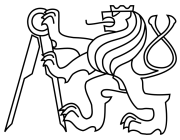
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Selection of surrogate models for evolutionary black-box optimization in noisy environment
Student:	Bc. Vojtěch Hejl
Supervisor:	doc. Ing. RNDr. Martin Holeňa, CSc.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

Selection of surrogate models for evolutionary black-box optimization in noisy environment.

1. Get acquaianted with the covariance matrix adaptation evolution strategy (CMA-ES) for black-box optimization and with possibilities to use surragate models in its context.

2. Get acquaianted with noisy benchmark functions for black-box optimization available in the black-box optimization benchmarking (BBOB) platform.

3. Test 4 from surrogate models proposed in the literature, including 2 models based on Gaussian processes, with the BBOB noisy benchmarks.

4. Propose an algorithm for simultaneous testing various models and their variants, selecting the model according to RMSE.

5. Elaborate several alternative possibilities of model selection, preferably alternatives based on the invariance of CMA-ES with respect to monotonous transformations.

6. Compare the investigated alternatives of model selection using the BBOB noisy benchmarks.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 7, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Master's thesis

Selection of surrogate models for evolutionary black-box optimization in noisy environment

Bc. Vojtěch Hejl

Supervisor: doc. Ing. RNDr. Martin Holeňa, CSc.

8th January 2018

Acknowledgements

My thanks are first of all to God for his mercy, and then to all those whom he has sent me in my way. I would like to thank my supervisor, Martin Holeňa, for his professional guidance, valuable advice and expressed patience, and also to Lukáš Bajer for his help in getting to know the project. Last but not least, I also want to thank my family and friends for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on 8th January 2018 .....

Czech Technical University in Prague
Faculty of Information Technology
© 2018 Vojtěch Hejl. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Hejl, Vojtěch. *Selection of surrogate models for evolutionary black-box optimization in noisy environment.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Tato práce navazuje na výzkum L. Bajera, Z. Pitry, J. Repického a M. Holeňi o náhradním modelování v algoritmu CMA-ES. Cílem této práce bylo ověřit kvalitu současného návrhu a poté navrhnout varianty s testováním více modelů. Hlavním přínosem této práce je návrh evoluce náhradních modelů v algoritmu CMA-ES, který má velký potenciál pro vylepšení.

*****. Klíčová slova black-box optimalizace, CMA-ES, zašumněné funkce, náhradní modely, Gaussovský proces, evoluce

Abstract

This diploma thesis draws on the research by L. Bajer, Z. Pitra, J. Repický and M. Holeňa, which deals with the surrogate modeling in the CMA-ES algorithm. The aim of this work was to verify the quality of the current design and then to propose the variants with testing of multiple models. The main contribution of this thesis is the proposal of the evolution of surrogate models in the CMA-ES algorithm, which has a great improvement potential.

*****. Keywords black-box optimization, CMA-ES, noisy environment, surrogate models, Gaussian, process, evolution

Contents

List of Figures

List of Tables

Nomenclature

I used notation, where bold letters, $\mathbf{v}$, are vectors, capital bold letters, $\mathbf{A}$, are matrices, and a transpose is denoted by $\mathbf{v}^T$

E expectation value.

$\mathcal{N}(\mathbf{0}, \mathbf{I})$ multivariate normal distribution with zero mean and unity covariance matrix. A vector distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$ has independent, (0,1)-normally distributed components.

$\mathcal{N}(\mathbf{m}, \mathbf{C}) \sim \mathbf{m} + \mathcal{N}(\mathbf{0}, \mathbf{C})$ multivariate normal distribution with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance matrix $\mathbf{C} \in \mathbb{R}^{n\ timesn}$. The matrix C is symmetric and positive definite.

$\sigma^{(g)} \in \mathbb{R}_+$ step-size.

$\mathbf{B} \in \mathbb{R}^n$ an orthogonal matrix. Columns of $\mathbf{B}$ are eigenvectors of $\mathbf{C}$.

$\mathbf{C}^{(g)} \in \mathbb{R}^{n \times n}$ covariance matrix at generation g.

$\mathbf{D} \in \mathbb{R}^n$ a diagonal matrix. The diagonal elements of $\mathbf{D}$ are square root of eigenvalues of $\mathbf{C}$ and correspond to the respective columns of $\mathbf{B}$.

$\mathbf{I} \in \mathbb{R}^{n \times n}$ identity matrix

$\mathbf{p} \in \mathbb{R}^n$, evolution path, a sequence of successive (normalized) steps.

$g \in \mathbb{N}_0$ generation counter, iteration number.

CMA-ES Covariance Matrix Adaptation Evolution Strategy

DTS-CMA-ES Double Trained CMA-ES

GP Gaussian process

Introduction

"There is only one good, knowledge, and one evil, ignorance."
(Socrates)

In past, knowledge was very important and hard to obtain. Only a few people could read, books were expensive and rare. Ancient philosophers knew the value of knowledge. Many people people seek them to learn from them. Today we have knowledge (better data) more than enough. Today's struggle is how to process it or find desired information. Therefore, we need better way to do so. The solution is optimization.

Optimization is process of finding an alternative with the most cost effective or highest achievable performance under the given constraints, by maximizing desired factors and minimizing undesired ones. In comparison, maximization means trying to attain the highest or maximum result or outcome without regard to cost or expense. Practice of optimization is restricted by the lack of full information, and the lack of time to evaluate what information is available. In computer simulation (modeling) of business problems, optimization is achieved usually by using linear programming techniques of operations research[1].

Optimization is not only about processing of finding information or data. On the contrary, it is just beginning. In each group of data we can observe some pattern. This data corresponds to an unknown function. This function can be estimated by using some approximation function that fits given data points. Another important information is that even this unknown function have global optimum, considering that function is continuous. This kind of optimization is called black-box optimization (see 1.2).

The collected data is mostly inaccurate and contains noise, so used algorithm should be robust and should handle noise with ease. Otherwise, it may stuck in local optimum and return false results. Algorithm CMA-ES used in this thesis is believed to handle even highly multi-modal functions with severe noise. One subtask is validate it.

Even best optimization algorithm may be cost ineffective, if obtain data points if expensive and time consuming. In this case, a surrogate model is used. This approach consist of two steps. First step is approximate original function with simpler function, using only a few original evaluated data points. After that optimalization can be used. If error rate is higher than given treshold, same points should be evaluated with original function and surrogate model retrained.

According to the assignment, the following questions arise and will be answered in conclusion:

- What suggest for the future ?
- Which alternative of CMA-ES is better ?
- Can CMA-ES handle noisy functions ?

Chapter 1

Preliminaries

The aim of this chapter is to explain a several important concepts needed to understand this thesis.

1.1 Evolution Strategy

An evolution strategy in computer science, simulates biological evolution, I will start with explaining some definitions.*An individual* (also called phenotype) is one particular solution. The set of all considered solutions is known as their *population.* The evolution is an iterative process, with the population in each iteration called a *generation. A chromozome* (also called genotype)) is a set of parameters which define a proposed solution to the problem that the evolution strategy is trying to solve (for example, binary array).

Evolution strategy is an optimalization technique based on evolution and natural selection. Candidate solutions are evaluated and only the best ones survives. New candidates are created by recombination from the previous generation. Evolution strategy has a potential to solve black-box optimalization problems, because it needs only information about the quality of evaluated solutions. Evolution strategies use natural problem-dependent representations, and primarily mutation and selection, as search operators. In common with other kinds of evolutionary algorithms, the operators are applied in a loop. An iteration of the loop is called a generation. The sequence of generations is continued until a termination criterion is met (e.g, the number of generations, or some quality indicator of the solution)[2].

The simplest evolution strategy operates on a population of size two: the current point (parent) and the result of its mutation. Only if the mutant's fitness is at least as good as the parent one, it replaces the parent of the next generation. Otherwise the mutant is disregarded. This is called a $(1+1)-ES$. More generally, λ mutants can be generated and compete with the parent, called $(1+\lambda)-ES$. In $(1,\lambda)-ES$ the best mutant becomes the parent of the next generation while the current parent is always disregarded. For some of

these variants, proofs of linear convergence (in a stochastic sense) have been derived on unimodal objective functions. Contemporary kinds of evolution strategy often use a population of μ parents and also recombination as an additional operator, expressed with the notation $(\mu/\rho+, \lambda) - ES$. This makes them less prone to get stuck in local optima.

1.1.1 Selection

The selection in evolution strategies considered in this thesis is deterministic and only based on the fitness rankings, not on the actual fitness values. The resulting algorithm is, therefore, invariant with respect to monotonic transformations of the objective function. Selection is used to choose individuals with high fitness value and go straight to an optimal solution, while preserving diversity to avoid stucking in local optima. There are several variants of selection[3]:

- Elitism: choose the best λ individuals to the next generation. Sometimes good candidates can be lost when the crossover or mutation results in offsprings that are weaker than the parents. Often the EA will re-discover these lost improvements in a subsequent generation, but there is no guarantee. To hinder this, we can use an approach known as elitism. Elitism involves copying a small propotion of the fittest candidates, unchanged, into the next generation. This can sometimes have a dramatic impact on performance by ensuring that the EA does not waste time re-discovering previously discarded partial solutions. Candidate solutions that are preserved unchanged through elitism remain eligible for selection as parents when breeding the remainder of the next generation[4].

- Roulette: the probability of selecting an individual depends on its fitness. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number valued in the union of all these segments is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals, called mating population, is obtained. This technique can be illustrated as a roulette wheel with each slice proportional in size to the fitness.

  Table 1.1 shows the selection probability for eight individuals. The individual number one is the most fit individual and occupies the largest interval, whereas the individual number eight as the least fit individual has the smallest interval. For selecting the mating population, an appropriate number of uniformly distributed random numbers (between 0.0 and 1.0) is independently generated.

- Tournament: a tournament selection involves running several "tournaments" among a few individuals chosen at random from the population.

individual	1	2	3	4	5	6	7	8
Fitness value	3.0	1.75	1.5	1.25	1.0	0.75	0.5	0.25
Select. prob.	0.3	0.175	0.15	0.125	0.01	0.075	0.05	0.025

Table 1.1: Roulette selection example

The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. Each tournament has the following steps:

1) choose k (tournament size) individuals from the population at random
2) choose the best individual from the participants of this tournament with probability p
3) choose the second best individual with probability $p * (1 - p)$
4) choose the third best individual with probability $p * ((1 - p)^2)$

$k + 2$) continue with step 1 until new population is selected.

1.1.2 Crossover

The crossover operator is analogous to reproduction and biological crossover. In this approach, more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. In this section, we will discuss some of the most popularly used crossover operators. It should be noted that these crossover operators are very generic and the designer might choose to implement a problem-specific crossover operator as well.

- In the one-point crossover, a random crossover point is selected and the tails of the two parents are swapped to get new off-springs.

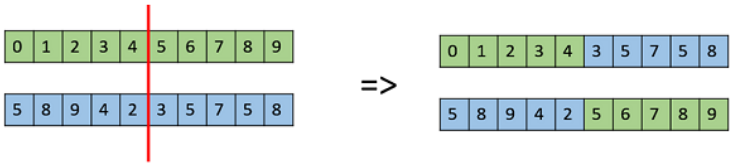


Figure 1.1: one-point crossover

- Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-spring

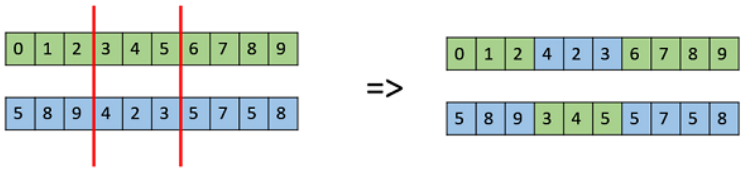


Figure 1.2: multi-point crossover

- In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this kind of crossover, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

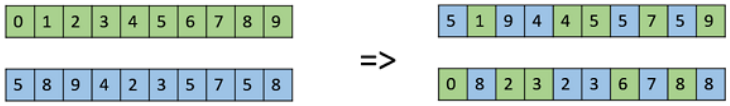


Figure 1.3: uniform crossover

1.1.3 Mutation

As far as real-valued search spaces are concerned, mutation is usually performed by adding a normally distributed random value to each vector component. Both the step size and the mutation strength (i.e. the standard deviation of the normal distribution) is often governed by self-adaptation. Individual step sizes for each coordinate or correlations between coordinates are either governed by self-adaptation or by covariance matrix adaptation. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. A mutation alters one or more gene values in a chromosome from its initial state. In a mutation, the new individuals may differ entirely from individuals in the previous generation. Hence the evolution can come to a better solution by using mutation. Mutation occurs during the evolution according to a user-definable mutation probability. This probability should be set low, otherwise the search will turn into a primitive random search. A classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing a mutation operator involves generating a random variable for each chromosome in a sequence. This random variable determines whether or not a particular chromosome will be modified. This mutation procedure, based on biological mutation, is called single point muta-

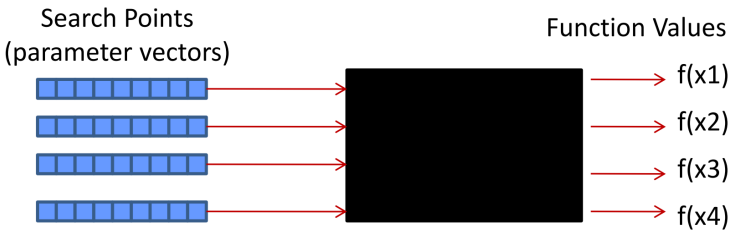


Figure 1.4: Schema of black-box optimalization

tion. Other types are inversion and floating point mutation. The purpose of mutation in the evolution is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing down or even stopping evolution. This reasoning also explains the fact that most evolution strategies avoid taking only the fittest of the population, but they use rather a random (or semi-random) selection with a bias toward those that are fitter[5].

1.2 Black-box optimalization

Black-box optimization refers to a problem setup in which an optimization algorithm is supposed to optimize (e.g., minimize) an objective function f through a so-called black-box interface: the algorithm may query the value $f(x)$ for a point x, but it does not obtain gradient information, and in particular it cannot make any assumptions on the analytic form of f (e.g., being linear or quadratic). We think of such an objective function as being wrapped in a black-box. The goal of optimization is to find an as good as possible value $f(x)$ within a predefined time, often defined by the number of available queries to the black box. Problems of this type regularly appear in practice, e.g., when optimizing parameters of a model that is either in fact hidden in a black box (e.g., a third party software library) or just too complex to be modeled explicitly[6].

A large variety of optimization algorithms for continuous black box optimization has been proposed. The power of these algorithm is usually assessed on collections of benchmark problems. This is an important approach to comparability of scientific results. However, it means that the problem to be optimized is known to the experimenter, and hence it is not truly a black box.

$$f : \mathbb{R}^n \to \mathbb{R} x \to f(x)$$

1.2.1 Benchmarking

A platform for systematic and sound comparisons of real-parameter global optimisers, COmparing Continuous Optimisers (COCO), is used in this work. The COCO provides benchmark function testbeds, experimentation templates which are easy to parallelize, and tools for processing and visualizing data generated by one or several optimizers. The COCO platform has been used for the Black-Box-Optimization-Benchmarking (BBOB) workshops that took place during the GECCO conference in years 2009, 2010, 2012, 2013, 2015, 2016 and 2017. The COCO uses several noisy function in benchmarking, in this work are considered the following[7]:

- functions with moderate noise
  - f102 Sphere with moderate uniform noise
  - f104 Rosenbrock with moderate Gaussian noise
- functions with severe noise
  - f107 Sphere with Gaussian noise
  - f111 Rosenbrock with uniform noise
  - f117 Ellipsoid with uniform noise
- highly multi-modal functions with severe noise
  - f122 Schaffer's F7 with Gaussian noise
  - f123 Schaffer's F7 with uniform noise
  - f129 Gallagher's Gaussian Peaks 101-me with uniform noise (The function consists of 101 optima with position and height being unrelated and randomly chosen.)

1.3 Surrogate models

Surrogate models, or metamodels, are compact scalable analytic models that approximate the multivariate input/output behavior of complex systems, based on a limited set of computationally expensive simulations or measurements. Surrogate models mimic the complex behavior of the underlying simulation model, and can be used for design automation, parametric studies, design space exploration, optimization and sensitivity analysis. Surrogate modeling is a supervised Machine Learning (ML) technique [8].

Parameterized surrogate models are increasingly important as a means of exploring the behavior of complex systems, and for sensitivity analysis and optimization. Application areas include bioinformatics, network simulation, Electronic Design Automation (EDA), ecological modeling, and many others.

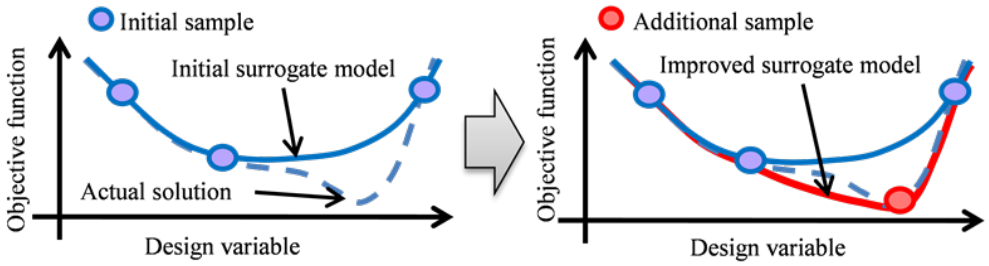


Figure 1.5: Improvement of surrogate model with additional sample

For example, an engineer may explore the behavior of an airplane wing by running a computational model of the air flow multiple times while varying key design parameters such as length, angle, etc. The results of these multiple experiments yield a picture of how the wing behaves in different parts of the design space. Thorough understanding of the relationship between design parameters and performance over the entire design space is important for optimal design, to reduce the number of design iterations, to lower the costs, and to improve overall quality. Usually, for parametric computational experiments, the user selects a huge amount of simulation experiments, covering the whole design space of interest, by defining all parameter ranges and sample distributions (min, max, step size). Note that the cross product of the parameter ranges does not guarantee an accurate and efficient coverage of the design space, as certain regions might be under-sampled while others might be over-sampled. The scientific challenge of surrogate modeling is the generation of a surrogate that is as accurate as possible, using as few simulation evaluations as possible. The process comprises three major steps which may be interleaved iteratively:

1) Sample selection (also known as sequential design, optimal experimental design (OED) or active learning)

2) Construction of the surrogate model and optimizing the model parameters (bias–variance trade-off)

3) Appraisal of the accuracy of the surrogate.

The accuracy of the surrogate depends on the number and location of samples (expensive experiments or simulations) in the design space. Various design of experiments (DOE) techniques cater to different sources of errors, in particular errors due to noise in the data or errors due to an improper surrogate model.

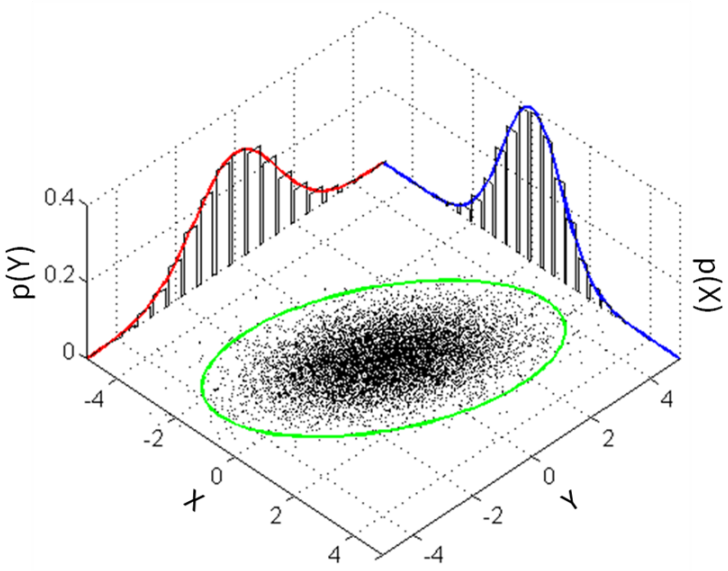


Figure 1.6: Samples of 2D normal distribution.

1.4 The Multivariate Normal Distribution

A multivariate normal distribution, $\mathcal{N}(\mathbf{m}, \mathbf{C})$, has a unimodal, "bell-shaped" density, where the top of the bell (the modal value) corresponds to the distribution mean, $\mathbf{m}$. The distribution $\mathcal{N}(\mathbf{m}, \mathbf{C})$ is uniquely determined by its mean $\mathbf{m} \in \mathbb{R}^n$ and its symmetric and positive definite covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. Covariance (positive definite) matrices have an appealing geometrical interpretation: they can be uniquely identified with the (hyper-)ellipsoid $\mathbf{x} \in \mathbb{R}^n | \mathbf{x}^T \mathbf{C}^{-1} \mathbf{x} = 1$, as shown in fig. 1.6. The ellipsoid is a surface of equal density of the distribution. The principal axes of the ellipsoid correspond to the directions of eigenvectors of C, the squared axes lengths correspond to the eigenvalues. The eigendecomposition of $\mathbf{C}$ is

$$\mathbf{C} = \mathbf{B}(\mathbf{D})^2 \mathbf{B}^T \tag{1.1}$$

where

$\mathbf{B}^2$ is an orthogonal matrix, $\mathbf{B}^T\mathbf{B} = \mathbf{B}\mathbf{B}^T = \mathbf{I}$. Both columns and rows of B form an orthonormal basis of eigenvectors.

$\mathbf{D}^2 = \mathbf{D}\mathbf{D} = \text{ diag}(d_1, \ldots, d_n)^2 = \text{ diag}(d_1^2, \ldots, d_n^2)$ is a diagonal matrix with eigenvalues of $\mathbf{C}$ as diagonal elements.

$D = \text{ diag}(d_1, \ldots, d_n)$ is a diagonal matrix with square roots of eigenvalues of C as diagonal elements.

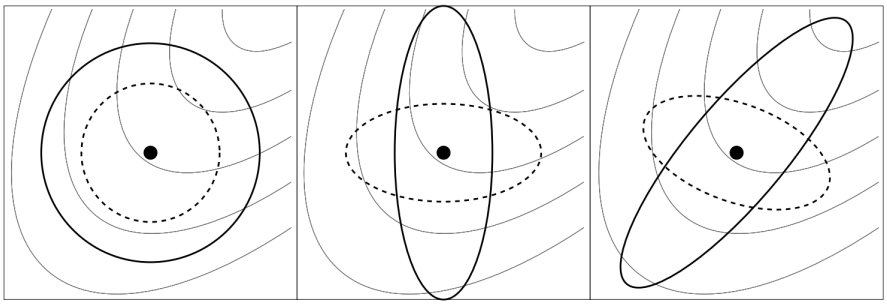


Figure 1.7: Ellipsoids depicting one-σ lines of equal density of six different normal distributions, where $\sigma \in R_+$, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{C}$ is a positive definite full covariance matrix. Thin lines depict possible objective function contour lines.

If $\mathbf{D} = \sigma\mathbf{I}$, where $\sigma \in \mathbb{R}_+$ and $\mathbf{I}$ denotes the identity matrix, $\mathbf{C} = \sigma^2\mathbf{I}$ and the ellipsoid is isotropic 1.7, left). If $\mathbf{B} = \mathbf{I}$, then $\mathbf{C} = \mathbf{D}^2$ is a diagonal matrix and the ellipsoid is axis parallel oriented (middle). Therefore, in the coordinate system given by the columns of $\mathbf{B}$, the distribution , $\mathcal{N}(\mathbf{m}, \mathbf{C})$ is always uncorrelated. The normal distribution, $\mathcal{N}(\mathbf{m}, \mathbf{C})$ can be written in different ways [9].

$$
\begin{aligned}
\mathcal{N}(\mathbf{m}, \mathbf{C}) &= \mathbf{m} + \mathcal{N}(\mathbf{m}, \mathbf{C}) \\
&= \mathbf{m} + \mathbf{C}^{\frac{1}{2}}\mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \mathbf{m} + \mathbf{B}\mathbf{D}\underbrace{\mathbf{B}^T\mathcal{N}(\mathbf{0}, \mathbf{I})}_{=\mathcal{N}(\mathbf{0}, \mathbf{I})} \\
&= \mathbf{m} + \mathbf{B}\underbrace{\mathbf{D}\mathcal{N}(\mathbf{0}, \mathbf{I})}_{=\mathcal{N}(\mathbf{0}, \mathbf{D}^2)}
\end{aligned}
$$

where $\mathbf{C}^{\frac{1}{2}} = \mathbf{B}\mathbf{D}\mathbf{B}^T$. The last row can be well interpreted, from right to left

- $\mathcal{N}(\mathbf{0}, \mathbf{I})$ produces an spherical (isotropic) distribution as in 1.7, left.
- $\mathbf{D}$ scales the spherical distribution within the coordinate axes as in 1.7, middle. $\mathcal{N}(\mathbf{0}, \mathbf{I}) = \mathcal{N}(\mathbf{0}, \mathbf{D}^2)$ has n indepentent components. The matrix $\mathbf{D}$ can be interpreted as (individual) step-size matrix and its diagonal entries are the standard deviations of the compoments.

- **B** defines a new orientation for the ellipsoin, where the new principal axes of the ellipsoin correspond to the columns of **B**. Note that B has $\frac{n^2-n}{2}$ degrees of freedom [9].

The equation above is useful to compute $\mathcal{N}(\mathbf{m}, \mathbf{C})$ distributed vectors, because $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is a vector of independent (0,1)-normally distributed numbers that can easily be sampled.

1.5 CMA-ES

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a popular algorithm in continuous global optimization. The CMA-ES relies on an iterative updating of covariance matrix, which determines relationships between the search point $(x_1, x_2 \dots x_n)$ and the output value $f(x)$ as described in 1.2. Continuous atributes in the input vector are typicaly distributed acording to normal density. The number of atributes determines the dimension of the multivariate normal distribution. A new population of sampling points for the generation number $g = 0,1,2 \dots$ is distributed as

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)} \mathcal{N}\left(\mathbf{0}, \mathbf{C}^{(g)}\right) \quad \text{for } k = 1, \dots, \lambda \tag{1.2}$$

where

$\sim$ denotes the same distribution on the left and right side.

$\mathcal{N}(\mathbf{0}, \mathbf{C}^{(g)})$ is a multivariate normal distribution with zero mean and covariance matrix $\mathbf{C}^{(g)}$.

$\mathbf{x}_k^{(g+1)} \in \mathbb{R}^n$, k-th offspring (individual, search point) from generation g+1.

$\mathbf{m}^{(g)} \in \mathbb{R}^n$, mean value of the search distribution at generation g.

$\sigma^{(g)} \in \mathbb{R}_+$, step-size, at generation g.

$\mathbf{C}^{(g)} \in \mathbb{R}^{n \times m}$,up to the scalar factor $\sigma^{(g)^2}$, $\mathbf{C}^{(g)}$ is the covariance matrix of the search distribution at generation g.

$\lambda \geq 2$, population size, sample size, number of offspring.

CMA-ES is sometimes interpreted as a robust local search method. Its robustness is related to invariance properties with respect to objective function scaling and coordinate system rotations. This algorithm was consistently

Algorithm 1: CMA-ES (simplified pseudocode)[10]

Input : original fitness function f, step-size $\sigma^{(0)} \in \mathbb{R}_+$, initial mean $\mathbf{m}^{(0)} \in \mathbb{R}^D$

Output: $\hat{\mathbf{x}}^{\text{opt}}$ – point with the minimum achieved fitness

set the population size λ, μ, $w_{1,\ldots,\mu}$ and other parameters $(c_\sigma, d_\sigma, c_\text{c}, \mu_\text{cov}, c_\text{cov})$ to default values (for λ and μ, defaults are $\lambda = 4 + \lfloor 3 \ln D \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$)

initialize the evolution path $\mathbf{p}_\sigma^{(0)} = \mathbf{0}$, $\mathbf{p}_\text{c}^{(0)} = \mathbf{0}$, covariance matrix $\mathbf{C}^{(0)} = \mathbf{I}$

for *generation* $g = 0, 1, 2, \ldots$ *until stopping conditions met* **do**

  $\mathbf{x}_k \sim \mathcal{N}(\mathbf{m}^{(g)}, (\sigma^{(g)})^2 \mathbf{C}^{(g)})$ for $k = 1, \ldots, \lambda$ */* sample a new population */*

  sorted $\mathbf{x}_{1:\lambda} \leftarrow f$-evaluate all $\mathbf{x}_1, \ldots, \mathbf{x}_\lambda$ */* fitness evaluation */*

  $\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$ */* selection and recombination */*

  $\mathbf{p}_\sigma^{(g+1)} \leftarrow$ aggregate the $(\sigma^{(g)})^2\mathbf{C}$-normalized difference of means $(\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)})/(\sigma^{(g)}\sqrt{\mathbf{C}^{(g)}})$ into the evolution path $\mathbf{p}_\sigma^{(g)}$

  $\sigma^{(g+1)} \leftarrow$ update the step-size according to the length $\|\mathbf{p}_\sigma^{(g+1)}\|$

  $\mathbf{p}_\text{c}^{(g+1)} \leftarrow$ aggregate the $\sigma^{(g)}$-normalized difference of means $(\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^{(g)}$ into the evolution path $\mathbf{p}_\text{c}^{(g)}$

  $\mathbf{C}^{(g+1)} \leftarrow$ perform the rank-one update (based on $\mathbf{p}_\text{c}^{(g+1)}$) and rank-$\mu$ update (based on differences $(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)}),\ i = 1, \ldots, \mu$) of $\mathbf{C}^{(g)}$

end

found to be highly performing in the Black-Box Optimization Benchmarking (BBOB) workshops for low, moderate and highly multimodal functions of dimensions between 5 and 40 if it is coupled with a restart mechanism [9].

1.5.1 Moving the mean

Mean in next generation is weighted average of best μ points from candidate solutions $x_1^{(g+1)}, \ldots, x_\lambda^{(g+1)}$

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)} \tag{1.3}$$

$$\sum_{i=1}^{\mu} w_i = 1, w_1 \geq w_2 \geq \cdots \geq w_\mu > 0 \tag{1.4}$$

where

$\mu \leq \lambda$ is the parent population size, *i.e.* the number of selected points.

$w_{i...\mu} \in R_+$ positive weight coefficients for recombination. For $w_{i=i...\mu} = 1/\mu$ Equation 1.3 calculates the mean value of μ selected points.

$x_{i:\lambda}^{(g+1)}$, i-th best individual out of $x_1^{(g+1)},\ldots,x_\lambda^{(g+1)}$. The index $i : \lambda$ detones the index of the i-th ranked individual ad $f(x_{1:\lambda}^{(g+1)}) \leq f(x_{2:\lambda}^{(g+1)}) \leq \cdots \leq f(x_{\lambda\lambda}^{(g+1)})$, where f is the objective function to be minimized.

Equation 1.3 implements truncation selection by choosing $\mu < \lambda$ out of λ offspring points. Assigning different weights w_i must also be interpreted as a selection mechanism. Equation 1.3 implements weighted intermediate recombination by taking $\mu > 1$ individuals into account for a weighted average.

The measure

$$\mu_{\text{eff}} = \left(\frac{\|\mathbf{w}\|_1}{\|\mathbf{w}\|_2}\right)^2 = \frac{\|\mathbf{w}\|_1^2}{\|\mathbf{w}\|_2^2} = \frac{1}{\|\mathbf{w}\|_2^2} = \left(\sum_{i=1}^{\mu} w_i^2\right)^{-1} \tag{1.5}$$

will be repeatedly used in the following and can be paraphrased as *variance effective selection mass*. From the definition of w_i in 1.4 we derive $1 \leq \mu_{\text{eff}} \leq \mu$, and $\mu_{\text{eff}} = \mu$ for equal recombination weights, i.e. $w_i = 1/\mu$ for all $i = 1 \ldots \mu$. Usually, $\mu_{\text{eff}} \approx \lambda/4$ indicates a reasonable setting of w_i. A typical setting could be $w_i \propto \mu - i - 1$, and $\mu \approx \lambda/2$.

1.5.2 Adapting the covariance matrix

In this section, the update of the covariance matrix, $\mathbf{C}$, is derived. We will start out estimating the covariance matrix from a single population of one generation (Sect. 1.5.2.1). For small populations this estimation is unreliable and an adaptation procedure has to be invented (rank-μ-update, Sect. 1.5.2.2). In the limit case only a single point can be used to update (adapt) the covariance matrix at each generation (rank-one-update, Sect. 1.5.2.3). The adaptation can be enhanced by exploiting dependencies between successive steps applying cumulation (Sect. 3.3.2). Finally we combine the rank-μ and rank-one updating methods (Sect. 3.4).

1.5.2.1 Estimating the covariance matrix from scratch

For the moment we assume that the population contains enough information to reliably estimate a covariance matrix from the population. We can (re-)estimate the original covariance matrix $C^{(g)}$ using the sampled popula-

tion, $x_1^{(g+1)} \dots x_\lambda^{(g+1)}$. To "estimate" a "better" covariance matrix, weighted selection mechanism is used.

$$\mathbf{C}_\mu^{(g+1)} = \sum_{i=1}^{\mu} w_i \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)} \right) \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)} \right)^T \tag{1.6}$$

The matrix $\mathbf{C}_\mu^{(g+1)}$ is an estimator for the distribution of selected steps. Sampling from $\mathbf{C}_\mu^{(g+1)}$ tends to reproduce selected, i.e. successful steps, giving a justification for what a "better" covariance matrix means. In order to ensure, that $\mathbf{C}_\mu^{(g-1)}$ is a reliable estimator, the variance effective selection mass μ_{eff} must be large enough.

1.5.2.2 Rank-μ-update

To achieve fast search (opposite to more robust or more global search), the population size λ must be small. Because $\mu_{\text{eff}} \approx \lambda/4$ also μ_{eff} must be small and we may assume, e.g., $\mu_{\text{eff}} \leq 1 + \ln n$. Then, it is not possible to get a reliable estimator for a good covariance matrix from scratch, as was in previous section. As a remedy, information from previous generations is used additionally. For example, after a sufficient number of generations, the mean of the estimated covariance matrices from all generations,

$$\mathbf{C}^{(g+1)} = \frac{1}{g+1} \sum_{g+1}^{g} \frac{1}{\sigma^{(i)^2}} \mathbf{C}_\mu^{(i+1)} \tag{1.7}$$

becomes a reliable estimator for the selected steps. To make $C_\mu^{(g)}$ from different generations comparable, the different $\sigma^{(i)}$ are incorporated. In equation above, all generation steps have the same weight. To assign recent generations a higher weight, exponential smoothing is introduced. Choosing $C^{(0)} = I$ to be the unity matrix and a learning rate $0 < c_\mu \leq 1$, then $C^{(g+1)}$ reads

$$\begin{aligned}
\mathbf{C}^{(g+1)} &= (1 - c_\mu)\mathbf{C}^{(g)} + c_\mu \frac{1}{\sigma^{(g)^2}} \mathbf{C}_\mu^{(g+1)} && (1.8)\\
&= (1 - c_\mu)\mathbf{C}^{(g)} + c_\mu \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{(g+1)} \mathbf{y}_{i:\lambda}^{(g+1)^T} && (1.9)\\
&= \mathbf{C}^{(g)^{1/2}} \left(\mathbf{I} + c_\mu \sum_{i=1}^{\mu} w_i \left(\mathbf{z}_{i:\lambda}^{(g+1)} \mathbf{z}_{i:\lambda}^{(g+1)^T} - \mathbf{I} \right) \right) \mathbf{C}^{(g)^{1/2}} && (1.10)
\end{aligned}$$

where

$c_\mu \leq 1$ learning rate for updating the covariance matrix. For $c_\mu = 1$, no prior information is retained and $\mathbf{C}^{(g+1)} = \frac{1}{\sigma^{(g)^2}} \mathbf{C}_\mu^{(g+1)}$ For $c_\mu = 0$, no learning takes place and $\mathbf{C}^{(g+1)} = \mathbf{C}^{(0)}$. Here, $c_\mu \approx \min(1, \mu_{\text{eff}}/n^2)$ is a reasonably

choice.

$\mathbf{y}_{i:\lambda}^{(g+1)} = (\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^{(g)}$.

$\mathbf{z}_{i:\lambda}^{(g+1)} = \mathbf{C}^{(g)^{-1/2}}\mathbf{y}_{i:\lambda}^{(g+1)}$ is the mutation vector expressed in the unique coordinate system where the sampling is isotropic and the respective coordinate system transformation does not rotate the original principal axes of the distribution.

This covariance matrix update is called rank-μ-update, because the sum of outer products is of rank $\min(\mu, n)$ The choice of c_μ is crucial. Small values lead to slow learning, too large values lead to a failure, because the covariance matrix degenerates. Fortunately, a good setting seems to be largely independent of the function to be optimized. A first order approximation for a good choice is $c_\mu \approx \mu_{\text{eff}}/n^2$. Therefore, the characteristic time horizon is roughly n^2/μ_{eff} . Even for the learning rate $c_m u = 1$, adapting the covariance matrix cannot be accomplished within one generation. The effect of the original sample distribution does not vanish until a sufficient number of generations. Assuming fixed search costs (number of function evaluations), a small population size λ allows a larger number of generations and therefore usually leads to a faster adaptation of the covariance matrix.

1.5.2.3 Rank-one-update

In Section 1.5.2.1 we estimated the complete covariance matrix from scratch, using all selected steps from a single generation. We now take precisely the opposite viewpoint. We will repeatedly update the covariance matrix in the generation sequence using a single selected step only. First, this perspective will give a justification of the adaptation rule in previous section. Second, we will introduce the so-called evolution path that is finally used for a rank-one update of the covariance matrix.

A Different Viewpoint

We consider a specific method to produce n-dimensional normal distributions with zero mean. Let the vectors $y_1, \dots y_{g_0} \in R^n, g_0 \geq n$, span R^n and let $N(0,1)$ denote independent (0, 1)-normally distributed random numbers, then

$$\mathcal{N}(0,1)\mathbf{y}_1 + \cdots + \mathcal{N}(0,1)\mathbf{y}_{g_0} \sim \mathcal{N}\left(\mathbf{0}, \sum_{i=1}^{g_0} \mathbf{y}_i\mathbf{y}_i^T\right) \tag{1.11}$$

is a normally distributed random vector with zero mean and covariance matrix $\sum_{i=1}^{g_0} \mathbf{y}_i\mathbf{y}_i^T$ The random vector is generated by adding "line-distribution" $\mathcal{N}(0,1)\mathbf{y}_i$. The singular distribution $\mathcal{N}(0,1)\mathbf{y}_i \sim \mathcal{N}(0, \mathbf{y}_i\mathbf{y}_i^T)$ generates the vector $\mathbf{y}_i$ with maximum likelihood considering all normal distributions with zero mean.

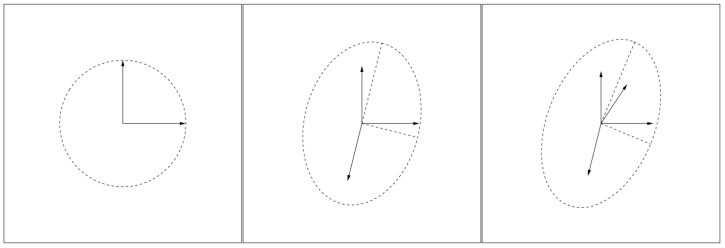


Figure 1.8: Change of the distribution according to the covariance matrix update. Left: vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, and $\mathbf{C}^{(0)} = \mathbf{I} = \mathbf{e}_1\mathbf{e}_1^T + \mathbf{e}_2\mathbf{e}_2^T$. Middle: vectors $0.91\mathbf{e}_1, 0.91\mathbf{e}_2$, and $0.41\mathbf{y}_1$ (the coefficients deduce from $c_1 = 0.17$), and $\mathbf{C}^{(1)} = (1-c_1)I + c_1\mathbf{y}_1\mathbf{y}_1^T$, where $y_1 = \left(\begin{smallmatrix}-0.59\\-2.2\end{smallmatrix}\right)$. The distribution ellipsoid is elongated into the direction of $\mathbf{y}_1$, and therefore increases the likelihood of $\mathbf{y}_1$. Right: $\mathbf{C}^{(2)} = (1-c_1)\mathbf{C}^{(1)} + c_1\mathbf{y}_2\mathbf{y}_2^T$, where $\mathbf{y}_2 = \left(\begin{smallmatrix}0.97\\1.5\end{smallmatrix}\right)$.

Considering previous distribution and a slight simplification of rank-μ-update, we try to gain insight into the adaptation rule for the covariance matrix. Let the sum in rank-μ-update consist of a single summand only (e.g. $\mu = 1$), and let $\mathbf{y}_{g+1} = \frac{\mathbf{x}_{1:\lambda}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}}$. Then, the rank-one update for the covariance matrix reads

$$\mathbf{C}^{(g+1)} = (1-c_1)\mathbf{C}^{(g)} + c_1\mathbf{y}_{g+1} + \mathbf{y}_{g+1}^T \tag{1.12}$$

The right summand is of rank one and adds the maximum likelihood term for $\mathbf{y}_{g+1}$ into the covariance matrix $\mathbf{C}^{(g)}$. Therefore the probability to generate $\mathbf{y}_{g+1}$ in the next generation increases.

An example of the first two iteration steps is shown in Figure 1.8. The distribution $\mathcal{N}(\mathbf{0}, \mathbf{C}^{(1)})$ tends to reproduce y_1 with a larger probability than the initial distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, the distribution $\mathcal{N}(\mathbf{0}, \mathbf{C}^{(2)})$ tends to reproduce $\mathbf{y}_2$ with a larger probability than $\mathcal{N}(\mathbf{0}, \mathbf{C}^{(1)})$, and so forth. When $\mathbf{y}_1, \ldots, \mathbf{y}_g$ denote the formerly selected, favorable steps, $\mathcal{N}(\mathbf{0}, \mathbf{C}^{(g)})$ tends to reproduce these steps. The process leads to an alignment of the search distribution $\mathcal{N}(\mathbf{0}, \mathbf{C}^{(g)})$ to the distribution of the selected steps. If both distributions become alike, as under random selection, in expectation no further change of the covariance matrix takes place.

Cumulation: Utilizing the Evolution Path

We have used the selected steps, $\mathbf{y}_{i:\lambda}^{(g+1)} = (\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)})/\sigma(g)$, to update the covariance matrix in rank-μ-update. We call a sequence of successive steps, the strategy takes over a number of generations, an evolution path. An evolution path can be expressed by a sum of consecutive steps. This summation is referred to as cumulation. To construct an evolution path, the

step-size σ is disregarded. For example, an evolution path of three steps of the distribution mean m can be constructed by the sum

$$\frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}} + \frac{\mathbf{m}^{(g)} - \mathbf{m}^{(g-1)}}{\sigma^{(g-1)}} + \frac{\mathbf{m}^{(g-1)} - \mathbf{m}^{(g-2)}}{\sigma^{(g-2)}} \tag{1.13}$$

In practice, to construct the evolution path, $\mathbf{p}_c \in R^n$, we use exponential smoothing as in (14), and start with $\mathbf{p}_c^{(0)} = \mathbf{0}$.

$$\mathbf{p}_c^{(g+1)} = (1 - c_c)\mathbf{p}_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}}\frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}} \tag{1.14}$$

where

$\mathbf{p}_c^{(g)} \in R^n$, evolution path at generation g.

$c_c \leq 1$. $1/c_c$ is the backward time horizon of the evolution path p_c that contains roughly 63% of the overall weight. A time horizon between $\sqrt{n}$ and n is reasonable.

The factor $\sqrt{c_c(2 - c_c)\mu_{\text{eff}}}$ is a normalization constant for p_c. For $c_c = 1$ and $\mu_{\text{eff}} = 1$, the factor reduces to one, and $\mathbf{p}_c^{g+1} = (\mathbf{x}_{1:\lambda}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^{(g)}$.

The (rank-one) update of the covariance matrix $\mathbf{C}^{(g)}$ via the evolution path $\mathbf{p}_c^{(g+1)}$ reads

$$\mathbf{C}^{(g+1)} = (1 - c_1)\mathbf{C}^{(g)} + c_1\mathbf{p}_c^{(g+1)}\mathbf{p}_c^{(g+1)^T}. \tag{1.15}$$

An empirically validated choice for the learning rate is $c_1 \approx 2/n^2$.

Using the evolution path for the update of C is a significant improvement of rank-μ-update for small μ_{eff}, because correlations between consecutive steps are exploited. The leading signs of steps, and the dependencies between consecutive steps play a significant role for the resulting evolution path $\mathbf{p}_c^{(g+1)}$.

1.5.3 Combining rank-μ-Update and cumulation

The final CMA update of the covariance matrix combines rank-μ-update and rank-one-update.

$$\begin{aligned}\mathbf{C}^{(g+1)} = (1 - c_1 - c_\mu)\mathbf{C}^{(g)} \\ + c_1 \underbrace{\mathbf{p}_c^{(g+1)}\mathbf{p}_c^{(g+1)^T}}_{\text{rank-one update}} + c_\mu \underbrace{\sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{(g+1)} \left(\mathbf{y}_{i:\lambda}^{(g+1)}\right)^T}_{\text{rank-}\mu\text{ update}}\end{aligned} \tag{1.16}$$

where

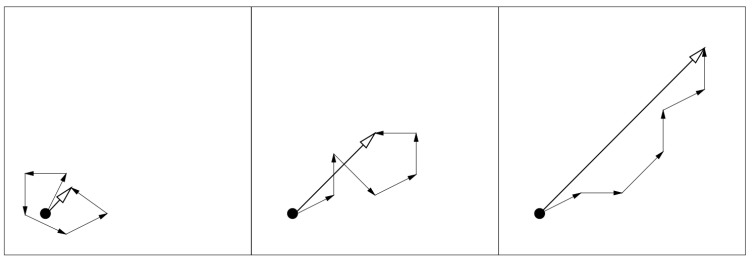


Figure 1.9: Three evolution paths of respectively six steps from different selection situations (idealized). The lengths of the single steps are all comparable. The length of the evolution paths (sum of steps) is remarkably different and is exploited for step-size control.

$c1 \approx 2/n^2$.

$c_\mu \approx \min(\mu_{\text{eff}}/n^2, 1 - c_1)$.

$\mathbf{y}_{i:\lambda}^{(g+1)} = (\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)})/\sigma^(g)$

The equation combines the advantages of rank-μ-update and rank-one-update. On the one hand, the information within the population of one generation is used efficiently by the rank-μ update. On the other hand, information of correlations between generations is exploited by using the evolution path for the rank-one update. The former is important in large populations, the latter is in particular important in small populations.

1.5.4 Step-size control

The covariance matrix adaptation, introduced in the last section, does not explicitly control the "overall scale" of the distribution, the step-size. The covariance matrix adaptation increases the scale only in one direction for each selected step, and it decreases the scale only implicitly by fading out old information via the factor $1 - c_1 - c_\mu$. There are two specific reasons to introduce a step-size control in addition to the adaptation rule for $\mathbf{C}^{(g)}$.

1. The optimal overall step length cannot be well approximated by covariance matrix adaptation, in particular if μ_{eff} is chosen larger than one.

2. The largest reliable learning rate for the covariance matrix update in is too slow to achieve competitive change rates for the overall step length.

To control the step-size $\sigma^{(g)}$ an evolution path is utilized, *i.e.* a sum of successive steps. The method can be applied independently of the covariance

matrix update and is denoted as *cumulative path length control*, cumulative step-size control, or *cumulative step length adaptation (CSA)*. The length of an evolution path is exploited, based on the following reasoning:

- Whenever the evolution path is short, single steps cancel each other out (Fig. 1.9, left). Loosely speaking, they are anti-correlated. If steps annihilate each other, the step-size should be decreased.
- Whenever the evolution path is long, the single steps are pointing to similar directions (Fig. 1.9, right). Loosely speaking, they are correlated. Because the steps are similar, the same distance can be covered by fewer but longer steps into the same directions. In the limit case, where consecutive steps have identical direction, they can be replaced by an enlarged single step. Consequently, the step-size should be increased.
- Subsuming, in the desired situation the steps are (approximately) perpendicular in expectation and therefore uncorrelated (Fig. 1.9, middle).

To decide whether the evolution path is "long" or "short", we compare the length of the path with its expected length under random selection. Under random selection consecutive steps are independent and therefore uncorrelated. If selection biases the evolution path to be longer then expected, σ is increased, and, vice versa, if selection biases the evolution path to be shorter than expected, σ is decreased. In the ideal situation, selection does not bias the length of the evolution path and the length equals its expected length under random selection.

To construct the evolution path, $\mathbf{p}_\sigma$, a conjugate evolution path is constructed, because the expected length of the evolution path $\mathbf{p}_c$ depends on its direction. Initialized with $\mathbf{p}_\sigma^{(0)} = 0$, the conjugate evolution path reads

$$\mathbf{p}_\sigma^{(g+1)} = (1 - c_\sigma)\mathbf{p}_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\mathbf{C}^{(g)^{-\frac{1}{2}}} \frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}} \tag{1.17}$$

where

$\mathbf{p}_\sigma^{(g)} \in \mathbb{R}^n$ is the conjugate evolution path at generation g.

$c_\sigma < 1$. Again $1/c_\sigma$ is the backward time horizon of the evolution path. For small μ_{eff}, a time horizon between $\sqrt{n}$ and n is reasonable.

$\sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}$ is a normalization constant.

$\mathbf{C}^{(g)^{-\frac{1}{2}}} \overset{def}{=} \mathbf{B}^{(g)}\mathbf{D}^{(g)^{-1}}\mathbf{B}^{(g)^T}$, where $\mathbf{C}(g) = \mathbf{B}^{(g)}\big(\mathbf{D}^{(g)}\big)^2\mathbf{B}^{(g)^T}$ is an eigendecomposition of $\mathbf{C}^{(g)}$, where $\mathbf{B}^{(g)}$ is an orthonormal basis of eigenvectors,

and the diagonal elements of the diagonal matrix $\mathbf{D}^{(g)}$ are square roots of the corresponding positive eigenvalues.

The transformation $\mathbf{C}^{(g)^{-\frac{1}{2}}}$ makes the expected length of $\mathbf{p}_\sigma^{(g+1)}$ independent of its direction, and for any sequence of realized covariance matrices $\mathbf{C}^{(g)}_{g=0,1,2,\ldots}$ we have under random selection $\mathbf{p}_\sigma^{(g+1)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, given $\mathbf{p}_\sigma^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. To update $\sigma^{(g)}$, we "compare" $\|\mathbf{p}_\sigma^{(g+1)}\|$ with its expected length $\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$, so final σ update is:

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma^{(g+1)}\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1\right)\right) \tag{1.18}$$

where

$d_\sigma \approx 1$, damping parameter, scales the change magnitude of $\sigma^{(g)}$.

$\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$, expectation of the Euclidean norm of a $\mathcal{N}(\mathbf{0}, \mathbf{I})$ distributed random vector.

For $\|\mathbf{p}_\sigma^{(g+1)}\| = \mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$ is $\sigma^{(g)}$ unchanged, while $\sigma^{(g)}$ is increased for $\|\mathbf{p}_\sigma^{(g+1)}\| > \mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$, and $\sigma^{(g)}$ is decreased for $\|\mathbf{p}_\sigma^{(g+1)}\| < \mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|$.

The length of the evolution path is an intuitive and empirically well validated goodness measure for the overall step length. For $\mu_{\text{eff}} > 1$ it is the best measure to our knowledge. Nevertheless, it fails to adapt nearly optimal step-sizes on very noisy objective functions.

Chapter 2

Current state

This work is a part of a larger project focusing on surrogate modeling in CMA-ES. In this project, the CMA-ES is interconnected with Gaussian processes (GP) or random forests. In each experiment, only one surrogate model is used, which is retrained from scratch in every iteration of the CMA-ES. This work used a surrogate model for CMA-ES called the DTS-CMA-ES (double trained surrogate CMA-ES) (see alg. 2). This alternative to CMA-ES is developed by L. Bajer, Z. Pitra, J. Repický, and M. Holeňa.[10].

In every generation of the DTS-CMA-ES, there are several operations:

1) Train a new GP model from scratch using archived original evaluated points. (If enough points are available).
2) Sample `minTrainSize` points from the current CMA-ES distribution.
3) Evaluate new points with the trained GP model.
4) Evaluate some points with the original objective function.
5) Validate the GP model with those points.
6) Retrain the GP model with some of those points.
7) Save the points evaluated with the original objective function to the archive for future model training.

2.1 Gaussian Processes

A Gaussian process on the considered set $\mathcal{X} \subseteq \mathbb{R}^D$ is a collection of random variables $(f_{GP}(\mathbf{x}))_{\mathbf{x}\in\mathcal{X}}$, indexed by the space $\mathcal{X}$, such that any finite n-element sub-collection has a joint n-dimensional normal distribution.

Each Gaussian process is specified by its mean $\mu : \mathcal{X} \to \mathbb{R}$ and covariance function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ where $\mu(\mathbf{x}) = \mathbb{E}[f_{GP}(\mathbf{x})]$ and $K(\mathbf{x}_1, \mathbf{x}_2) =$

Algorithm 2: DTS-CMA-ES

Input : original fitness function f, step-size $\sigma^{(0)} \in \mathbb{R}_+$, initial mean $\mathbf{m}^{(0)} \in \mathbb{R}^d$, ratio of original-evaluated points $\alpha^{(0)}$, criterion for the selection of original-evaluated points $\mathcal{C}$, maximum training set size $N_{\max}$,covariance function K

Output: $\hat{\mathbf{x}}^{\text{opt}}$ – point with the minimum achieved fitness

$\mathcal{A} \leftarrow \emptyset$; λ, $\sigma^{(0)}, \mathbf{m}^{(0)}, \mathbf{C} \leftarrow$ CMA-ES initialize ** initialization **/

for *generation* $g = 0, 1, 2, \ldots$ *until stopping conditions met* **do**

  $\mathbf{x}_k \sim \mathcal{N}(\mathbf{m}^{(g)}, (\sigma^{(g)})^2 \mathbf{C}^{(g)})$ for $k = 1, \ldots, \lambda$ ** CMA-ES sampling **/

  $f_{\mathcal{M}1} \leftarrow \text{trainModel}(\mathcal{A}, N_{\max}, K, \sigma^{(g)}, \mathbf{C}^{(g)})$ ** 1^{st} model training **/

  $(\hat{\mathbf{y}}, \hat{\mathbf{s}}^2) \leftarrow f_{\mathcal{M}1}([\mathbf{x}_1, \ldots, \mathbf{x}_\lambda])$ ** model-fitness evaluation **/

  $\mathbf{X}_{\text{orig}} \leftarrow$ select $\lceil \alpha^{(g)} \lambda \rceil$ best points according to the criterion $\mathcal{C}$

  $\mathbf{y}_{\text{orig}} \leftarrow f(\mathbf{X}_{\text{orig}})$ ** original-fitness evaluation **/

  $\mathcal{A} = \mathcal{A} \cup \{(\mathbf{X}_{\text{orig}}, \mathbf{y}_{\text{orig}})\}$ ** archive update **/

  $f_{\mathcal{M}2} \leftarrow \text{trainModel}(\mathcal{A}, N_{\max}, K, \sigma^{(g)}, \mathbf{C}^{(g)})$ ** model retrain **/

  $\mathbf{y} \leftarrow f_{\mathcal{M}2}([\mathbf{x}_1, \ldots, \mathbf{x}_\lambda])$ ** 2^{nd} model prediction **/

  $(\mathbf{y})_k \leftarrow (\mathbf{y}_{\text{orig}})_i$ for all original-evaluated $(\mathbf{y}_{\text{orig}})_i \in \mathbf{y}_{\text{orig}}$ ** fitness replace **/

  sorted $\mathbf{x}_{1:\lambda} \leftarrow$ sort $\mathbf{x}_1, \ldots, \mathbf{x}_\lambda$ based on $(\mathbf{y}_1, \ldots, \mathbf{y}_\lambda)^\top$ ** population sort **/

  $\sigma^{(g+1)}, \mathbf{m}^{(g+1)}, \mathbf{C}^{(g+1)} \leftarrow$ CMA-ES update based on $\mathbf{x}_{1:\lambda}$

end

$\hat{\mathbf{x}}^{\text{opt}} \leftarrow \mathbf{x}_k$ from $\mathcal{A}$ corresponding to the minimal y_k

$\text{cov}(f_{GP}(\mathbf{x}_1), f_{GP}(\mathbf{x}_2))$. Both functions are parametrized by a relatively small number of parameters which are usually fitted by the maximum-likelihood (ML) or leave-one-out cross-validation (LOO-CV) method. Since both functions μ, K are themselves parameters of the GP, their parameters are usually called hyperparameters of the GP.

Function values y are often accessible only as noisy observations $y = f_{GP}(\mathbf{x}) + \varepsilon$ where ε is a zero-mean Gaussian noise with the variance σ_n^2. The noise variance σ_n^2 determines how precisely the GP can fit to the training data. Consequently, the covariance of noisy observations becomes

$$\text{cov}(y_p, y_q) = K(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{p,q} \tag{2.1}$$

where $\delta_{p,q} = 1$ for $p = q$ and $\delta_{p,q} = 0$ otherwise.

Prediction. Using a Gaussian process for prediction always starts with a training set of N points $\mathbf{X}_N = \{\mathbf{x}_i \mid \mathbf{x}_i \in \mathcal{X},\ i = 1, \ldots, N\}$ for which the function values $\mathbf{y}_N = \{y_i = f(\mathbf{x}_i),\ i = 1, \ldots, N\}$ are known. If the mean function

μ of the GP is zero, then $\mathbf{y}_N \,|\, \mathbf{X}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_N)$. We consider the prediction in a new, $(N+1)$-st point $(\mathbf{x}^*, y^*)$. Adding this point to the training set, the conditional distribution of the extended vector $\mathbf{y}_{N+1} = (y_1, \ldots, y_N, y^*)^\top$ is

$$\mathbf{y}_{N+1} \,|\, \mathbf{X}_{N+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{N+1}). \tag{2.2}$$

Let us look at the covariance matrices if the observations are considered *noisy*:

$$\begin{aligned} \mathbf{C}_N &= \mathbf{K}_N + \sigma_n^2 \mathbf{I}_N, \quad \text{where} && (2.3)\\ (\mathbf{K}_N)_{i,j} &= (K(\mathbf{X}_N, \mathbf{X}_N))_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) && (2.4) \end{aligned}$$

and $\mathbf{K}_N = K(\mathbf{X}_N, \mathbf{X}_N)$ is the matrix of the covariance-function values between all the training points. The extended covariance matrix can be written as

$$\mathbf{C}_{N+1} = \begin{pmatrix} K(\mathbf{X}_N, \mathbf{X}_N) + \sigma_n^2 \mathbf{I}_N & K(\mathbf{X}_N, \mathbf{x}^*) \\ K(\mathbf{x}^*, \mathbf{X}_N) & K(\mathbf{x}^*, \mathbf{x}^*) \end{pmatrix} \tag{2.5}$$

where

$K(\mathbf{X}_N, \mathbf{x}^*) = K(\mathbf{x}^*, \mathbf{X}_N)^\top$ is the vector of covariances between the new point $\mathbf{x}^*$ and the training data $\mathbf{X}_N$.

Finally, conditioning the prior distribution of $\mathbf{y}_{N+1}$ (2.2) on the observations $\mathbf{y}_N$, the one-dimensional Gaussian distribution of y^* is

$$\begin{aligned} y^* \,|\, \mathbf{X}_{N+1}, \mathbf{y}_N &\sim \mathcal{N}(\hat{y}^*, (\hat{s}^*)^2), \quad \text{where} && (2.6)\\ \hat{y}^* &= K(\mathbf{x}^*, \mathbf{X}_N)\, \mathbf{C}_N^{-1}\, \mathbf{y}_N && (2.7)\\ (\hat{s}^*)^2 &= K(\mathbf{x}^*, \mathbf{x}^*) - K(\mathbf{x}^*, \mathbf{X}_N)\, \mathbf{C}_N^{-1}\, K(\mathbf{x}^*, \mathbf{X}_N)^\top. && (2.8) \end{aligned}$$

Abandoning the assumption $\mu(\mathbf{x}) = \mathbf{0}$, equation (2.7) generalizes to

$$\hat{y}^* = \mu(\mathbf{x}^*) + K(\mathbf{x}^*, \mathbf{X}_N)\, \mathbf{C}_N^{-1}\, (\mathbf{y}_N - \mu(\mathbf{X}_N)) \tag{2.9}$$

where the mean function is most commonly set to a constant $\mu(\mathbf{x}) = m_\mu$.

From the computational perspective, the calculation of the covariance matrix $\mathbf{C}_N$ takes $\mathcal{O}(DN^2)$ time, and the complexity of the likelihood calculation for hyperparameter estimation is $\mathcal{O}(N^3)$ due to inversion of $\mathbf{C}_N$. Once the $\mathbf{C}_N^{-1}$ is calculated, the complexity of the prediction is only $\mathcal{O}(N^2)$.

Criteria for the selection of original-evaluated points. While the non-gaussian-process surrogate algorithms select points for the original evaluation mostly according to the predicted fitness, the Gaussian process surrogates, which for any point $\mathbf{x}$ predict the whole Gaussian distribution $\mathcal{N}(\hat{y}(\mathbf{x}), (\hat{s}(\mathbf{x}))^2)$, offer more options when used with the individual-based or doubly trained EC. The following criteria are considered in the DTS-CMA-ES. Whereas the first four are defined for any point of the input space and have been used in Bayesian optimization for decades, the last one, *Expected ranking difference error*, is our new contribution directly exploiting the DTS-CMA-ES' Gaussian processes prediction and is defined only for the points from the considered population.

- *GP predictive mean.* The GP mean prediction $\hat{y}(\mathbf{x})$ is the maximum-likelihood estimate of the original fitness. This criterion is defined as its negative value

$$\mathcal{C}_{\mathrm{M}}(\mathbf{x}) = -\,\hat{y}(\mathbf{x})\,.$$

- *GP predictive standard deviation.* Choosing the points with the highest uncertainty leads to the criterion

$$\mathcal{C}_{\mathrm{STD}}(\mathbf{x}) = \hat{s}(\mathbf{x})\,.$$

- *Expected improvement* (EI). If $y_{\min}$ stands for the minimum fitness in the considered training set $y_1, \ldots, y_N$, the EI criterion is

$$\begin{aligned} \mathcal{C}_{\mathrm{EI}}(\mathbf{x}) &= E((y_{\min} - f(\mathbf{x}))I(f(\mathbf{x}) < y_{\min}) \,|\, y_1, \ldots, y_N)\,, \text{ where} \\ I(f(\mathbf{x}) < y_{\min}) &= \begin{cases} 1 & \text{for } f(\mathbf{x}) < y_{\min} \\ 0 & \text{for } f(\mathbf{x}) \geq y_{\min} \end{cases}. \end{aligned}$$

- *Probability of improvement* (PoI). The PoI express the probability of finding lower fitness than some threshold T

$$\mathcal{C}_{\mathrm{PoI}}(\mathbf{x}, T) = P(f(\mathbf{x}) \leq T \,|\, y_1, \ldots, y_N) = \phi\left(\frac{T - \hat{y}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right)$$

  where ϕ is the distribution function of $\mathcal{N}(0, 1)$. As T, the value $T = y_{\min}$ or a slightly higher value is usually chosen.

- *Expected ranking difference error* (ERDE). The goal of this criterion is to select the points from the current population for which the expected RDE would decrease most after adding them to the GP training set. It is a ranking counterpart of the *GP predictive standard deviation* criterion since it selects the points for which the model is least certain. In DTS-CMA-ES, the RDE_{μ} has been used as an error measure.

2.2 Random forests

Another type of surrogate models used in this project is a random forest. A random forest is actually an ensemble of decision trees. Decision trees have a wide spectrum of forms, utilizations, and properties. This project pays attention only to binary regression trees with real inputs. In such regression trees, each observation $\mathbf{x} = (x_1, x_2, \ldots, x_D) \in \mathbb{R}^D$ passes through a series of binary decisions $(x_i \overset{?}{<} c \in \mathbb{R})$ associated with internal nodes and arrives in the leaf node containing a real-valued constant utilized for the prediction of function values y. Each binary decision determines whether the observation

proceeds to the left or right child of the respective internal node. The tree growing starts with one node (root) and a set of all input data. As the first step, real constants (left and right) for all allowable splits of the input set in all variables of the input space are calculated by averaging function values of training points in respective subsets. As the second step, a split into two subsets according to the with the minimal mean-squared error (MSE) between the resulting averages and the training points is chosen, and apropriate child nodes are connected to the root. Those two steps are repeated recursively with the children. The tree growing stops whenever any of the user-defined constraints holds: the tree reaches the allowed maximum of splits, the value of the MSE of the predictors decreases below a specified threshold, or if each node contains at least the defined number of points and any additional split would violate it. An important aspect of random forests are random differences between individual trees within the ensemble. This increases robustness and improves predictive generalization. The forest gains randomness during training by bagging (see 3). The overall forest prediction is provided by averaging all tree predictions. This form of prediction means the larger the ensemble the greater the robustness to noise.

2.3 Experimental setting

One goal of my work is to test surrogate models with the BBOB noisy benchmark. The project to which my work belongs is focused only on BBOB benchmarks without noise so it isn't optimalized for this task. However, the CMA-ES algorithm is believed to handle noisy functions with ease. My task is to validate this. The project has many settings concerning the BBOB, surrogate management, model parameters and the CMA-ES itself. Experiments are configured in a text file and performed on the Czech MetaCentrum[11], online grid infrastructure. Here is an experiment configuration example[12]

```
% BBOB/COCO framework settings
bbobParams = {
  'dimensions',      {2},
  'functions',       {102},
  'opt_function',    {@opt_s_cmaes },
  'instances',       {[1:5, 41:50] },
  'maxfunevals',     {'250 * dim' },
};
```

- `dimensions` – list of dimensions of input space which should be tested 2,3,5,10,20 (integer, integer...)
- `functions` – list of BBOB function numbers to be tested cell2num(1:24) (integer, integer...)

- `opt_function` – handle to a BBOB wrapper for the optimizer @opt_s_cmaes (function handle)
- `instances` – vector of instances to be tested; for BBOB final results, use the default value [1:5, 41:50] (integer vector)
- `maxfunevals` – maximal number of (original) function evaluations – string to be eval-ed in surrogateManager(); particularly, the dim parameter can be used '250 * dim' (string)

```
% Surrogate manager parameters
surrogateParams = {
 'evoControl',           { 'doubletrained' },
 'modelType',            { 'gp' },
 'evoControlPreSampleSize', { 0 },
 'evoControlTrainRange', { 5, 10 },
 'evoControlTrainNArchivePoints', { '15*dim', '30*dim' }
 'evoControlSampleRange', { 1 },
 'evoControlRestrictedParam', { 0.1, 0.2, 0.6, 0.9},
};
```

surrogateParams – structure array defining behaviour and settings of the surrogate modelling, i.e. for the function surrogateManager() and functions called from within there. The following settings is recommended to be set:

- `evoControl` – type of evolution control to be used 'none' ('doubletrained' — 'individual' — 'generation' — 'none')
- `modelType` – the type of a surrogate model to be used, Gaussian processes and random forests are implemented so far. The special model 'bbob' stands for a virtual model which in fact returns exact values from the respective BBOB functions. The default value '' causes an error, so it really should be set to any of the three valid models. '' ('gp' — 'rf' — 'bbob')
- `evoControlPreSampleSize` – 0.25, 0.5, 0.75, will be multiplied by population size. This number determine number of points evaluated by original fitness function.
- `evoControlTrainRange` – will be multiplied by σ in CMA-ES covariace matrix. It is maximum point distance from CMA-ES mean in current generation. It is used to get subset of archived points.
- `evoControlTrainNArchivePoints` – selects points from subset given by `evoControlTrainRange` to re-train model. if too many points are given, clustering is performed (e.g. by k-means).

- `evoControlSampleRange` – will multiply σ used in CMA-ES, therefore increase probability of selecting more distant points.

- `evoControlRestrictedParam` – percentage of points in subset evaluated by original fitness function.

```
% Model parameters
modelParams = { ...
  'useShift',              { false },
  'predictionType',        { 'sd2' },
  'trainAlgorithm',        { 'fmincon' },
  'covFcn',                { '{@covMaterniso, 5}' },
  'hyp',                   { struct('lik', log(0.01), ...
                                    'cov', log([0.5; 2]))},
};
```

modelParams – structure array defining behaviour and settings of the model.

- `useShift` – whether use shift mean during generationUpdate().

- `predictionType` – type of prediction (f-values, PoI, EI).

- `trainAlgorithm` – type of learning algoritm (usualy fmincon).

- `covFcn` – type of covariation function.

- `hyp` – structure array of starting values of hyperparameters for likelihood function and covariation function.

```
% CMA-ES parameters
cmaesParams = { ...
   'PopSize',              { '(4 + floor(3*log(N)))'},
   'Restarts',             { 100 },
};
```

cmaesParams

- `PopSize` – number of individuals in each generation.

- `Restarts` – number of CMA-ES restarts. After every restart, CMA-ES population size is doubled and CMA-ES starting point is best point in previous run.

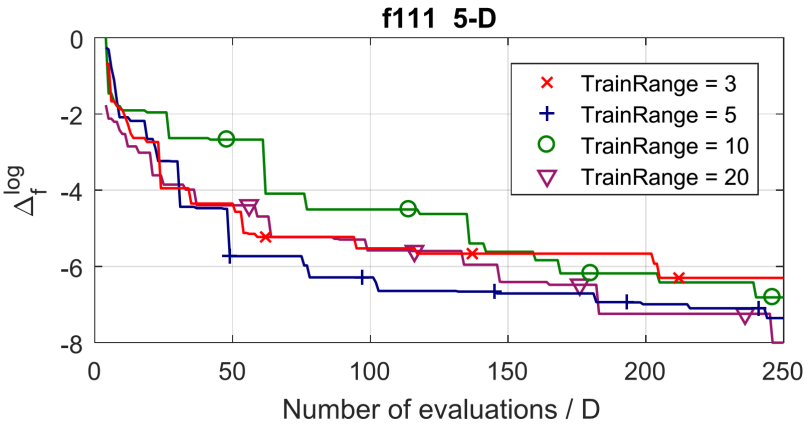


Figure 2.1: evoControlTrainRange comparison

Even though there are many parameters, some of them are rarely changed or used. The most important are:

- `dimensions`
- `functions`
- `evoControlTrainRange`
- `evoControlTrainNArchivePoints`
- `evoControlRestrictedParam`
- `PopSize`

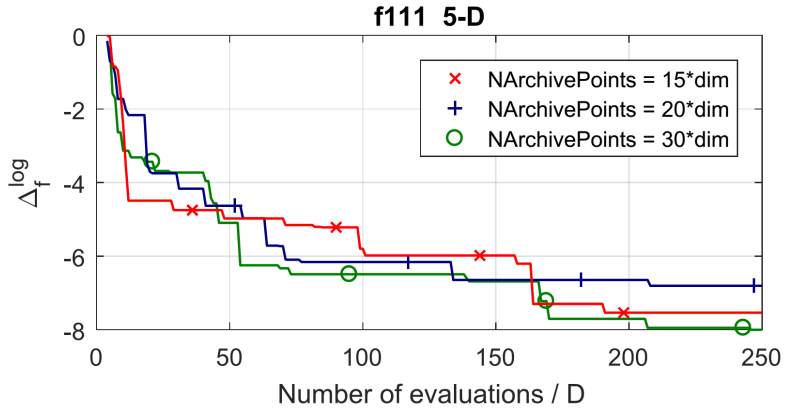


Figure 2.2: evoControlTrainNArchivePoints comparison

The evaluation of individual parameters is discussed in the following sections. The most important quality measure is th number of function evaluations divided by dimension (FE/D). This value says how many evaluation of actual fitness function the DTS-CMA-ES needs to reach a given error threshold, currently set to 10^{-8}. Every algorithm performs differently in each function and/or dimension. I choosed function F111 in 5D to parameters tunning.

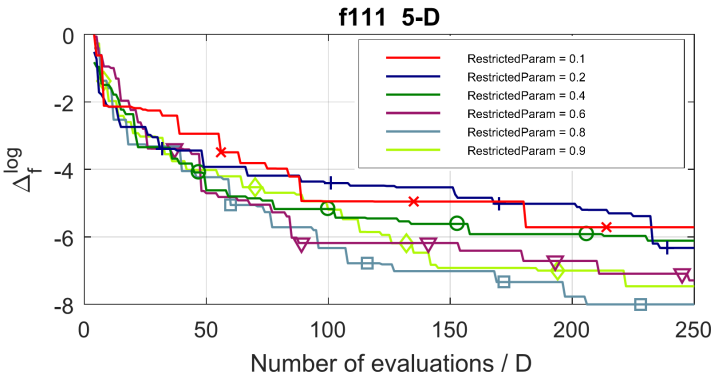


Figure 2.3: evoControlRestrictedParam comparison

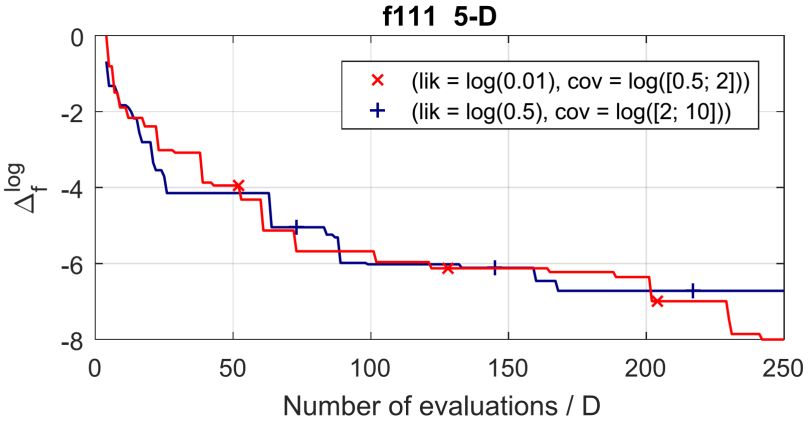


Figure 2.4: GP model's hyperparameters comparison

evoControlTrainRange.

Figure 2.1 compares several values of `evoControlTrainRange`. In particular (3,5,10,20). As described in experiment configuration example (see 2.3) . Figure shows that best value is 20, but 10 is very good too. I used `evoControlTrainRange` = 10 in final experiment.

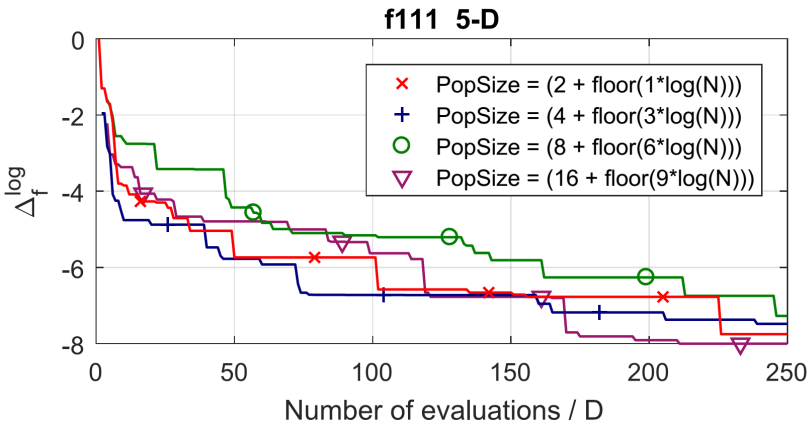


Figure 2.5: PopSize comparison

evoControlTrainNArchivePoints. Figure 2.2 shows few different values of `evoControlTrainNArchivePoints`. Similar to train range, higher values are better. Once again, I will not use best value, but a second one, which performs similarly to the best value.

evoControlRestrictedParam. `evoControlRestrictedParam` is a very interesting parameter. It is similar to the learning rate in machine learning algorithms. In the performed experiments, results for a low value close to 0.0 and results for a high value close to 1.0 were similar. In parameter tunning for the function f111 in 5D, the best value was 0.8, according to figure 2.3, so I used this value for the final experiments.

hyp. GP model's hyperparameters are very important for good performance. According to figure 2.4 `struct('lik', log(0.01), 'cov', log([1; 2]))` is far better, than `struct('lik', log(0.5), 'cov', log([2; 10]))` . This value will be used in final experiments.

PopSize. Last, but not least parameter is population size. Common sense suggests that more is better, but the results in 2.5 shows otherwise. The best is largest population, but second to that is the smallest one. I will use $PopSize =' (8 + floor(6 * log(N)))'$ in final experiments, because it recommended value in main project. Higher value will take much longer to evaluate and smaller one may prevent diversity needed in evolution, which is part of final experiments.

Chapter 3

Novel contributions

In the project, only one surrogate model is currently used with hyperparameters likelihood and covariance function. This model has same starting configuration in each CMA-ES generation. According to the original idea, there is no need to have more models, because Gaussian model will be retrained from scratch to fit the needs. However, if more models were used, improvements could be achieved. One possible solution is to use a model pool and every few generatoions, choose the best model based on several previous generations. In this case, it is important to solve how to choose the new model.

Second idea is improvement of previous one. Model pool requires some archive for models and a function to compare models. With there preconditions it is easy to implement a model evolution. This approach needs only recombination on top of model pool. There will be higher requirements on computing time, but it should be insignificant compared to model pool.

Another possibility is not to choose one model from model pool, but use ensembling methods to predict function evaluations. This approach was consulted with Lukáš Bajer, author of DTS-CMA-ES. He doesn't approve this idea, because it will take much CPU time and probably won't be effective enouqh to pay off, because ensembling in regresion is difficult and easily afected by distant search points. For ensembling is important, to combine diffent models, otherwise, the improvement is insignificant. So I implement only model pool with evolution. However, I believe these ensembling methods are worth of mention[13]:

- bagging - [14] is machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithm. Given a standard training set $\mathbf{D}$ of size n, bagging generates m new training sets $\mathbf{D}_i$, each of size n' by sampling from $\mathbf{D}$ uniformly and with replacement.The m models are fitted sing the above m bootstrap samples and combined by averaging the output (for regresion) or voting (for classification).

- boosting - is a two-step approach, where one first uses subsets of original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.

- stacking - is a similar to boosting: you also apply several models to your original data. The difference here is, however, that you don't have just empirical formula for your weight function, rather you introduce a meta-level and use of another model/approach to estimate the input together with outputs of every model to estimate the weights or, in other words, to determine what models perform well and what badly given there input data.

3.1 Proposed metamodel

The main goal of this work is to design and implement a meta-model that will train several or a multitude of different GP models and automatically select one that performs best in the current state of the optimization. Experimental setting needs additional fields and parameters for a proper model pool configuration. The proposed are:

Fields:

- `currentGeneration` – CMA-ES iteration number g

- `archive` – archive A from the `surrogateManager`() Archive is a Matlab handle class (and hence passed to functions effectively by reference)

- `models` – 2D cell array (of size $N_{GP}\times$ `historyLength` + 1) of models from generations g, (g - 1), (g - 2), ..., (g - `historyLength`) - the models from generations g, (g - 1), ...are in `models`(:,1), `models`(:,2), ...respectively

- `bestModel` - 1D array (of size `historyLength` + 1) of the best found models' indices $i^{(g)} \in 1, ..., N_{GP}$ for respective generations g, (g - 1), (g - 2), ..., (g - `historyLength`) measured according to the the point (2.) from the `trainModel`(). The model (usually re-trained using the current archive A) with the `parameterSet` corresponding to the best model from the generation (g - `historyLength`) will be used in the calls of `modelPredict`() until the next call of `trainModel`().

Parameters:

- `retrainPeriod` - how often should the `ModelPool` retrain the hyperparameters for the models. Default: 1 (i.e. every generation).
- `parameterSets` – structure array of parameter sets which define the N_{GP} different models being maintained within the `ModelPool`.
  - `trainsetType` ∈ allPoints, clustering*, nearest*, nearestToPopulation* – how should be the points selected from the points in `archive` A from the models' train range (see next item); trainsetType's marked with * calculate with the maximum number of points trainsetSizeMax. Default: clustering.
  - `trainRange` ∈ 99%, 99.9% – χ^2-percentile of the maximum squared $\sigma^2\mathbf{C}$-distance from the current CMA-ES mean $\mathbf{m}^{(g)}$ of the points considered from the `archive` A for being used in the train set for models. Default: 99%.
  - `trainsetSizeMax` – the maximal number of points for selection into the model's train set (applicable for the trainsetType's marked with star). Default: 15*dim.
  - `covFcn` ∈ '@covSEiso','@covMaterniso,3','@covMaterniso,5' – covariance function. Default: '@covMaterniso,5'.
  - `ard` ∈ true, false – whether use ARD (automatic relevance determination) or not. If yes, covariance functions in the previous point should be transformed to their ARD counterparts. Default: false.
  - `meanFcn` ∈ meanConst, meanLinear. Default: meanConst.
- `historyLength` how many generations old models should be tested against unseen data. Default: 4.

Example of the ModelPool's parameter setting in experiment definition ($N_{GP} = 3$):

```
surrogateParams = { ...
... \% ...
'modelType', { 'modelPool' }, ...
... \% ...
};
modelParams = { ...
'predictionType', { 'expectedRank' }, ...
'retrainPeriod', { 2 }, ...
'parameterSets', { struct( ...
'trainsetType', { 'allPoints', 'clustering', 'nearest' }, ...
'trainRange', { 0.99, 0.99, 0.999 }, ...
'trainsetSizeMax', { '10*dim', '10*dim', '5*dim' }, ...
```

```
'covFcn', { '{@covSEiso}', ...
'{@covMaterniso,5}', ...
'{@covMaterniso,5}' }, ...
'ard', { false, false, false }, ...
'hyperparamStartValues', { ...
struct('lik', log(0.05), 'cov', log([1; 2])), ...
struct('lik', log(0.01), 'cov', log([0.5; 2])), ...
struct('lik', log(0.01), 'cov', log([0.5; 2])) }, ...
'meanFcn', { 'meanConst', 'meanConst', 'meanLinear' } ...
) }, ...
'historyLength', { 4 }, ...
};
```

trainModel().

(Usually) re-fitting of the hyperparameters of the models and (always) selection of the best model for the future predictions.

1. if g — `retrainPeriod` (g modulo `retrainPeriod` == 0, g is the number of current CMA-ES generation), train N_{GP} Gaussian process models $M_1^{(g)}, \ldots, M_{N_{GP}}^{(g)}$ according to their definition in `parameterSets` – struct array of N_{GP} GP model parameter sets (which include, for example, covariance function or hyperparameters' starting values for the model training). Save these models into `models`(:,1). These models are trained based on the current state of the DTSCMA-ES in generation g: particularly using the current CMA-ES mean $\mathbf{m}^{(g)}$, the step-size $\sigma^{(g)}$, the square root of the CMA-ES covariance matrix $\mathbf{B}(g)\mathbf{D}(g)$ where $\mathbf{B}^{(g)}(\mathbf{D}^{(g)})^2(\mathbf{B}^{(g)})^2 = \mathbf{C}^{(g)}$ and the `archive` of original-evaluated points $A^{(g)} = x_i, y_i|_{i=1}^{|A|}$.

2. select the best model to be used in the current generation g according to one of the following criteria:

   – Ranking Difference Error (EDE, Err_{RD}^{μ}) measured on the original-evaluated points from the models' future generations: the previously saved models from the generation (g - `historyLength`) are assessed on the points from the `archive` A from the generations (g - `historyLength` + 1),...,(g - 1) using the RDE without the parameter μ (technically, μ for the `errRankMu`() can be set to the number of these points from the `archive` A). If the original-evaluated points from the current generation g are already saved into the `archive` A, they are used for measuring error, too.

   Ranking Difference Error (Err_{RD}^{μ}, RDE), impelemented in [errNorm, errSum, maxErr] = `errRankMu`($\mathbf{y}_1, \mathbf{y}_2, \mu$), is a non-symetric

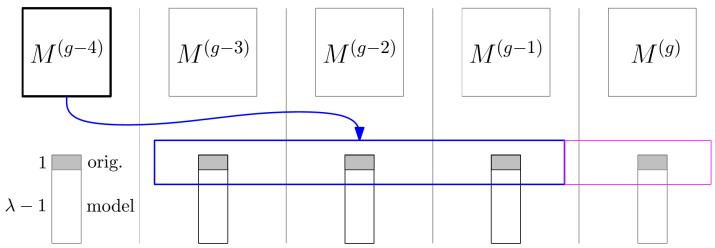


Figure 3.1: RDE measured on points from next generations

measure of the difference between rankings of two vectors of population's f-values $\mathbf{y}_1^*, \mathbf{y}_2^*$

$$Err_{\mathrm{RD}}^{\mu}(\mathbf{y}_1^*, \mathbf{y}_2^*). \tag{3.1}$$

It measures how much would the difference between ranking of $\mathbf{y}_1^*$ and $\mathbf{y}_2^*$ (possibly negatively) contribute for the CMA-ES μ -updates where $\mu, 1 \leq \mu \leq \lambda$ is the number of best-ranked individuals which are solely used for the CMA-ES' updates.

The measure takes rankings τ_1,
tau_2 of the values of the input vectors (e.g. $\tau_1(1)$ is the rank of the $y_1^*(1)$- the first element of the vector $\mathbf{y}_1^*$), and calculates the ranking distance as a normalized sum of the element-wise differences between these two rankings while omitting the indices for which the rank of $\mathbf{y}_2^*$ is larger than μ. The second f-values vector $\mathbf{y}_2^*$ is considered as being more precisely measured.

$$Err_{\mathrm{RD}}^{\mu}(\mathbf{y}_1^*, \mathbf{y}_2^*) = \frac{\sum_{i:\tau_2(i)\leq\mu} |\tau_2(i) - \tau_1(i)|}{\max_{\pi\in\text{Permutations of}(1,\ldots,\lambda)} \sum_{i:\pi(i)\leq\mu} |i-\pi(i)|} \in \langle 0, 1\rangle \tag{3.2}$$

– Ranking Difference Error (RDE, Err_{RD}^{μ}) measured on the whole populations from the models' next generation: the respective previously saved models from the generation $i = (g$ - `historyLength`),..., $(g$ - 2) are assessed on the whole population from the generations (i + 1) using the RDE with the original parameter μ. If the original-evaluated points from current generation g are already saved into the `archive` A, this population is used for measuring error, too. For the final error, the averaged Err_{RD}^{μ} from these generations is taken.

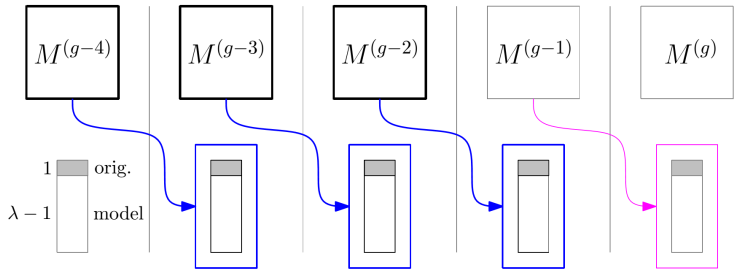


Figure 3.2: RDE measured on the whole populations.

- 3. The selected model will be used in the next calls of `modelPredict`(). Shift `bestModel`(1:(end-1)) by one position further and save the best index into the just freed `bestModel`(1).

modelPredict().

Having trained the hyperparamters of the models and having selected the best model for the current generation in `bestModel`(1), this method simply predicts expected value and variance of the best Gaussian process model which is saved in `models`{1, `bestModel`(1)}.

3.2 Enhanced metamodel

Metamodel `ModelPool` introduced in previous section was improved by model evolution. GP models in ModelPool are static, defined by user on the very beginning. Aim of the enhancement proposed in this section is make GP models dynamic to fit the needs of CMA-ES in the current search area. Dynamics of models is achieved by recombination of the fittest models. This version of `metamodel` is similar to ModelPool, however, with few addition functions:

- `Evolve`() - main function to handle evolution. Prepare models, run other funtions and return new models.
- `Mutation` () - alter each gene in chromosome with given probability. Discrete values are randomly chosen 1:N, continuous values are changed by 1%.
- `Crossover`() - unimodal crossover, parents are radomly choosen from population.
- `Selection`() - by elitism, only the fittest half survive. The rest is replaced by new offsprings. Fitness value is calculated by RDE, same way as in ModelPool.

- `geneCode`() / **geneDecode**() - transformation from string to number and back, for easier recombination.

Chapter 4

Final experiments

This chapter is focused on interpreting final experiments performed on metacentrum. I compared these three algorithms: Original DTS-CMA-ES (red line), DTS-CMA-ES with modelPool meta-model (called MP-CMA-ES, green line) and DTS-CMA-ES with model evolution (called evo-CMA-ES, blue line).

Experimental setting for MP-CMA-ES and evo-CMA-ES:

```
% BBOB/COCO framework settings

bbobParams = { ...
  'dimensions',         { 2, 3, 5, 10}, ...
  'functions',          { 102, 104, 107, 111, 117, 122, 123, 129}, ...
  'opt_function',       { @opt_s_cmaes }, ...
  'instances',          num2cell( [1:5, 41:50] ), ...
  'maxfunevals',        { '250 * dim' }, ...
  'resume',             { true }, ...
};

% Surrogate manager parameters

surrogateParams = { ...
  'evoControl',         { 'doubletrained' }, ...
  'observers',          { {'DTScreenStatistics', 'DTFileStatistics'} },...
  'modelType',          { 'modelPool' }, ...
  'updaterType',        { 'rankDiff' }, ...
  'evoControlMaxDoubleTrainIterations', { 1 }, ...
  'evoControlPreSampleSize',       { 0.75 }, ...
  'evoControlOrigPointsRoundFcn',  { 'ceil' }, ...
  'evoControlTrainRange',          { 10 }, ...
  'evoControlTrainNArchivePoints', { '20*dim' },...
  'evoControlSampleRange',         { 1 }, ...
```

```
  'evoControlRestrictedParam',     { 0.8 }, ...
};

% Model parameters

structure = struct();

structure(1).covFcn = '{@covMaterniso, 5}';
structure(1).trainsetType = 'allPoints';
structure(1).trainRange = 1;
structure(1).trainsetSizeMax = '10*dim';
structure(1).meanFcn = 'meanConst';
structure(1).trainAlgorithm = 'fmincon';
structure(1).hyp.lik = -4.605170;
structure(1).hyp.cov = [-0.693147; 0.693147];


modelParams = { ...
    'retrainPeriod',      { 1 }, ...
    'bestModelSelection', { 'rdeAll' }, ...
    'historyLength',      { 7 }, ...
    'minTrainedModelsPercentileForModelChoice', {0.5},...
    'maxGenerationShiftForModelChoice', {2},...
    'predictionType',     { 'poi' }, ...
    'useShift',           { false }, ...
    'normalizeY',         { true }, ...
    'parameterSets', {  structure  }};

% CMA-ES parameters

cmaesParams = { ...
  'PopSize',            { '(8+floor(6*log(N)))' }, ...
  'Restarts',           { 50 }, ...
  'DispModulo',         { 0 }, ...
};
```

Setting for DTS-CMA-ES is similar, except model parameters, which are:

```
% Model parameters

modelParams = { ...
  'covFcn',             { '{@covMaterniso, 5}' }, ...
  'hyp',                { struct('lik', log(0.01), 'cov', log([1; 2])) }, ...
```

```
  'covBounds',          { [log(1) log(1); -2 25 ] }, ...
  'meanFcn',            { 'meanConst' }, ...
  'trainAlgorithm',     { 'fmincon' }, ...
  'predictionType',     { 'poi' }, ...
  'useShift',           { false }, ...
  'normalizeY',         { true }, ...
  'trainsetType'        { 'nearest' }, ...
  'trainRange',         { 4 }, ...
  'trainsetSizeMax'     { '20*dim' }, ...
};
```

All graphs in this chapter show decreasing error value in time, more precisely in original function evaluations divided by dimension. Every experiment is 50-times restarted and values are averaged. This is very important not only because noise in fuctions, but also for evo-CMA-ES algorithm where recombination depends on random chance.

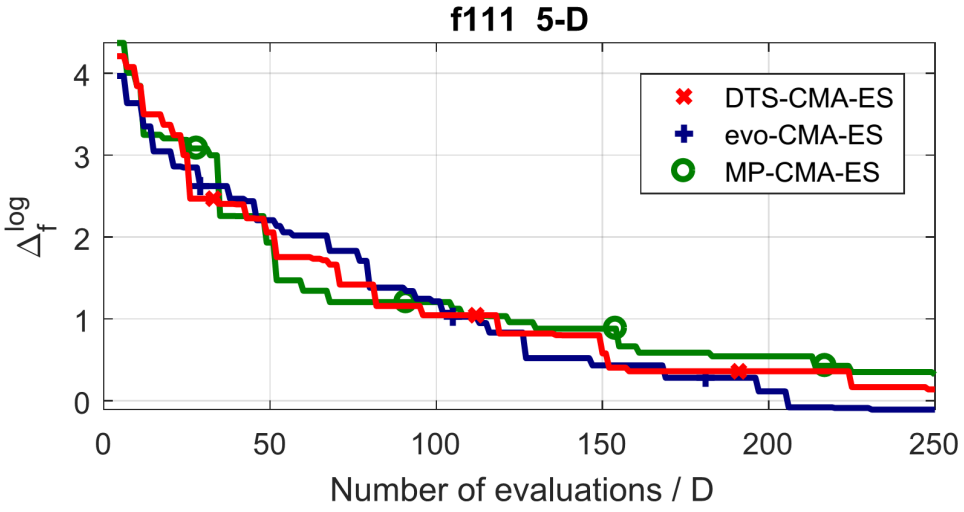


Figure 4.1: Function F111 in 5D

In a graph 4.1 we can see, that evo-CMA-ES (blue line) is the best in this particular experiment. On the other hand, MP-CMA-ES is worst. This may raise the question, how can be MP-CMA-ES worse than GP-CMA-ES, if the GP model used in GP-CMA-ES is in a modelPool meta-model in MP-CMA-ES. The answer is simple, MP-CMA-ES choose best model according to performance in generation $g - 1$, so the chosen surrogate model is best for previous generation, not the current one.

Graph 4.2 shows, that all three alternatives performs similar, with a little worse DTS-CMA-ES. This is expected result, because DTS-CMA-ES has only

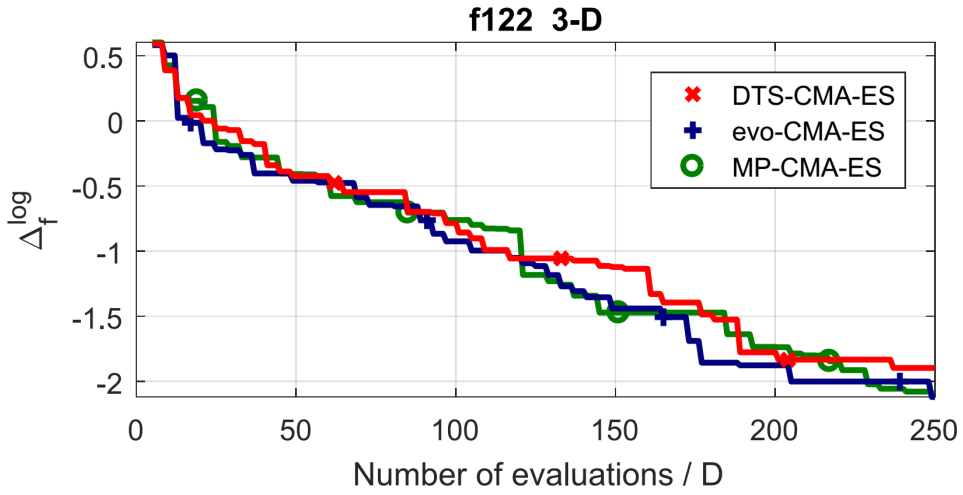


Figure 4.2: Function F122 in 3D

one surrogate model, so it can't adapt to match the environment. On the other hand, DTS-CMA-ES advantage is speed performance. Other two algorithms has to retrain several surrogate models in every generation, however, DTS-CMA-ES retrain only once in every generation.

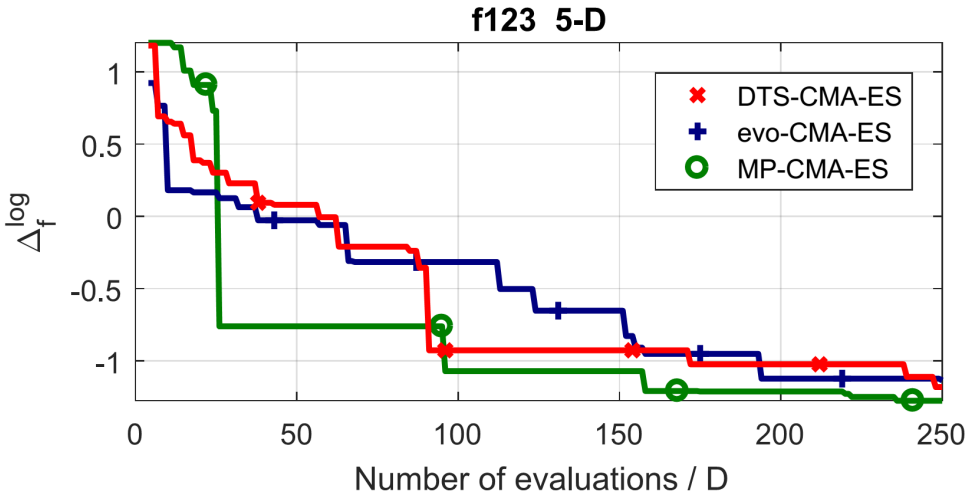


Figure 4.3: Function F123 in 5D

In a graph 4.3 MP-CMA-ES bested other two algorithms. My theory is that one surrogate model in MP-CMA-ES perfectly matches needs of this function. Evo-CMA-ES was unable to find better model that is better than a surrogate model used in DTS-CMA-ES. This suggests that evolution process

in evo-CMA-ES needs to be improved.

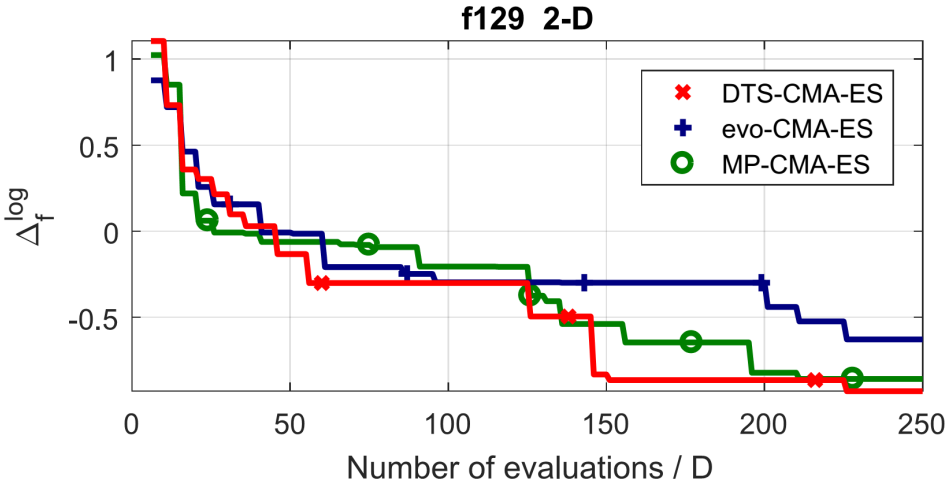


Figure 4.4: Function F129 in 2D

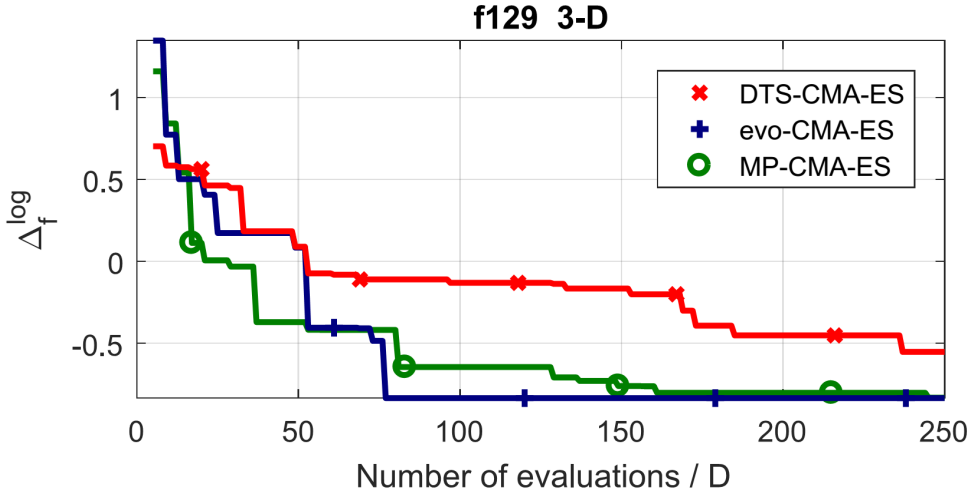


Figure 4.5: Function F129 in 3D

Function F129 is one of the most difficult functions in this experiment, and provides very interesting results. Graph 4.4 shows DTS-CMA-ES and MP-CMA-ES as best algorithms, in 3-Dimensions 4.4 are best eco-CMA-ES and MP-CMA-ES leaving DTS-CMA-ES far behind. Function F129 in 5D and 10D show similar performace for DTS-CMA-ES and evo-CMA-ES, however, MP-CMA-ES performace is very different in 5D and 10D.

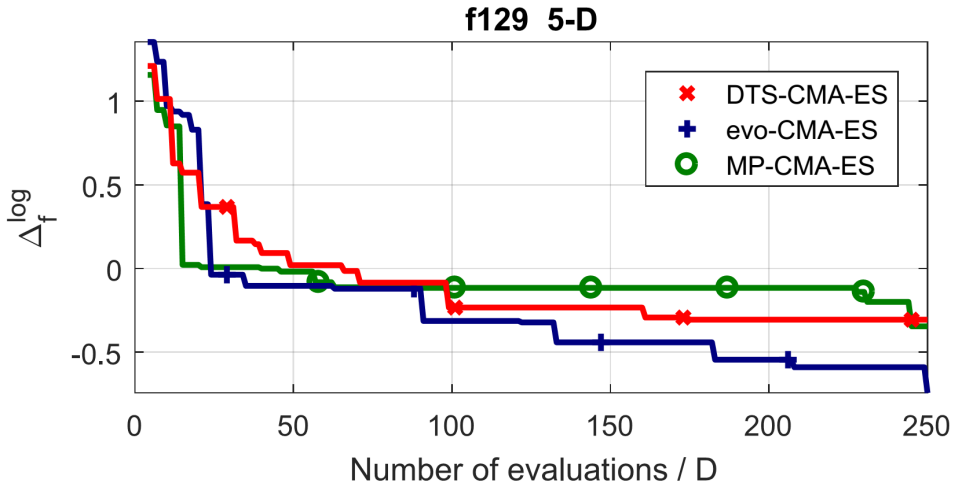


Figure 4.6: Function F129 in 5D

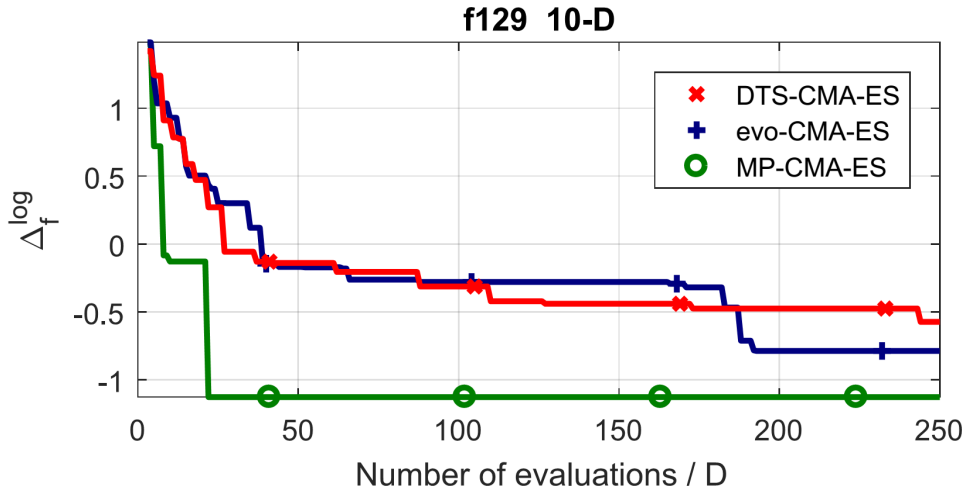


Figure 4.7: Function F129 in 10D

Conclusion

All graphs in the previous chapter shows only one function in given dimension, thus although they may be interesting, to realy compare the algorithms, I needed some more statistics. Statistics, in particular, aggregated functions in 10D are shown in figure 4.11. Graphs for dimensions 2,3 and 5 shows similar performance. These results are very important because they answer the questions asked at the beginning this thesis.

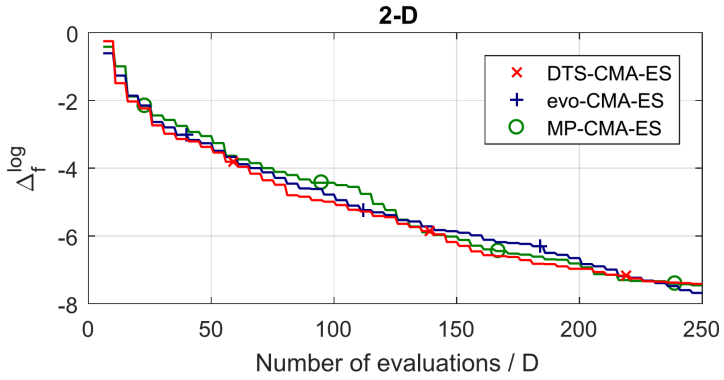


Figure 4.8: All functions in 2D aggregated.

Q: Can CMA-ES handle noisy functions ?

A: Yes it can. According to figure 4.11, even several alternatives to CMA-ES reach the given error treshhold 10^{-8}. It should be recalled that these alternatives use surrogate models to reduce the needs for the evaluations of the original fitness function, which restricts the full CMA-ES potencial.

Q: Which alternative to the CMA-ES is better ?

A: It depends on the considered problem. If time is important, I suggest to use the original DTS-CMA-ES. After years of experimenting and researching,

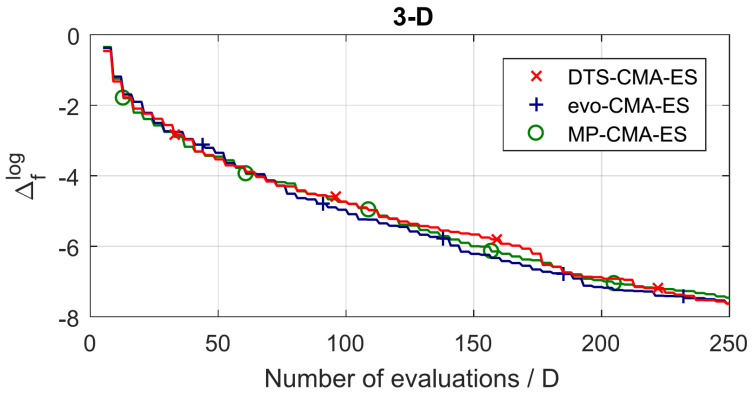


Figure 4.9: All functions in 3D aggregated.

a very good surrogate GP model has been found, that should perform well in many situations. The MP-CMA-ES may be helpful, if surrogate models in modelPool meta-models happen to match evaluated function. Another alternative is the evo-CMA-ES, which can be used to prepare models for the MP-CMA-ES. The evo-CMA-ES is similar in functionality to the MP-CMA-ES. The main difference is its potential to evolve new, better models to fit current needs. Disadvantage of the evo-CMA-ES is a higher computional time demand, due to the need to retrain and validate several models (unlike in DTS-CMA-ES) and due to recombination. However, the time consumption is balanced by far greater improvement potencial.

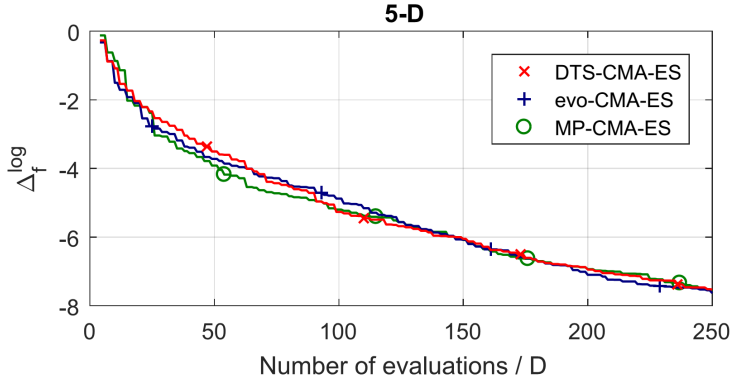


Figure 4.10: All functions in 5D aggregated.

Q: What does this thesis suggest for the future research ?

A: In my opinion, the DTS-CMA-ES almost depleted its improvement potencial, and any further improvements will be very difficult. MP-CMA-ES could be improved by a better algorithm for choosing surrogate models. On

the other hand, evo-CMA-ES can be improved in many ways. I suggest to try some sort of tabu-search to prevent cycling and modify the fitness function by dividing fitness by number of the exact same models, to better maintain diversity on population.

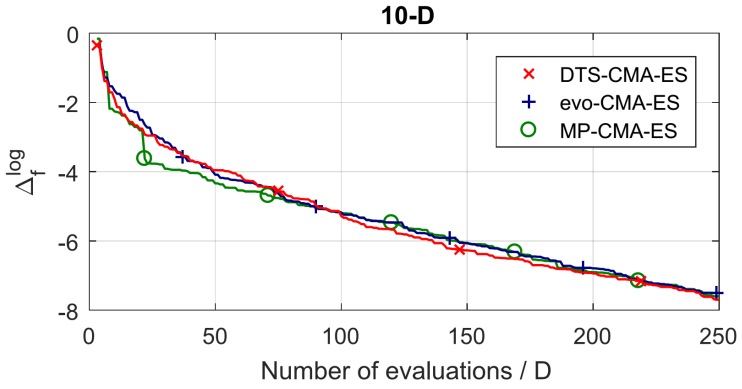


Figure 4.11: All functions in 10D aggregated.

Bibliography

[1] optimization. [online], Last visited 8. 1. 2018. Dostupné z: http://www.businessdictionary.com/definition/optimization.html

[2] Evolution strategy. [online], Last visited 8. 1. 2018. Dostupné z: https://en.wikipedia.org/wiki/Evolution_strategy

[3] Evolutionary Algorithms 3 Selection. [online], Last visited 8. 1. 2018. Dostupné z: http://www.geatbx.com/docu/algindex-02.html

[4] Elitism. [online], Last visited 8. 1. 2018. Dostupné z: https://watchmaker.uncommons.org/manual/ch03s06.html

[5] Genetic Algorithms - Mutation. [online], Last visited 8. 1. 2018. Dostupné z: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

[6] Black Box Optimization Competition. [online], Last visited 8. 1. 2018. Dostupné z: https://bbcomp.ini.rub.de/

[7] COmparing Continuous Optimisers: COCO. [online], Last visited 8. 1. 2018. Dostupné z: http://coco.gforge.inria.fr/

[8] Surrogate Modeling and Optimization. [online], Last visited 8. 1. 2018. Dostupné z: https://www.ibcn.intec.ugent.be/content/surrogate-modeling-and-optimization

[9] Hansen, N.: The CMA Evolution Strategy: A Tutorial. [online], Last visited 8. 1. 2018. Dostupné z: https://arxiv.org/abs/1604.00772

[10] Bajer, L.; Pitra, Z.; Repický, J.; aj.: Gaussian Process Surrogate Models for the CMA Evolution Strategy. *Evolutionary Computation*, ročník Under review, č. xxx, 2018: s. 1–30, ISSN 1063-6560.

[11] metacentrum. [online], Last visited 8. 1. 2018. Dostupné z: https://metavo.metacentrum.cz/

[12] Bajer, L.: Surrogate-cmaes. [online], Last visited 8. 1. 2018. Dostupné z: https://github.com/bajeluk/surrogate-cmaes

[13] Bagging, boosting and stacking in machine learning. [online], Last visited 8. 1. 2018. Dostupné z: https://stats.stackexchange.com/questions/18891/bagging-boosting-and-stacking-in-machine-learning

[14] Bootstrap aggregating. [online], Last visited 8. 1. 2018. Dostupné z: https://en.wikipedia.org/wiki/Bootstrap_aggregating

Appendix **A**

The contents of the enclosed CD

- `readme.txt` ..... brief description
- `TeX` ..... source code for LaTeX
- `DP-hejlvojt` ..... text of thesis in PDF
- `surrogate-cmaes-evolution` ..... modification of L. Bajer's project
  - `doc` ..... documenation
  - `exp` ..... experiments and scripts to process experiments
    - `experiments` ..... folder with experiments
  - `src` ..... source code
  - `test` ..... tests for debugging