where required.

The variable functions are the mutation (line 7), selection, recombination (lines 18–19) and parameter update (line 26). Here, the variability is shown by the added variables such as *sampler* and *threshold* for mutation. The sampler is a special case that merges three variations: Quasi-Gaussian sampling replaces the regular Gaussian sampling that is used as *base-sampler*. If orthogonal sampling is selected, it will use the previously

**Algorithm 3** Customizable CMA-ES Framework

```
1:  t ← 0
2:  f_opt ← inf
3:  x̄ ← randomly generated individual
4:  while not terminate do
5:    canBreak ← False
6:    for i = 1 to λ do                          ▷ Mutation and evaluation
7:      x_i ← mutate(x̄, sampler, threshold)
8:      f_i ← evaluate(x_i)
9:      if f_i < f_opt then
10:       f_opt ← f_i
11:       x_opt ← x_i
12:       canBreak ← True
13:     end if
14:     if SeqSel and i ≥ seq-cutoff and canBreak then   ▷ Sequential selection
15:       break for
16:     end if
17:   end for
18:   P^(t+1) ← select({x⃗_1:λ, ..., x⃗_μ:λ}, elitist, pairwise)
19:   x̄ ← recombine(P^(t+1), weights)
20:   if TPA then
21:     stepSizeChange ← performTPA()
22:   end if
23:   if IPOP or BIPOP then
24:     λ ← updateLambda(IPOP, BIPOP)
25:   end if
26:   parameters.update(stepSizeChange, active)
27:   t ← t + 1
28: end while
```

selected base-sampler as source for vectors to orthonormalize. Mirrored sampling is the last option to be added to the sampler.

Sequential selection (lines 12, 14–16), TPA (lines 20–22) and (B)IPOP (lines 23–25) cannot be passed as parameters to the variable functions as they change the core structure of the main loop in an ES. Instead, these methods are added as separate if-guarded blocks of code.

Each ES-variant can be activated independently of all others. This allows an ES to be represented as a simple list of discrete choices that can be mutated without any need for checking dependencies or validity. Furthermore, functions can be replaced, and new variations can be added without rewriting the entire algorithm.

## 3.5 Representation

| # | Variant name | 0 (default) | 1 | 2 |
|---|---|---|---|---|
| 1 | Active | off | on | - |
| 2 | Elitism | off | on | - |
| 3 | Mirrored Sampling | off | on | - |
| 4 | Orthogonal Sampling | off | on | - |
| 5 | Sequential Selection | off | on | - |
| 6 | Threshold Convergence | off | on | - |
| 7 | TPA | off | on | - |
| 8 | Pairwise Selection | off | on | - |
| 9 | Recombination Weights | $\log(\mu+\frac{1}{2})-\frac{\log(i)}{\sum_j w_j}$ | $\frac{1}{\mu}$ | - |
| 10 | Quasi-Gaussian Sampling | off | Sobol | Halton |
| 11 | Increasing Population | off | IPOP | BIPOP |

TABLE 3.1: *Overview of the available ES variants studied in this research.* For most of these variants the only required options are *off* and *on*, encoded by 0 and 1. For quasi-Gaussian sampling and increasing population, the additional option is encoded by 2. The default CMA-ES is encoded by choosing 0 for all variants. The entries in row 9, recombination weights, specify the formula for calculating each weight $w_i$.

Table 3.1 provides a summarized overview of the ES variants considered in this research in the same order as introduced in Section 3.2. By choosing options for each variant and listing them in the specified order, an ES can be represented as a list of integers. The resulting representations range from **00000000000** = default CMA-ES, to **11111111122** = CMA-ES with all variations activated.

Decoding a given representation $\vec{r} = \{r_1, r_2, \ldots, r_{11}\}$ can be done as follows: For each integer $r_i$ in the representation $\vec{r}$, find the ES variation $i$ in Table 3.1, and use the option indicated by $r_i$. For example: The representation $\vec{r} =$ **01100000100** represents the non-default option for ES-variations 2, 3 and 9: *Elitism*, *mirrored sampling* and *pairwise selection*. In other words: A $(\mu+\lambda)$ mirrored-CMA-ES with pairwise selection.

## 3.6 Genetic Algorithm

A mutation only, self-adaptive GA according to Kruisselbrink *et al.* [17] is used as optimizer for the ES-structure (see Algorithm Algorithm 4). Crossover is omitted to

**Algorithm 4** $(1, \lambda)$-self-adaptive GA

1: $t \leftarrow 0$
2: $P^{(0)} \leftarrow$ generate individual $\vec{I}$, randomly
3: **while** not terminate **do**
4: **for** $i = 1$ to $\lambda$ **do** ▷ Create $\lambda$ offspring
5: $(\vec{r}_i, p_{\mathrm{m},i}) = \vec{I}_i \leftarrow \mathrm{copy}(P^{(t)})$
6: $p_{\mathrm{m},i} \leftarrow \mathrm{mutateRate}(p_{\mathrm{m},i})$ ▷ Update mutation rate
7: $\vec{r}_i \leftarrow \mathrm{mutate}(\vec{r}_i, p_{\mathrm{m},i})$ ▷ Update ES structure with mutation rate $p_{\mathrm{m},i}$
8: $f_i \leftarrow \mathrm{evaluate}(\vec{r}_i)$
9: **end for**
10: $P^{(t+1)} \leftarrow \vec{I}_{1:\lambda}$, select single best from $\lambda$
11: $t \leftarrow t + 1$
12: **end while**

reduce the number of exogenous parameters of our GA. An individual in the GA consists of an ES-structure $\vec{r}$ (previously described in Section 3.2) and the self-adaptive mutation rate $p_{\mathrm{m}}$. This algorithm was picked because of its fast and reliable convergence, as shown in [17].

We define the fitness of an ES with respect to a certain optimization problem as follows: An ES will produce a solution vector $\vec{x}$ with its corresponding fitness value $f(\vec{x})$, which may be different for each run because optimization by an ES is a stochastic process. Using the value of a single run, or the best of several runs can therefore be heavily influenced by outliers. Instead, the *median* value of fifteen runs is used.

# Chapter 4

# Experiments

We use a (1,12) GA with a budget of 250 evaluations of ES-structures for these experiments. This choice is partially based on the length of the representation for an ES, and partially on the desire for fast convergence using parallel computing. Every ES is given a budget of $10^3 n$ function evaluations.

Our algorithm breeding framework is tested using the *black-box optimization benchmark* (BBOB) suite [13]. Use of the BBOB suite allows us to use the distance to target values as absolute fitness of an ES. Fourteen of the more difficult noiseless functions out of the available twenty-four are selected for experiments beforehand. The remaining ten noiseless functions in BBOB are omitted because any variation on the CMA-ES is expected to perform well in these simple, unimodal fitness landscapes. This means that no difference is expected between any ES-structures in terms of the FCE measure. Five dimensionality settings are used for each of the fourteen functions. Table 4.1 lists the functions and dimensionalities that are tested.

To evaluate the convergence of the search towards the optimal ES possible within the framework, a brute force search over all possible ES-structures is also performed. Again, each ES-structure is evaluated by the median result of fifteen runs.

Our framework is written in Python using the *mpi4py* package [9] and the experiments are performed on a cluster, allowing the parallelization of both the twelve individuals per

| | |
|---|---|
| Dimensions | 2, 3, 5, 10, 20 |
| Function IDs | 3, 4, 7, 9, 10, 12, 13, 16, 17, 19, 20, 21, 23, 24 |

TABLE 4.1: List of the five dimensionalities and fourteen BBOB function ID's that are used for the experiments. The remaining ten out of twenty-four BBOB functions are omitted. Each listed function is tested in each of the listed dimensions, for a total of 70 experiments.

generation of the GA and the fifteen runs per ES, resulting in 180-fold parallelization. During the brute force search, sixteen ES-structures were evaluated simultaneously for a total of 240-fold parallelization.

# Chapter 5

# Results

A GA can effectively evolve improving ES structures using our framework. Fig. 5.1 shows this as all GA runs converge in their budget of 250 evaluations, or 21 generations. Most of these runs lasted around 10–11 minutes, with the longest run lasting up to 17 minutes.

Looking at these graphs, a few different cases can be distinguished. For functions F9, F10, F12, F13, F17 and F21, an ES that reaches the BBOB default threshold of $10^{-8}$ to the optimum value is found in the first few generations of the GA. This is also the expected behavior for the functions that were left out for being too simple. Traditional convergence is seen in the graphs for functions F3, F4, F7, F16, F19, F23 and F24, more so for the optimizations in higher dimensions. However, the GA seems unable to improve significantly for the remaining function F20.

The graphs in Fig. 5.1 show a clear ordering between the obtained fitness values for different dimensionalities. Only rarely does the 20-dimensional run converge faster than in any experiment in less dimensions.

Even at 240-fold parallelization, each brute force search lasted between 150 and 210 minutes, meaning the computations for all 70 experiments lasted for nine days in total, compared to the twelve hours in total for all optimizations by the GA.

Tables 5.1 and 5.2 lists the optimal ES structures for each function and number of dimensions that were found by the GA and compare them with the result found by brute force search (i.e., complete enumeration of all possible combinations). As the absolute target values of all BBOB optimization functions are known, no negative fitness values are expeted. The occurrence of values such as $-8.27\text{e}{-16}$ is therefore due to the precision limit of Python/NumPy. It is then also likely that more than one ES-structure reached

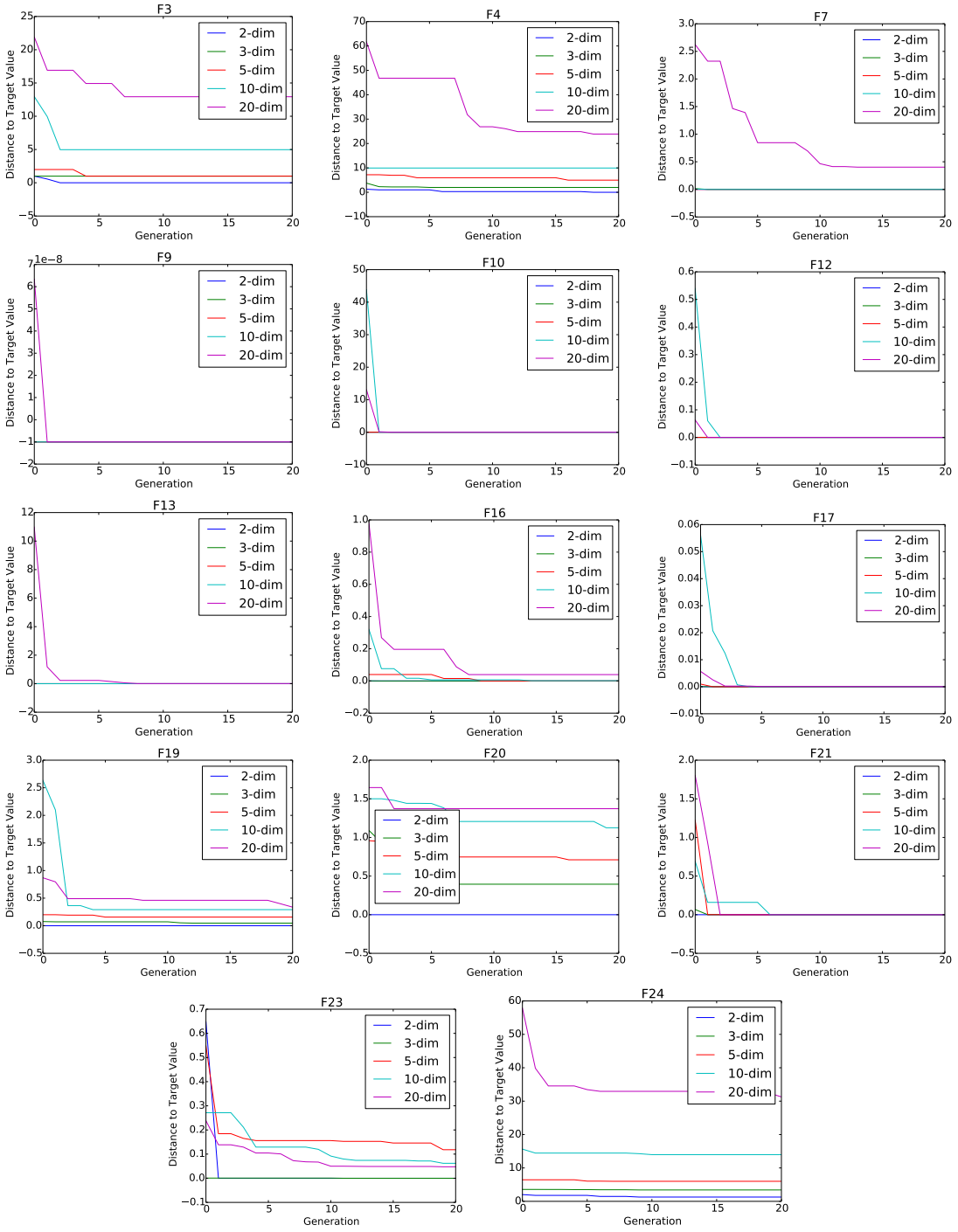



FIGURE 5.1: *GA convergence*. The above graphs show the rate of convergence during the optimization of ES-structures by the GA for different BBOB optimization functions. All five dimensionalities have been plotted per function, each line representing the ES-structures that were found to perform best for that combination during the optimization process.

| F-ID | $N$ | Brute force | Fitness | GA | Fitness |
|---|---|---|---|---|---|
| F3 | 2 | 00000000021 | -8.27e-16 | 01001001011 | -8.27e-16 |
| F3 | 3 | 00101000011 | 0.782 | 10100000001 | 0.995 |
| F3 | 5 | 10100000121 | 1.15 | 10110001022 | 0.995 |
| F3 | 10 | 00000000012 | 3.98 | 00110000001 | 4.97 |
| F3 | 20 | 00100000002 | 14.9 | 00110000011 | 12.9 |
| F4 | 2 | 01111001021 | 6.28e-15 | 11011101011 | 2.58e-11 |
| F4 | 3 | 11001001002 | 1.99 | 00110000012 | 1.99 |
| F4 | 5 | 00001000012 | 3 | 00100000011 | 4.97 |
| F4 | 10 | 00101001022 | 9.95 | 00111000002 | 9.95 |
| F4 | 20 | 00100001101 | 23.9 | 00110001002 | 23.9 |
| F7 | 2 | 00000000002 | -8.27e-16 | 10100000020 | -8.27e-16 |
| F7 | 3 | 00000000001 | -8.27e-16 | 00100010021 | -8.27e-16 |
| F7 | 5 | 00000000012 | -8.27e-16 | 11000111122 | -8.27e-16 |
| F7 | 10 | 00000110001 | -8.27e-16 | 00100110001 | -8.27e-16 |
| F7 | 20 | 00100001022 | 0.453 | 00000110011 | 0.403 |
| F9 | 2 | 00000000000 | -8.27e-16 | 11011001011 | -8.27e-16 |
| F9 | 3 | 00000000000 | -8.27e-16 | 01011001001 | -8.27e-16 |
| F9 | 5 | 00000000000 | -8.27e-16 | 00001001000 | -8.27e-16 |
| F9 | 10 | 00000000000 | -8.27e-16 | 00001001010 | -8.27e-16 |
| F9 | 20 | 00110000010 | -8.27e-16 | 00110001010 | -8.27e-16 |
| F10 | 2 | 00000000000 | -8.27e-16 | 01010110011 | -8.27e-16 |
| F10 | 3 | 00000000000 | -8.27e-16 | 01010010102 | -8.27e-16 |
| F10 | 5 | 00000000000 | -8.27e-16 | 01101010122 | -8.27e-16 |
| F10 | 10 | 00000000000 | -8.27e-16 | 01010000100 | -8.27e-16 |
| F10 | 20 | 00000000010 | -8.27e-16 | 01100000110 | 6.28e-15 |
| F12 | 2 | 00000000000 | -8.27e-16 | 11110111101 | -8.27e-16 |
| F12 | 3 | 00000000000 | -8.27e-16 | 01100001011 | -8.27e-16 |
| F12 | 5 | 00000000010 | -8.27e-16 | 01100000120 | -8.27e-16 |
| F12 | 10 | 00110001010 | -8.27e-16 | 00100000012 | 2.19e-13 |
| F12 | 20 | 00001000011 | 7.31e-11 | 00010000011 | 1.39e-08 |
| F13 | 2 | 00001001020 | -8.27e-16 | 01100000120 | -8.27e-16 |
| F13 | 3 | 10111001000 | 7.17e-15 | 01100000110 | 6.28e-15 |
| F13 | 5 | 01000000110 | 1.06e-13 | 10111001020 | 1.06e-13 |
| F13 | 10 | 00011001020 | 4.7e-10 | 01101001101 | 2.67e-08 |
| F13 | 20 | 00000000000 | 0.0012 | 10100001012 | 0.00753 |

TABLE 5.1: *Best ES-structure found by brute force search and GA: F3 – F13.* This table lists the best ES found by brute force search and our GA for each combination of dimensions and function from BBOB. An ES-structure is represented by a list of integers as explained in Section 3.2. Underlined integers in the GA-column indicate a difference between the ES found by the GA and the brute force search. Fitness values are given for each ES as the median of fifteen runs. A negative fitness value indicates that the target value was approached within $10^{-8}$. All runs were performed with an evaluation budget of $10^3 D$.

| F-ID | $N$ | Brute force | Fitness | GA | Fitness |
|---|---|---|---|---|---|
| F16 | 2 | 00000000001 | -8.27e-16 | 00001000121 | -8.27e-16 |
| F16 | 3 | 00000000000 | -8.27e-16 | 10000000011 | -8.27e-16 |
| F16 | 5 | 00000001022 | -8.27e-16 | 00000000012 | -8.27e-16 |
| F16 | 10 | 00100001012 | 0.00586 | 00110001002 | 0.000969 |
| F16 | 20 | 00100001011 | 0.0566 | 00110001012 | 0.0394 |
| F17 | 2 | 00100001020 | -8.27e-16 | 11001001110 | -8.27e-16 |
| F17 | 3 | 00001000020 | -8.27e-16 | 10111001010 | -8.27e-16 |
| F17 | 5 | 00111000011 | 5.31e-13 | 00001000021 | 6.14e-13 |
| F17 | 10 | 00110000001 | 1.69e-06 | 00110001011 | 7.74e-06 |
| F17 | 20 | 00110001001 | 7.84e-05 | 00110001012 | 3.89e-05 |
| F19 | 2 | 11111001102 | 6.08e-17 | 11001000012 | 1.84e-15 |
| F19 | 3 | 10100000012 | 0.0393 | 10100000022 | 0.0472 |
| F19 | 5 | 00110000101 | 0.107 | 00110000110 | 0.157 |
| F19 | 10 | 00110000112 | 0.178 | 00111010001 | 0.292 |
| F19 | 20 | 00100000110 | 0.312 | 00100000112 | 0.337 |
| F20 | 2 | 11100010021 | -8.27e-16 | 11001001022 | -8.27e-16 |
| F20 | 3 | 01010111012 | 0.214 | 11010111101 | 0.395 |
| F20 | 5 | 01110111021 | 0.671 | 01000110021 | 0.711 |
| F20 | 10 | 01101011001 | 1.18 | 01010110022 | 1.13 |
| F20 | 20 | 01101010002 | 1.44 | 00000010021 | 1.37 |
| F21 | 2 | 00011000002 | -7.93e-15 | 11100100021 | -7.93e-15 |
| F21 | 3 | 00110001012 | -7.93e-15 | 01001100022 | -7.93e-15 |
| F21 | 5 | 10000001012 | -7.93e-15 | 11100111121 | 6.08e-17 |
| F21 | 10 | 00111000001 | 6.08e-17 | 00110001022 | 6.08e-17 |
| F21 | 20 | 01101000101 | 2.05e-14 | 01101001002 | 1.06e-13 |
| F23 | 2 | 00011000000 | 6.28e-15 | 10110001000 | 6.28e-15 |
| F23 | 3 | 00100000020 | 4.18e-13 | 00000000010 | 2.3e-11 |
| F23 | 5 | 10111000022 | 0.0939 | 10110000022 | 0.118 |
| F23 | 10 | 00111000021 | 0.0882 | 00111001021 | 0.0619 |
| F23 | 20 | 00100000001 | 0.0491 | 00100000002 | 0.0474 |
| F24 | 2 | 11110010011 | 0.709 | 10110101110 | 1.29 |
| F24 | 3 | 10000010011 | 3.15 | 00000010011 | 3.41 |
| F24 | 5 | 10000011012 | 5.9 | 10110000021 | 6 |
| F24 | 10 | 00110010001 | 13.6 | 00110000111 | 14 |
| F24 | 20 | 00100010002 | 32 | 00110000102 | 31.3 |

TABLE 5.2: *Best ES-structure found by brute force search and GA: F16 – F24.* This table lists the best ES found by brute force search and our GA for each combination of dimensions and function from BBOB. An ES-structure is represented by a list of integers as explained in Section 3.2. Underlined integers in the GA-column indicate a difference between the ES found by the GA and the brute force search. Fitness values are given for each ES as the median of fifteen runs. A negative fitness value indicates that the target value was approached within $10^{-8}$. All runs were performed with an evaluation budget of $10^3 D$.

| | GA | | | Brute Force | | |
|---|---|---|---|---|---|---|
| # | 0 | 1 | 2 | 0 | 1 | 2 |
| 1 | 48 | 22 | - | 59 | 11 | - |
| 2 | 44 | 26 | - | 59 | 11 | - |
| 3 | 26 | 44 | - | 35 | 35 | - |
| 4 | 37 | 33 | - | 49 | 21 | - |
| 5 | 50 | 20 | - | 50 | 20 | - |
| 6 | 57 | 13 | - | 67 | 3 | - |
| 7 | 55 | 15 | - | 59 | 11 | - |
| 8 | 41 | 29 | - | 49 | 21 | - |
| 9 | 51 | 19 | - | 62 | 8 | - |
| 10 | 18 | 30 | 22 | 33 | 22 | 15 |
| 11 | 16 | 30 | 24 | 25 | 24 | 21 |

TABLE 5.3: Frequency analysis of the options chosen per variant by both the GA and brute force search.

this value, but if two ES-structures reach this same value, they are indistinguishable due to our use of the FCE measure.

There is no exact match between the best found ES-structures according to the brute force search and the GA search, as there is always a difference in one or more of the active variants. Despite these differences in the ES-structures, the GA is able to match the fitness of the best performing ES as found by the brute force search.

Note that in some cases, the fitness value for the GA-found ES-structure is actually lower than the value for the value found by brute force search. Examples of this are F3 in 5D and 20D, F7 in 20D, F16 in 10D, F17 in 20D, F20 in 10D and 20D, F21 in 5D, F23 in 10D and 20D and F24 in 20D.

The ES-structure represented by **00000000000** is reported surprisingly often as best-performing structure by the brute force search for F9, F10 and F12. This confirms the results shown by Fig. 5.1: Any ES-structure is good enough to reach the practical optimization limit. The default CMA-ES is then only selected because in the order of evaluation, it is the first structure to reach this limit.

By examining the frequency of the integers in each position of the ES-representation (Table 5.3), it can be seen that all choices are relatively equally distributed. The default CMA-ES seems to be more prevalent in the results found by brute force search than by the GA, as indicated by the higher frequency of the 0 option. Overall trends are similar in both sets of results: *Mirrored Sampling*, *Quasi-Gaussian Sampling* and *Increasing Population* are the most activated options.

Finally, Tables 5.4 to 5.8 shows the *estimated running time* (ERT) measure as calculated by the BBOB suite for the ES-structures found by the GA. For these results, fifteen runs were performed with a larger evaluation budget of $10^4 n$. Note that in these tables, a comparison is made to the best performing optimization method from BBOB-2009, which is often not an ES-variation.

| $\Delta f$ | 1e+1 | 1e+0 | 1e-1 | 1e-2 | 1e-3 | 1e-5 | 1e-7 | #succ |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{f_3}$ | 15 | 271 | 445 | 446 | 450 | 454 | 464 | 15/15 |
| | 3.9(3) | 5.0(4) | 7.9(10) | 8.1(8) | 8.2(11) | 8.6(11) | 8.7(10) | 15/15 |
| $\mathbf{f_4}$ | 22 | 344 | 459 | 496 | 523 | 544 | 566 | 15/15 |
| | 5.2(3) | 14(27) | 27(15) | 26(26) | 25(37) | 28(21) | 49(97) | 7/15 |
| $\mathbf{f_7}$ | 3.2 | 21 | 60 | 193 | 217 | 217 | 241 | 15/15 |
| | 5.8(6) | 11(16) | 13(12) | 5.9(5) | 5.9(4) | 5.9(7) | 5.3(8) | 15/15 |
| $\mathbf{f_9}$ | 1 | 18 | 30 | 44 | 68 | 81 | 92 | 15/15 |
| | 49(44) | 10(13) | 11(11) | 10(10) | 6.8(4) | 6.5(5) | 6.3(2) | 15/15 |
| $\mathbf{f_{10}}$ | 30 | 46 | 54 | 61 | 68 | 82 | 98 | 15/15 |
| | 4.6(4) | 4.1(2) | 4.5(3) | 4.8(2) | 4.8(1) | 4.8(0.3) | 4.8(0.8) | 15/15 |
| $\mathbf{f_{12}}$ | 35 | 46 | 75 | 94 | 105 | 153 | 195 | 15/15 |
| | 8.5(12) | 12(18) | 11(16) | 13(13) | 25(95) | 24(41) | 24(30) | 13/15 |
| $\mathbf{f_{13}}$ | 23 | 35 | 46 | 60 | 71 | 95 | 122 | 15/15 |
| | 5.8(5) | 7.7(10) | 7.9(5) | 7.6(5) | 7.1(3) | 6.4(3) | 6.0(2) | 15/15 |
| $\mathbf{f_{16}}$ | 9.1 | 50 | 174 | 326 | 358 | 409 | 538 | 15/15 |
| | 11(34) | 7.6(11) | 10(3) | 6.4(2) | 6.1(3) | 5.5(7) | 4.5(7) | 15/15 |
| $\mathbf{f_{17}}$ | 2.7 | 61 | 133 | 275 | 396 | 1086 | 1657 | 15/15 |
| | 3.2(3) | 7.2(0.9) | 6.8(21) | 6.3(21) | 5.6(7) | 6.4(6) | 4.3(3) | 13/15 |
| $\mathbf{f_{19}}$ | 1 | 1 | 26 | 216 | 227 | 252 | 276 | 15/15 |
| | 6.1(5) | 31(28) | 7.5(5) | 29(37) | 27(25) | 25(67) | 23(17) | 14/15 |
| $\mathbf{f_{20}}$ | 3.7 | 61 | 365 | 366 | 366 | 370 | 375 | 15/15 |
| | 3.9(2) | 23(35) | 8.9(6) | 9.1(11) | 9.3(9) | 9.4(10) | 10(10) | 15/15 |
| $\mathbf{f_{21}}$ | 1.7 | 51 | 174 | 276 | 290 | 324 | 330 | 15/15 |
| | 1.2(0.8) | 1.2(0.5) | 1.0(0.7) | 1.0(0.8) | 1.3(0.6) | 1.7(1) | 2.2(0.7) | 15/15 |
| $\mathbf{f_{23}}$ | 7.8 | 193 | 234 | 263 | 299 | 348 | 379 | 15/15 |
| | 1.7(0.7) | 5.8(9) | 30(67) | 32(21) | 28(68) | 25(29) | 23(53) | 11/15 |
| $\mathbf{f_{24}}$ | 18 | 857 | 8515 | 23399 | 24113 | 24721 | 24721 | 5/15 |
| | 2.1(1) | 16(16) | 10(13) | 3.5(4) | 3.4(8) | 3.3(2) | 3.3(4) | 3/15 |

TABLE 5.4: *Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 for F3–F24 in 2D space.* The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-ile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target [Df]-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. **Bold** entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

| $\Delta f$ | 1e+1 | 1e+0 | 1e-1 | 1e-2 | 1e-3 | 1e-5 | 1e-7 | #succ |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{f_3}$ | 38 | 822 | 830 | 835 | 842 | 847 | 853 | 15/15 |
| | 7.1(5) | 2.2(4) | 13(18) | 13(21) | 13(12) | 14(2) | 14(15) | 12/15 |
| $\mathbf{f_4}$ | 40 | 808 | 866 | 921 | 952 | 1015 | 1044 | 15/15 |
| | 6.3(3) | 33(50) | 161(233) | 151(249) | 146(99) | 138(154) | 135(206) | 3/15 |
| $\mathbf{f_7}$ | 11 | 65 | 342 | 464 | 482 | 482 | 535 | 15/15 |
| | 4.2(2) | 2.2(4) | 2.1(0.8) | 1.9(1) | 2.5(2) | 2.5(1) | 2.3(2) | 15/15 |
| $\mathbf{f_9}$ | 21 | 65 | 127 | 149 | 159 | 169 | 178 | 15/15 |
| | 12(3) | 13(12) | 8.7(7) | 8.6(5) | 8.6(6) | 8.8(5) | 8.8(6) | 15/15 |
| $\mathbf{f_{10}}$ | 114 | 152 | 168 | 180 | 194 | 218 | 242 | 15/15 |
| | 4.2(2) | 3.9(1) | 4.1(1.0) | 4.3(1) | 4.3(1) | 4.2(1) | 4.1(0.8) | 15/15 |
| $\mathbf{f_{12}}$ | 65 | 168 | 338 | 401 | 445 | 696 | 790 | 15/15 |
| | 7.6(8) | 4.8(7) | 4.2(3) | 4.3(4) | 4.5(4) | 3.7(3) | 3.8(3) | 15/15 |
| $\mathbf{f_{13}}$ | 49 | 85 | 108 | 136 | 215 | 281 | 365 | 15/15 |
| | 4.5(5) | 5.1(4) | 5.4(2) | 5.0(1) | 3.6(1) | 3.6(1) | 3.3(0.5) | 15/15 |
| $\mathbf{f_{16}}$ | 41 | 319 | 582 | 789 | 1864 | 3204 | 3361 | 15/15 |
| | 1.4(3) | 2.5(3) | 1.6(2) | 1.5(1) | 0.71(0.6) | **0.45**(0.3)$^{\downarrow 2}$ | **0.46**(0.3)$^{\downarrow 2}$ | 15/15 |
| $\mathbf{f_{17}}$ | 3.6 | 78 | 282 | 491 | 1134 | 2347 | 3469 | 15/15 |
| | 189(699) | 11(33) | 7.8(51) | 5.2(15) | 2.3(0.1) | 15(19) | 10(15) | 7/15 |
| $\mathbf{f_{19}}$ | 1 | 1 | 109 | 6764 | 7367 | 7399 | 7441 | 15/15 |
| | 12(10) | 260(134) | 49(44) | 5.4(4) | 6.7(9) | 7.9(6) | 7.9(9) | 6/15 |
| $\mathbf{f_{20}}$ | 8.3 | 385 | 2291 | 2398 | 2481 | 2573 | 2776 | 15/15 |
| | 1.9(1) | 5.7(2) | 10(10) | 10(16) | 10(23) | 9.5(13) | 9.0(10) | 10/15 |
| $\mathbf{f_{21}}$ | 5.9 | 184 | 425 | 439 | 458 | 469 | 482 | 14/15 |
| | 1.4(0.6) | 3.7(4) | 4.0(7) | 4.8(6) | 4.8(6) | 5.8(4) | 6.5(4) | 15/15 |
| $\mathbf{f_{23}}$ | 2.6 | 407 | 906 | 1215 | 2214 | 2293 | 2393 | 15/15 |
| | 3.9(4) | 23(37) | 67(66) | 50(62) | 28(31) | 27(33) | 26(35) | 5/15 |
| $\mathbf{f_{24}}$ | 97 | 10391 | 1.0e5 | 3.6e5 | 3.6e5 | 3.6e5 | 3.6e5 | 2/15 |
| | 2.7(4) | 19(30) | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$*3.0e4* | 0/15 |

TABLE 5.5: *Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 for F3–F24 in 3D space.* The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-ile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target [Df]-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. **Bold** entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

| $\Delta f$ | 1e+1 | 1e+0 | 1e-1 | 1e-2 | 1e-3 | 1e-5 | 1e-7 | #succ |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{f_3}$ | 716 | 1622 | 1637 | 1642 | 1646 | 1650 | 1654 | 15/15 |
| | 1.2(0.6) | 14(20) | 98(93) | 98(99) | 98(100) | 98(107) | 98(189) | 4/15 |
| $\mathbf{f_4}$ | 809 | 1633 | 1688 | 1758 | 1817 | 1886 | 1903 | 15/15 |
| | 1.1(1) | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$*5.0e4* | 0/15 |
| $\mathbf{f_7}$ | 24 | 324 | 1171 | 1451 | 1572 | 1572 | 1597 | 15/15 |
| | 3.8(6) | 2.9(12) | 2.0(1) | 2.0(2) | 1.9(0.5) | 1.9(0.5) | 1.9(1) | 15/15 |
| $\mathbf{f_9}$ | 35 | 127 | 214 | 263 | 300 | 335 | 369 | 15/15 |
| | 14(2) | 108(198) | 66(234) | 55(48) | 49(126) | 44(113) | 41(69) | 12/15 |
| $\mathbf{f_{10}}$ | 349 | 500 | 574 | 607 | 626 | 829 | 880 | 15/15 |
| | 2.8(0.9) | 2.7(0.3) | 2.7(0.3) | 2.6(0.3) | 2.6(0.3) | 2.1(0.3) | 2.1(0.3) | 15/15 |
| $\mathbf{f_{12}}$ | 108 | 268 | 371 | 413 | 461 | 1303 | 1494 | 15/15 |
| | 5.8(2) | 4.3(3) | 5.0(2) | 5.5(4) | 5.5(2) | 2.5(2) | 2.5(1) | 15/15 |
| $\mathbf{f_{13}}$ | 132 | 195 | 250 | 319 | 1310 | 1752 | 2255 | 15/15 |
| | 4.1(0.9) | 4.3(1) | 4.7(2) | 4.6(2) | 1.3(0.6) | 1.7(2) | 1.6(2) | 15/15 |
| $\mathbf{f_{16}}$ | 120 | 612 | 2662 | 10163 | 10449 | 11644 | 12095 | 15/15 |
| | 2.0(1) | 2.7(2) | 1.7(2) | 0.98(0.8) | 1.2(1) | 1.2(0.9) | 1.2(0.7) | 15/15 |
| $\mathbf{f_{17}}$ | 5.2 | 215 | 899 | 2861 | 3669 | 6351 | 7934 | 15/15 |
| | 5.8(5) | 1.6(0.6) | 0.54(0.1) | 0.24(0.1) | 0.64(0.7) | 0.89(0.8) | 0.99(0.8) | 15/15 |
| $\mathbf{f_{19}}$ | 1 | 1 | 242 | 1.0e5 | 1.2e5 | 1.2e5 | 1.2e5 | 15/15 |
| | 33(22) | 525(308) | 577(518) | $\infty$ | $\infty$ | $\infty$ | $\infty$*5.0e4* | 0/15 |
| $\mathbf{f_{20}}$ | 16 | 851 | 38111 | 51362 | 54470 | 54861 | 55313 | 14/15 |
| | 2.9(2) | 11(17) | 5.8(5) | 4.3(8) | 4.1(4) | 4.1(8) | 4.0(4) | 3/15 |
| $\mathbf{f_{21}}$ | 41 | 1157 | 1674 | 1692 | 1705 | 1729 | 1757 | 14/15 |
| | 4.5(8) | 8.2(23) | 11(18) | 11(30) | 11(22) | 11(10) | 11(10) | 12/15 |
| $\mathbf{f_{23}}$ | 3.0 | 518 | 14249 | 27890 | 31654 | 33030 | 34256 | 15/15 |
| | 2.0(2) | 1.9(0.5) | 2.0(3) | 1.8(2) | 2.6(3) | 3.1(6) | 3.0(2) | 6/15 |
| $\mathbf{f_{24}}$ | 1622 | 2.2e5 | 6.4e6 | 9.6e6 | 9.6e6 | 1.3e7 | 1.3e7 | 3/15 |
| | 1.0(0.9) | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$*5.0e4* | 0/15 |

TABLE 5.6: *Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 for F3–F24 in 5D space.* The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-ile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target [Df]-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. **Bold** entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

| $\Delta f$ | 1e+1 | 1e+0 | 1e-1 | 1e-2 | 1e-3 | 1e-5 | 1e-7 | #succ |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{f_3}$ | 1739 | 3600 | 3609 | 3636 | 3642 | 3646 | 3651 | 15/15 |
| | 2.6(3) | 393(417) | ∞ | ∞ | ∞ | ∞ | ∞*1.0e5* | 0/15 |
| $\mathbf{f_4}$ | 2234 | 3626 | 3660 | 3695 | 3707 | 3744 | 28767 | 12/15 |
| | 12(12) | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*1.0e5* | 0/15 |
| $\mathbf{f_7}$ | 172 | 1611 | 4195 | 5099 | 5141 | 5141 | 5389 | 15/15 |
| | 2.0(0.6) | 1.7(0.7) | 1.1(1) | 1.3(0.7) | 1.4(0.7) | 1.4(0.5) | 1.3(0.6) | 15/15 |
| $\mathbf{f_9}$ | 200 | 648 | 857 | 993 | 1065 | 1138 | 1185 | 15/15 |
| | 5.1(1) | 17(0.8) | 14(1) | 13(0.6) | 12(24) | 11(0.8) | 11(0.8) | 14/15 |
| $\mathbf{f_{10}}$ | 1835 | 2172 | 2455 | 2728 | 2802 | 4543 | 4739 | 15/15 |
| | 2.2(0.7) | 2.2(1.0) | 2.1(0.4) | 1.9(0.8) | 1.9(0.7) | 1.2(0.2) | 1.2(0.3) | 15/15 |
| $\mathbf{f_{12}}$ | 515 | 896 | 1240 | 1390 | 1569 | 3660 | 5154 | 15/15 |
| | 2.3(0.3) | 1.9(0.8) | 2.5(1) | 2.8(0.6) | 3.1(1) | 1.8(0.6) | 1.5(0.3) | 15/15 |
| $\mathbf{f_{13}}$ | 387 | 596 | 797 | 1014 | 4587 | 6208 | 7779 | 15/15 |
| | 5.3(6) | 5.8(5) | 7.3(4) | 6.6(4) | 1.8(1) | 2.3(2) | 2.8(1) | 15/15 |
| $\mathbf{f_{16}}$ | 425 | 7029 | 15779 | 45669 | 51151 | 65798 | 71570 | 15/15 |
| | 2.5(0.5) | **0.21**(0.0)$^{\downarrow 3}$ | **0.33**(0.5)$^{\downarrow 2}$ | **0.42**(0.7)$^{\downarrow}$ | 0.98(1) | 1.6(3) | 1.5(1) | 9/15 |
| $\mathbf{f_{17}}$ | 26 | 429 | 2203 | 6329 | 9851 | 20190 | 26503 | 15/15 |
| | 3.6(3) | 1.9(0.3) | 0.52(0.1) | **0.29**(0.0)$^{\downarrow}$ | **0.35**(0.4)$^{\downarrow 3}$ | 0.73(0.4) | 0.87(0.5) | 15/15 |
| $\mathbf{f_{19}}$ | 1 | 1 | 10609 | 9.8e5 | 1.4e6 | 1.4e6 | 1.4e6 | 15/15 |
| | 54(32) | 5307(508) | 69(71) | ∞ | ∞ | ∞ | ∞*1.0e5* | 0/15 |
| $\mathbf{f_{20}}$ | 32 | 15426 | 5.5e5 | 5.7e5 | 5.7e5 | 5.8e5 | 5.9e5 | 15/15 |
| | 3.8(1) | 5.7(4) | ∞ | ∞ | ∞ | ∞ | ∞*1.0e5* | 0/15 |
| $\mathbf{f_{21}}$ | 130 | 2236 | 4392 | 4487 | 4618 | 5074 | 11329 | 15/15 |
| | 13(12) | 10(24) | 13(24) | 13(16) | 13(12) | 12(45) | 5.2(4) | 11/15 |
| $\mathbf{f_{23}}$ | 2.8 | 915 | 16425 | 1.8e5 | 2.0e5 | 2.1e5 | 2.1e5 | 15/15 |
| | 1.7(2) | 10(6) | 0.90(0.6) | 2.5(3) | 7.2(4) | 7.0(7) | 6.9(6) | 1/15 |
| $\mathbf{f_{24}}$ | 98761 | 1.0e6 | 7.5e7 | 7.5e7 | 7.5e7 | 7.5e7 | 7.5e7 | 1/15 |
| | 15(14) | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*1.0e5* | 0/15 |

TABLE 5.7: *Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 for F3–F24 in 10D space.* The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-ile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target [Df]-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. **Bold** entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

| $\Delta f$ | 1e+1 | 1e+0 | 1e-1 | 1e-2 | 1e-3 | 1e-5 | 1e-7 | #succ |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{f_3}$ | 5066 | 7626 | 7635 | 7637 | 7643 | 7646 | 7651 | 15/15 |
| | 85(80) | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_4}$ | 4722 | 7628 | 7666 | 7686 | 7700 | 7758 | 1.4e5 | 9/15 |
| | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_7}$ | 1351 | 4274 | 9503 | 16523 | 16524 | 16524 | 16969 | 15/15 |
| | 1.3(0.8) | 3.3(2) | 5.3(5) | 7.6(8) | 7.7(10) | 7.7(12) | 7.5(5) | 12/15 |
| $\mathbf{f_9}$ | 1716 | 3102 | 3277 | 3379 | 3455 | 3594 | 3727 | 15/15 |
| | 4.3(1) | 14(32) | 14(0.3) | 14(30) | 14(58) | 13(15) | 13(27) | 13/15 |
| $\mathbf{f_{10}}$ | 7413 | 8661 | 10735 | 13641 | 14920 | 17073 | 17476 | 15/15 |
| | 1.7(0.3) | 1.8(0.3) | 1.6(0.2) | 1.3(0.1) | 1.2(0.1) | 1.1(0.1) | 1.1(0.1) | 15/15 |
| $\mathbf{f_{12}}$ | 1042 | 1938 | 2740 | 3156 | 4140 | 12407 | 13827 | 15/15 |
| | 2.4(0.1) | 2.5(1) | 2.9(2) | 3.3(1) | 3.1(0.9) | 1.4(0.5) | 1.5(0.4) | 15/15 |
| $\mathbf{f_{13}}$ | 652 | 2021 | 2751 | 3507 | 18749 | 24455 | 30201 | 15/15 |
| | 4.5(0.8) | 3.8(2) | 7.6(9) | 11(15) | 3.7(5) | 35(24) | 45(36) | 1/15 |
| $\mathbf{f_{16}}$ | 1384 | 27265 | 77015 | 1.4e5 | 1.9e5 | 2.0e5 | 2.2e5 | 15/15 |
| | 1.5(0.3) | 0.16(0.2) | **0.42**(0.4)$^{\downarrow 2}$ | 1.6(1) | 2.2(1) | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_{17}}$ | 63 | 1030 | 4005 | 12242 | 30677 | 56288 | 80472 | 15/15 |
| | 13(2) | 1.3(0.2) | 0.52(0.1) | 0.60(0.5) | **0.55**(0.2)$^{\downarrow 3}$ | 2.9(3) | 11(8) | 2/15 |
| $\mathbf{f_{19}}$ | 1 | 1 | 3.4e5 | 4.7e6 | 6.2e6 | 6.7e6 | 6.7e6 | 15/15 |
| | 761(435) | 4680(2932) | ∞ | ∞ | ∞ | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_{20}}$ | 82 | 46150 | 3.1e6 | 5.5e6 | 5.5e6 | 5.6e6 | 5.6e6 | 14/15 |
| | 8.7(3) | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_{21}}$ | 561 | 6541 | 14103 | 14318 | 14643 | 15567 | 17589 | 15/15 |
| | 3.0(4) | 8.1(16) | 10(9) | 10(9) | 10(16) | 9.4(13) | 8.4(12) | 10/15 |
| $\mathbf{f_{23}}$ | 3.2 | 1614 | 67457 | 3.7e5 | 4.9e5 | 8.1e5 | 8.4e5 | 15/15 |
| | 1.5(1) | 2.0(0.1) | **0.24**(0.2)$^{\downarrow 2}$ | 8.2(7) | ∞ | ∞ | ∞*2.0e5* | 0/15 |
| $\mathbf{f_{24}}$ | 1.3e6 | 7.5e6 | 5.2e7 | 5.2e7 | 5.2e7 | 5.2e7 | 5.2e7 | 3/15 |
| | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞*2.0e5* | 0/15 |

TABLE 5.8: *Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 for F3–F24 in 20D space.* The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-ile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target [Df]-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. **Bold** entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

# Chapter 6

# Conclusions

By extracting structural features from different CMA-ES variations, a framework for arbitrarily combining ES-variants into new ES-structures can be created. Varying between these structures for different optimization problems can be done to a large extent by using this framework. Furthermore, a GA can be used on top of this framework to evolve an ES-structure that is optimal for a given optimization problem. The GA is able to converge fast, and does so consistently for all given problems.

It is crucial to the performance of the GA how the fitness of an ES is evaluated. Using the median fitness of fifteen independent runs reduces the influence of both positive and negative outliers, making it a more stable measure of fitness than using a mean or even just a single run. However, the fitness value of an ES is not completely stable as shown by the results in Tables 5.1 and 5.2 when the GA finds a better fitness value than the brute force search. It would be interesting to see if stability of these fitness values can be increased by using the median of more runs, or by using a different, more rigorous statistical test.

All implemented ES-variations can be activated independently in our presented framework, with minimal dependency checking required. This approach can be extended to include other ES variants that have not been considered in this research. Every ES-variation added will increase the search space significantly. A brute force search lasting 9 days under 240-fold parallelization is already on the edge of practicality, making it impractical to explore any expansion of the search space using brute force methods.

Although the optimization of the ES-structure is successful, the final results do not always reach the desired BBOB default threshold of $10^{-8}$ distance to the target value, in both the $10^3n$ and $10^4n$ experiments. However, these results have always been obtained by using the default parameter settings for each of the considered variations. As the

number of parameters add up in the combined ES-structures, it is likely that parameter tuning can further improve the results. This could be performed by a separate method after an ES-structure has been determined, or the parameters could be added into the representation of an ES by using a mixed-integer approach and optimized together with the structure.

It is interesting to note that *Mirrored Sampling*, *Quasi-Gaussian Sampling* and *Increasing Population* (both IPOP and BIPOP) are the only methods that are selected in more than 50% of all cases. Although this suggests that the default CMA-ES is better in the general case, there are always a few variants that have a positive impact on the convergence speed when activated.

The positive results of this research are a validation of this method for adapting the structure of an ES, and evolving it using another evolutionary algorithm. Now this method has been validated, future research can focus on extending this framework and applying it in practical settings, by optimizing the ES-structure for classes of similar problems, thereby avoiding the drawback of the No Free Lunch theorem.

# Bibliography


[1] A. Auger, D. Brockhoff, and N. Hansen. Mirrored Sampling in Evolution Strategies with Weighted Recombination. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 861–868. ACM, 2011.

[2] A. Auger and N. Hansen. A Restart CMA Evolution Strategy with Increasing Population Size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE, 2005.

[3] A. Auger, M. Jebalia, and O. Teytaud. Algorithms (x, sigma, eta): Quasi-Random Mutations for Evolution Strategies. In *Artificial Evolution*, pages 296–307. Springer, 2006.

[4] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. PhD thesis, Fakultät für Informatik, Technische Universität Dortmund, Germany, 1995.

[5] T. Bäck, C. Foussette, and P. Krause. *Contemporary Evolution Strategies*. Natural Computing Series. Springer Berlin Heidelberg, 2013.

[6] Å. Björck. Numerics of gram-schmidt orthogonalization. *Linear Algebra and its Applications*, 197198:297 – 316, 1994.

[7] E. Braaten and G. Weller. An improved low-discrepancy sequence for multidimensional quasi-monte carlo integration. *Journal of Computational Physics*, 33(2):249 – 258, 1979.

[8] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm. Mirrored sampling and sequential selection for evolution strategies. In *Parallel Problem Solving from Nature, PPSN XI*, pages 11–21. Springer, 2010.

[9] L. Dalcn, R. Paz, and M. Storti. MPI for Python . *Journal of Parallel and Distributed Computing*, 65(9):1108 – 1115, 2005.

[10] N. Hansen. The CMA Evolution Strategy: A tutorial. *Vu le*, 29, 2005. URL: https://www.lri.fr/~hansen/cmatutorial.pdf.

[11] N. Hansen. CMA-ES with Two-Point Step-Size Adaptation. *CoRR*, abs/0805.0231, 2008.

[12] N. Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2389–2396. ACM, 2009.

[13] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

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

[15] G. Jastrebski, D. V. Arnold, et al. Improving Evolution Strategies Through Active Covariance Matrix Adaptation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2814–2821. IEEE, 2006.

[16] Jheald. Pseudorandom, Sobol and Halton Sequence Plots, 2011. https://commons.wikimedia.org/wiki/User:Jheald/diagrams. Visited: 2016-02-16.

[17] J. Kruisselbrink, R. Li, E. Reehuis, J. Eggermont, and T. Bäck. On the Log-Normal Self-Adaptation of the Mutation Rate in Binary Search Spaces. In *GECCO'11*, pages 893–900. ACM, 2011.

[18] M. A. Martin and D. R. Tauritz. Evolving Black-Box Search Algorithms Employing Genetic Programming. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1497–1504. ACM, 2013.

[19] A. Piad-Morffis, S. Estevez-Velarde, A. Bolufe-Rohler, J. Montgomery, and S. Chen. Evolution Strategies with Thresheld Convergence. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2097–2104, May 2015.

[20] I. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86 – 112, 1967.

[21] S. van Rijn, H. Wang, and T. Bäck. Evolving Optimal Evolution Strategies. In *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, page *Under Submission.* IEEE, 2016.

[22] H. Wang, M. Emmerich, and T. Bäck. Mirrored Orthogonal Sampling with Pairwise Selection in Evolution Strategies. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 154–156. ACM, 2014.

[23] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.