Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS

Differential evolution with adaptive encoding

Author: Bc. Václav Klemš
Supervisor: Ing. Petr Pošík, Ph.D. Prague, 2011

Title: Differential evolution with adaptive encoding

Author: Bc. Václav Klemš

Department: Department of Cybernetics

Supervisor: Ing. Petr Pošík Ph.D.

Supervisor's e-mail address: posik@labe.felk.cvut.cz

Abstract In the present work, a system for adaptive encoding of representation space is applied to Differential Evolution algorithm. At first, the Differential Evolution (DE) and Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) are described. Then, the adaptive encoding system derived from CMA-ES is introduced and applied to DE.
The resulting algorithm DE+AE is tested using platform COCO for variety of setting and compared with original DE and CMA-ES algorithms. Experiments prove that, adaptive encoding improves performance of DE algorithm on non-separable function.

Keywords: numerical optimization, differential evolution, evolution strategy, covariance matrix adaptation, adaptive encoding

---

Název práce: Diferenciální evoluce s adaptivním kódováním

Autor: Bc. Václav Klemš

Katedra (ústav): Katedra kybernetiky

Vedoucí bakalářské práce: Ing. Petr Pošík Ph.D.

e-mail vedoucího: posik@labe.felk.cvut.cz

Abstrakt V předložené práci je algoritmus Diferenciální Evoluce vybaven systémem adaptivního kódovaní (AE) reprezentace optimalizovaného problému. Práce začíná popisem samotného algoritmu Diferenciální Evoluce (DE) a algoritmu CMA-ES, evoluční strategie založené na adaptaci kovariační matice. Poté je popsán mechanismus adaptace kódování odvozený z CMA-ES a ten je poté aplikován do algoritmu DE.
Výsledný algoritmus DE+AE je testován za pomoci platformy COCO pro různá nastavení a porovnán s původním algoritmem DE a CMA-ES. Provedene experimenty ukazují, že adaptivní kódování zlepšuje výkonnost DE algorithmu na neseparovatelných funkcích.

Klíčová slova: numerická optimalizace, diferenciální evoluce, evoluční strategie, adaptace kovarianční matice, adaptivní kódování

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Václav K l e m š
Study programme:	Electrical Engineering and Information Technology
Specialisation:	Cybernetics and Measurement – Artificial Intelligence
Title of Diploma Thesis	Differential Evolution with Adaptive Encoding

Guidelines:

1. Learn about the differential evolution (DE) algorithm and assess its ability to optimize non-separable optimization problems.
2. Learn the principles of coordinate system adaptiation (adaptive encoding, AE) as done imide CMA-ES.
3. Equip basic DE algorithm with the AE part. Compare the resulting DE+AE algorithm with original DE and with CMA-ES using a representative set of benchmark functions. Assess the results statistically and discuss the advantages and disadvantages of the DE+AE algorithm.

Bibliography/Sources: Will be provided by the supervisor.

Diploma Thesis Supervisor: Ing. Petr Pošík, Ph.D.

Valid until: the end of the summer semester of academic year 2011/2012

[illegible]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-

L.S.

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Boris Šimák, CSc.
Dean

Prague, May 18, 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Václav K l e m š

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Kybernetika a měření , blok KM2 – Umělá inteligence

Název tématu: Diferenciální evoluce s adaptivním kódováním

Pokyny pro vypracování:

1. Seznamte se s algoritmem diferenciální evoluce (DE) a posuďte jeho schopnost optimalizovat neseparovatelné optimalizační problémy.
2. Seznamte se s principem adaptace souřadného systému (adaptivním kódováním, AE) použitým v algoritmu CMA-ES.
3. Vybavte algoritmus DE adaptací souřadného systému. Porovnejte výsledný algoritmus DE+AE s algoritmy DE a CMA-ES na reprezentativní sadě testovacích funkcí. Výsledky statisticky vyhodnoťte a zhodnoťte přínos algoritmu DE+AE.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Petr Pošík

Platnost zadání: do konce letního semestru 2011/2012

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

[illegible]

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

[illegible]

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 18. 5. 2011

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Prague, August 8, 2011

Václav Klemš

Foremost, I would like to express great gratitude to Ing. Petr Pošík, PhD. for his excellent leadership, valuable advice and support.
Also, I want to thank my father, uncles Petr and Jan and aunt Růžena for their understanding support and generous patronage during my studies.

CONTENTS

CHAPTER
ONE

INTRODUCTION

1.1. Optimization

Let Θ be the domain of feasible values for a vector $\mathbf{x}$. In optimization problem, values of vector $\mathbf{x} \in \Theta \subseteq \mathbb{R}^n$ are searched to minimize[1] a scalar-valued loss function $f(\mathbf{x})$. Other common names for the loss function $f : \mathbb{R}^n \to \mathbb{R}$ are cost function, objective function, fitness function, or criterion. [15]

Optimization algorithms are classified in variety of ways. One of the main classifications is based on the continuity of the domain of $f(\mathbf{x})$. There are three main groups of optimization problems:

Continuous domain optimization problem when $f(\mathbf{x})$ is defined on continues domain for all elements of $\mathbf{x}$

Discrete domain optimization problem when $f(\mathbf{x})$ is defined on discrete domain for all elements of $\mathbf{x}$

Combination of continuous and discrete domain elements of $f(\mathbf{x})$ in one optimization problem

Another important classification of the optimization problems is based on the amount of information which is known about the fitness function e.g. it's derivatives in arbitrary place, discontinuities etc. In optimization, this knowledge about inner structure of the problem can be used to design a special optimization algorithm suited for given problem. This approach is called white-box optimization. However, many optimization problems have objective function for which no further information is known. In such case, the only obtainable information about the problem is the value of fitness function $f()$ for a given vector $\mathbf{x}$. To deal with

[1]In this work, minimization problem is addressed. However, maximization problem can be converted to minimization trivially by changing the sign of the criterion $min(f(X)) = max(-f(X))$.

this limitation, numerical black-box optimizers are designed to evaluate candidate solutions one by one, choosing them according to a specified strategy.

In general, numerical black-box optimization is solved by two types of algorithm: a local search algorithms which move from one solution to another by applying a local change to current solution and population based methods which perform a parallel search using previous set of solution to generate a new one.

1.2. Aim of the work

Differential evolution (DE) is a genetic algorithm –an example of population based methods– equipped with a special crossover and mutation operators suited for continuous function optimization. It is due to Prince and Storn who used DE [18] to solve the Chebychev polynomial fitting problem in 1995. Since then, DE became favorite and widely used optimization technique mainly because of its simplicity and good performance on wide range of problems. [17]

Adaptive encoding (AE) is a general method that makes the search independent of the coordinate system. It is applicable to any continuous domain search algorithm. In step-wise manner, AE changes the coordinate system and searches for a good encoding of original optimization problem. One popular way of AE is used in evolution strategy with covariance matrix adaptation (CMA-ES). [7]

In this work, an AE derived from CMA-ES is used to adjust the problem representation in DE search. Resulting novel algorithm (DE+AE) is tested on a set of standard benchmark functions featured in COmparing Continuous Optimizers platform (COCO) [1] and results are compared with original DE and CMA-ES.

1.3. Chapter order

In chapter 2 DE algorithm is described and its suitability for optimizing non-separable functions is considered.

In chapter 3 DE algorithm is equipped with a mechanism for adaptation of problem representation (AE). In the first part, the description of framework for adaptive encoding of the search space representation is given. The second part of the chapter is devoted to description of an updating procedure $\mathrm{AE}_{\mathrm{CMA}}$-Update which is a part of the framework. The $\mathrm{AE}_{\mathrm{CMA}}$-Update is based on principles employed in CMA-ES algorithm. Thus, the CMA-ES algorithm is explained first and $\mathrm{AE}_{\mathrm{CMA}}$-Update is introduced after that.

In chapter 4 the performance of resulting novel algorithm DE+AE is compared with original DE and CMA-ES algorithms. At first, a short description of testing framework COCO and it's setting is provided. Afterward, experiments for variety of setting are described and its results are assessed.

CHAPTER
TWO

DIFFERENTIAL EVOLUTION

Differential evolution (DE) is a simple population based algorithm for global optimization of continuous multi-modal functions of real variables. DE requires few control parameters, shows robust convergence quality and has proven its utility in many real-life and theoretical problems [17].

2.1. Basic DE

As any other evolution strategy algorithm [3], DE starts with a population of ε individuals each represented by n-dimensional real number vector. This initial population is randomly sampled from the entire search domain given by upper and lower bound for every search dimension. Population evolves in discrete time steps called generations. Thus, the i-th individual in population is denoted as vector $\mathbf{x}_i(t) = [x_{i,1}(t), x_{i,2}(t), ..., x_{i,n}(t)]$ where $t = 0, 1, 2, ..., t, t+1, ...$ denotes generation. Those vectors are often referred as "genomes" or "chromosomes" pointing back to its inspiration in natural evolution process. Every individual is assessed with real-valued fitness $f(\mathbf{x}_i(t))$, where $f()$ is objective function of the optimization problem.

In every iteration, new set of individuals –offspring– is generated using crossover and mutation. The number of offspring individuals is usually the same as number of individuals in population ε. Afterwards, fitness of each new individual is evaluated and based on its quality, a competition is carried out between parental individuals and offspring. Individuals with better fitness are more likely to be placed in next generation of ε individuals, so that, this competition stands for process of natural selection.

Genetic algorithms differ in operators of mutation, crossover and selection. For DE, a new differential mutation operator was designed, so it is suitable for optimizing functions of real variables. Classical DE creates a donor vector $\mathbf{v}_i(t)$ for every parental individual i. For its construction, three other randomly chosen vectors r_1, r_2 and r_3 are combined in following manner:

$$\mathbf{v}_i(t) = \mathbf{x}_{r1}(t) + g \cdot (\mathbf{x}_{r2}(t) - \mathbf{x}_{r3}(t)) \tag{2.1}$$

A scalar number $g > 0$ is one of DE's tuning parameters. Its value is constant and chosen usually within the range $[0,2]$ [16]. Donor vector $\mathbf{v}_i(t)$ has elements $v_{i,j}(t)$ where $j = 1,...,n$. Afterwards, for sake of higher diversity in population, crossover on every offspring individual is performed. In DE, two schemes of crossover are used most often - "binomial" and "exponential". The crossover method is not so important although Ken Price claims that *"binomial is never worse than exponential"* [17].

Before crossover a new - trial vector $\mathbf{u}_i(t) = \mathbf{x}_i(t)$ is assigned to every donor vector $\mathbf{v}_i(t)$. During crossover, elements of $u_{i,j}(t)$ are selected belong chosen crossover scheme and replaced by elements of respective donor vector $v_{i,j}(t)$. Binomial crossover generates a random number $rn \in [0,1]$ for every element and performs crossover when this random number is smaller then a scalar number cr which is another tuning constant of DE. Elements of trial vector $u_{i,j}$ are then given as:

$$\begin{aligned} u_{i,j}(t) &= v_{i,j}(t) \text{ if } rn < cr \\ &= x_{i,j}(t) \text{ else} \end{aligned}$$

Exponential crossover selects for replacement a random coherent block of elements. At first, two random integers are generated - the starting point s drawn from uniform discrete distribution $\mathcal{U}(0,n)$ which marks the beginning of the replaced block in trial vector and length of the block $l \in [1, n+1]$ drawn using the following scheme:

```
$l = 0$
$do\{$
 $l = l + 1$
$\}$while $(rn < cr)$ AND $(l < n)$
```

Elements of trial vector $u_{i,j}$ are then given along exponential crossover methods as:

$$\begin{aligned} u_{i,j}(t) &= v_{i,j}(t) \text{ for } j =< s >_n, < s+1 >_n, < s-l+1 >_n \\ &= x_{i,j}(t) \text{ for other } j \end{aligned}$$

The brackets $<>_n$ denote a modulus function with modulus n. After mutation and selection the trial vectors $\mathbf{u}_i(t)$ are already finished offspring individuals.

Last step in processing toward new generation is selection. Common selection method in DE is based on direct comparison which is performed between each

Algorithm 2.1 Classical DE.

```
1. initialize $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\varepsilon\} \in \mathbb{R}^n$ (a set of solutions)
2. $f_i = f(\mathbf{x}_i)$                for $i = 1, ..., \varepsilon$
3. repeat:
4.    $\mathbf{v}_i = \text{create\_donor}(\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\varepsilon\})$   for $i = 1, ..., \varepsilon$
5.    $\mathbf{u}_i = \text{crossover}(\mathbf{x}_i, \mathbf{v}_i)$                for $i = 1, ..., \varepsilon$
6.    $f_i^{offs} = f(\mathbf{u}_i)$                           for $i = 1, ..., \varepsilon$
7.    $\{\mathbf{x}_i, f_i\} = \text{selection}\left(\mathbf{x}_i, \mathbf{u}_i, f_i, f_i^{offs}\right)$   for $i = 1, ..., \varepsilon$
8. until stopping criteria is met
```

couple of parental individual $\mathbf{x}_i(t)$ and its particular offspring vector $\mathbf{u}_i(t)$. Thus, individuals advance into next generation in accordance with decision rule:

$$\begin{aligned} \mathbf{x}_i(t+1) &= \mathbf{u}_i(t) \text{ if } f(\mathbf{u}_i(t)) > f(\mathbf{x}_i(t)) \\ &= \mathbf{v}_i(t) \text{ if } f(\mathbf{u}_i(t)) < f(\mathbf{x}_i(t)) \end{aligned}$$

In result, new generation of ε individuals with better fitness is evolved and DE continues to next iteration.

The pseudo-code of DE algorithm is given in Alg. 2.1.

2.2. Variations of DE

The authors of DE Storn and Price themselves suggested whole family of variants of DE [20]. Standard notation to determine particular variation of DE has form DE/*x*/*y*/*z*. In this notation *x* is a string denoting which vector from the parental populations is chosen for perturbation during mutation. This vector can be chosen randomly (rand) or the best vector from population with respect to fitness value is chosen (best). Different combinations of best and random vectors are also feasible. *y* is the number of difference vectors used for perturbation of vector *x*. Finally, *z* determines the type of crossover (exponential or binomial).

In this notation, the classical variant of DE is denoted as DE/rand/1 with mutation in Eq. (2.1). Other common variation of DE is DE/best/1 with mutation operator:

$$\mathbf{v}_i(t) = \mathbf{x}_{\text{best}}(t) + g \cdot (\mathbf{x}_{r2}(t) - \mathbf{x}_{r3}(t))$$

where instead of random individual $\mathbf{x}_{r1}(t)$ the individual with best fitness $\mathbf{x}_{\text{best}}(t)$ is used.

For purpose of this work, a new mutation operator was created.

$$\mathbf{v}_i(t) = \mathbf{x}_{\text{avg}}(t) + g \cdot (\mathbf{x}_{r2}(t) - \mathbf{x}_{r3}(t))$$

Again, it is similar to Eq. (2.1) but instead of variation of $\mathbf{x}_{r1}(t)$, the new vector is used:

$$\mathbf{x}_{\text{avg}}(t) = \sum_{i=1}^{\varepsilon} \mathbf{x}_i(t)\, w_i$$

where

$$\begin{aligned} w_{i=1,\dots,\varepsilon} &= \frac{\ln(\mu+1)-\ln(i)}{\sum_{j=1}^{\mu}\ln(\mu+1)-\ln(j)} && \text{for } i = 1,\dots,\mu \\ &= 0 && \text{for } i = (\mu+1),\dots,\varepsilon \end{aligned}$$

where $\mu = \lfloor \varepsilon/2 \rfloor$.

The computation of $\mathbf{x}_{\text{avg}}(t)$ is identical with computation of $\mathbf{m}(t+1)$ in AE$_{\text{CMA}}$-Update (Chap. 3), and the motivation to create this mutation operator is to make DE more similar to CMA-ES.

2.3. Choice of parameters

Beside choice of crossover and mutation operator, there are three parameters to be tuned:

Population size ε Usually ranges from $2n$ to $40n$ [14]. According to [17], rising ε above 40 does not improve performance of the algorithm.

Mutation scale factor g Must be strictly greater than zero. Typically ranges between 0.4 and 1. Also, new value of g can be randomly generated for every generation.[17][14]

Crossover probability cr Range $0 \le cr \le 1$. Smaller values are recommended for separable functions. [14]

2.4. DE on separable and non-separable functions

Separable functions allow to reduce the search process on n one-dimensional search problems which makes it easier to be optimized. In contrary, optimizations of non-separable functions is considered to be much more challenging, since many optimization algorithms exploits separability and are not designed for non-separable functions. A rotation matrix $\mathbf{R}$ can be applied to a separable function to obtain a non-separable one. [6]

The crossover operators in DE are rotation dependent. Hence, for non-separable function, DE must rely mainly on rotation invariant mutation operators. This results in significant difficulty when dealing with non-separable functions [19]. Testing of DE on non-separable function is published e.g. in [14].

In this work, adaptive encoding mechanism (Chap. 3) is applied to DE algorithm to adjust the search space representation. As a result, crossover mutation becomes rotation independent and performance of DE algorithm on non-separable functions should improve.

CHAPTER
THREE

ADAPTIVE ENCODING

In search and optimization, way of state space representation is crucial. For a given optimization method, convenient representation can render an optimization problem trivial [7] whereas inappropriate representation can make the same problem practically unsolvable. Therefore, state space representation needs to be considered when an efficient method for solving optimization problem is to be designed.

In purpose of making an algorithm universal or when finding of eligible representation is difficult and not apparent to a man, convenient state space representation can be found in step-wise manner during the search automatically. In practice, variety of successful optimization methods beside iterative optimization steps, also adjusts problem representation according to current search state. The class of algorithms conducting implicit representational change contains e.g. quasi-Newton methods, covariance matrix adaptation (CMA), or estimation of distribution algorithm (. [7]

Adaptive encoding mechanism used in this work has proven to be beneficial also when applied to Coordinate Descent algorithm in work [13].

3.1. General Adaptive-Encoding Framework

In his research report from 2008 [7], Nikolaus Hansen describes a general framework for an iterative incremental representation change, adaptive encoding (GAEF) and extracts an universal update rule ($\mathrm{AE}_{\mathrm{CMA}}$-Update) based on evolution optimization algorithm CMA-ES [8]. This framework considers only linear transformations i.e. rotation and scaling. Search problem and representation problem are decoupled, thus the framework is applicable on any optimization problem in continuous domain. In particular, he expects good results of AE applied on population based optimization algorithms [3].

The principle of GAEF is based on three steps: **encode - decode - update**.

Algorithm 3.1 DE with AE using GAEF.

```
1. initialize $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\varepsilon\} \in \mathbb{R}^n$ (a set of solutions)
2. initialize $\mathbf{B} \in \mathbb{R}^{n \times n}$ (a transformation matrix)
3. $f_i = f(\mathbf{x}_i)$    for $i = 1, ..., \varepsilon$
4. repeat:
5.    $\mathbf{x}'_i = \mathbf{B}^{-1}\mathbf{x}_i$    for $i = 1, ..., \varepsilon$ (encode)
6.    begin
7.       $\mathbf{v}'_i$ = create_donor($\{\mathbf{x}'_1, \mathbf{x}'_2, ..., \mathbf{x}'_\varepsilon\}$)    for $i = 1, ..., \varepsilon$
8.       $\mathbf{u}'_i$ = crossover($\mathbf{x}'_i, \mathbf{v}'_i$)    for $i = 1, ..., \varepsilon$
9.       $f_i^{offs} = f(\mathbf{B}\,\mathbf{u}'_i)$    for $i = 1, ..., \varepsilon$
10.      $\{\mathbf{x}'_i, f_i\}$ = selection$\left(\mathbf{x}'_i, \mathbf{u}'_i, f_i, f_i^{offs}\right)$    for $i = 1, ..., \varepsilon$
11.   end
12.   $\mathbf{x}_i \leftarrow \mathbf{B}\mathbf{x}'_i$    for $i = 1, ..., \varepsilon$ (decode)
13.   $\mathbf{B} \leftarrow$update$\left(\mathbf{B}, \mathbf{x}_{1,...,\mu}\right)$ ($\mathbf{x}_{1,...,\mu}$ is $\mu$ best individuals ordered belong fitness)
14. until stopping criteria is met
```

Alg. 3.1 shows the three steps of GAEF applied on classical DE algorithm as described in chapter 2. In comparison with original DE algorithm as described in Alg. 2.1, all changes are marked with gray background.

The three steps of GAEF are repeated within main loop of original optimization algorithm. At the beginning of every iteration, the entire population is encoded, i.e. transformed into a space representation that should be profitable for the modification operators (Alg. 3.1, line 5). The consecutive step of optimization algorithm is done with encoded population and is slightly changed: The fitness evaluation of newly generated individuals is done in original space thus the offspring must be decoded first and then evaluated on fitness function (Alg. 3.1, line 9). After the optimization step is done, population of individuals advancing into next generation is decoded into original representation in which the fitness function is defined (Alg. 3.1, line 12). Encoding and decoding operations are trivial because linear transformation is implemented by matrix multiplication applied on every individual from current population which are determined by vectors $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\varepsilon\} \in \mathbb{R}^n$. This also implies, that AE is fully described by one matrix $\mathbf{B}$ of size $n \times n$.

The iteration step is finished with the update step (Alg. 3.1, line 13). In this step, the procedure adjusting the transformation matrix $\mathbf{B}$ is executed, hence the transformation is adapted and encoding of search space in next iteration will differ.

In Alg. 3.2, a small change to algorithm Alg. 3.1 is proposed. The function of the algorithm is the same and decoding is performed on offspring population

Algorithm 3.2 DE with AE using GAEF with direct offspring decoding.

```
1. initialize $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\varepsilon\} \in \mathbb{R}^n$ (a set of solutions)
2. initialize $\mathbf{B} \in \mathbb{R}^{n \times n}$ (a transformation matrix)
3. $f_i = f(\mathbf{x}_i)$                    for $i = 1, ..., \varepsilon$
4. repeat:
5.     $\mathbf{x}'_i = \mathbf{B}^{-1}\mathbf{x}_i$        for $i = 1, ..., \varepsilon$ (encode)
6.     begin
7.         $\mathbf{v}'_i = \text{create\_donor}(\{\mathbf{x}'_1, \mathbf{x}'_2, ..., \mathbf{x}'_\varepsilon\})$    for $i = 1, ..., \varepsilon$
8.         $\mathbf{u}'_i = \text{crossover}(\mathbf{x}'_i, \mathbf{v}'_i)$              for $i = 1, ..., \varepsilon$
9.         $\mathbf{u}_i \leftarrow \mathbf{B}\,\mathbf{u}'_i$                           for $i = 1, ..., \varepsilon$ (decode offspring)
10.        $f_i^{offs} = f(\mathbf{u}_i)$                         for $i = 1, ..., \varepsilon$
11.        $\{\mathbf{x}_i, f_i\} = \text{selection}\left(\mathbf{x}_i, \mathbf{u}_i, f_i, f_i^{offs}\right)$    for $i = 1, ..., \varepsilon$ (selection)
12.    end
13.    $\mathbf{B} \leftarrow \text{update}\left(\mathbf{B}, \mathbf{x}_{1,...,\mu}\right)$ ($\mathbf{x}_{1,...,\mu}$ is $\mu$ best individual ordered belong fitness)
14. until stopping criteria is met
```

before fitness evaluation (Alg. 3.2, line 9). However, in contrary to Alg. 3.1, selection is done with decoded individuals and resulting new population doesn't have to be decoded anymore. Thus, execution of one decoding on population is saved during every iteration.

3.2. AE$_{\text{CMA}}$ Updating Rule

In previous section, the general framework for adaptive encoding of state space was introduced. It allows to transform search space linearly and adapt this transformation after every iteration of arbitrary optimization algorithm. Linear transformation described by matrix **B** is separable into two operations: rotation and scaling. The remaining question is how to determine the transformation matrix **B** so the rotation and scaling of original search space assists optimization algorithms to perform better.

At this point CMA-ES can serve as an inspiration for an universal updating rule [7]. This subsection starts with description of CMA-ES and AE$_{\text{CMA}}$-Update rule is explained subsequently.

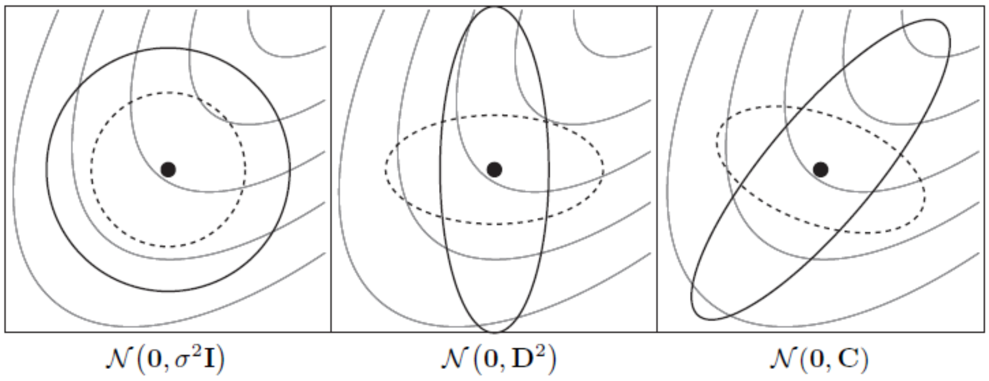
$\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$
$\mathcal{N}(\mathbf{0}, \mathbf{D}^2)$
$\mathcal{N}(\mathbf{0}, \mathbf{C})$

Figure 3.1.: Six ellipsoids, depicting lines of equal density of six different normal distributions, where $\sigma = \mathbb{R}^+$, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{C}$ is a positive definite full covariance matrix. Thin lines depict exemplary objective function contour lines.
Nikolaus Hansen. The CMA evolution strategy: A comparing review, 2006. [8]

3.2.1. Covariance Matrix Adaptation - Evolution Strategy

CMA-ES [8, 10] is based on the concept of self-adaptation in evolution strategies. It is originally suited for small population sizes. The algorithm adapts the mean and covariance matrix of a multi-variate normal distribution, see Fig. 3.1. It efficiently minimizes uni-modal objective functions and in particular is superior on ill-conditioned and non-separable problems. It was successfully applied to a considerable number of real world problems. [10]

The following description of CMA-ES is extract of *Nikolaus Hansen. The CMA evolution strategy: A comparing review, 2006. chap.: 2 - 4* [8]. The notation has been changed slightly.

CMA-ES works in an iteration-wise manner. Every iteration t, new set of λ individuals is generated from multi-variate normal distribution:

$$\mathbf{x}_i(t) \sim \mathcal{N}(\mathbf{m}(t), \sigma(t)^2\mathbf{C}(t)) \quad \text{for } i = 1, ..., \lambda$$

with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. The matrix $\mathbf{C}$ is symmetric and positive definite. $\sigma(t)^2$ is step size control parameter explained later.

All individuals are evaluated on fitness function, and μ best individuals are then used for adaptation of the multi-variate normal distribution. The adjusted distribution $\mathcal{N}(\mathbf{m}(t+1), \sigma(t+1)^2\mathbf{C}(t+1))$ is used for generating a new population in next iteration.

The adaptation is based on several principles described below.

Adjusting the mean $\mathbf{m}$

The mean used in next iteration $\mathbf{m}(t+1)$ is computed as an average of μ best individuals weighted with weights w_i:

$$\mathbf{m}(t+1) = \sum_{i=1}^{\mu} w_i \mathbf{x}_i(t) \tag{3.1}$$

This step substitutes the principles of selection and recombination from evolution strategies.

Hereinafter, the measure called *variance effective selection mass* μ_{eff} is used:

$$\mu_{\text{eff}} = \sum_{i=1}^{\mu} \left(w_i^2\right)^{-1}$$

Adapting the covariance matrix $\mathbf{C}$

The new covariance matrix $\mathbf{C}(t+1)$ is composed of three weighed parts: the current covariance matrix $\mathbf{C}(t)$, rank-μ-update $\mathbf{C}_\mu(t)$ and cumulation-update $\mathbf{p}_c(t)\mathbf{p}_c(t)^{\mathrm{T}}$. The weighing coefficients q_1, q_2, q_3 are specified below.

$$\mathbf{C}(t+1) = q_1 \mathbf{C}(t) + q_2 \mathbf{C}_\mu(t) + q_3 \mathbf{p}_{\mathrm{c}}(t)\mathbf{p}_{\mathrm{c}}(t)^{\mathrm{T}}$$

Rank-μ-update $\mathbf{C}_\mu(t)$ - Estimation of covariance matrix

An unbiased maximum likelihood estimate of covariance matrix $\mathbf{C}_{\text{emp}}$ given a sample population $\mathbf{x}_i(t)$, $i = 1, ..., \lambda$ is:

$$\mathbf{C}_{\text{emp}}(t) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \left(\mathbf{x}_i(t) - \bar{\mathbf{x}}(t)\right) \left(\mathbf{x}_i(t) - \bar{\mathbf{x}}(t)\right)^{\mathrm{T}} \tag{3.2}$$

where $\bar{\mathbf{x}}(t)$ is $\bar{\mathbf{x}}(t) = \sum_{i=1}^{\lambda} \mathbf{x}_i(t)$.

For update used in CMA-ES a slightly different equation is used. Rank-μ-update $\mathbf{C}_\mu(t)$ is computed as:

$$\mathbf{C}_\mu(t) = \sum_{i=1}^{\mu} w_i \left(\left(\mathbf{x}_i(t) - \mathbf{m}(t)\right) \left(\mathbf{x}_i(t) - \mathbf{m}(t)\right)^{\mathrm{T}} \right) \tag{3.3}$$

Unlike $\mathbf{C}_{\text{emp}}(t)$, the rank-$\mu$-update $\mathbf{C}_\mu(t)$ is computed with weighed elements using weights w_i, while $\mathbf{C}_{\text{emp}}(t)$ considers all vectors with the same weight. Also, $w_i = 0$ for $i > \mu$ thus only μ best samples $x_{1,...,\mu}$ is relevant to the sum. Other remarkable difference between $\mathbf{C}_{\text{emp}}(t)$ and $\mathbf{C}_\mu(t)$ is usage of different mean values. For $\mathbf{C}_{\text{emp}}(t)$ the mean of actual samples $\bar{\mathbf{x}}(t) = \sum_{i=1}^{\lambda} \mathbf{x}_i(t)$ is used while $\mathbf{C}_\mu(t)$ employs the mean value of samples from previous iteration $\mathbf{m}(t)$. Therefore, the estimators $\mathbf{C}_{\text{emp}}(t)$ and $\mathbf{C}_\mu(t)$ can be interpreted differently: while $\mathbf{C}_{\text{emp}}(t)$

estimates the distribution variance within the sampled points, $\mathbf{C}_\mu(t)$ estimates variances of sampled steps.

CMA-ES uses the $\mathbf{C}_\mu(t)$ estimate. $\mathbf{C}_{\text{emp}}(t)$-like estimation, where number of samples smaller then λ is considered, features in Estimation of Multivariate Normal Algorithm (EMNA) [12]. Fig. 3.2 compares the two approaches in an example. CMA-ES increases the variance in direction of the gradient while EMNA decreases variance. It is obvious that CMA-ES is less susceptible to premature convergence and its distribution estimate is more likely to sample better results in next iteration than EMNA.

The $\mathbf{C}_\mu(t)$ estimate is called rank-μ-update because it is of rank $\min(\mu, n)$.

CMA-ES is designated to run with small populations in order to achieve fast search. In contrary, to ensure reliable maximum likelihood estimate of covariance matrix used in rank-μ-update $\mathbf{C}_\mu$(t) in Eq. (3.3) the population needs to contain enough information i.e. to contain enough samples. Solution to this contending demands is to use information comprised in estimates done in previous iterations. Therefore, covariance matrix $\mathbf{C}(t)$ is used for computation of $\mathbf{C}(t+1)$ as well.

Using exponential smoothing, current estimates are given higher weight. Resulting rank-μ-update has form:

$$\begin{aligned}
\mathbf{C}(t+1) &= (1-c_{\text{cov}})\mathbf{C}(t) + c_{\text{cov}}\frac{1}{\sigma(t)^2}\mathbf{C}_\mu(t) \\
&= (1-c_{\text{cov}})\mathbf{C}(t) + c_{\text{cov}}\sum_{i=1}^{\mu} w_i \frac{(\mathbf{x}_i(t)-\mathbf{m}(t))}{\sigma(t)}\frac{(\mathbf{x}_i(t)-\mathbf{m}(t))^{\mathrm{T}}}{\sigma(t)} \\
&= (1-c_{\text{cov}})\mathbf{C}(t) + c_{\text{cov}}\sum_{i=1}^{\mu} w_i \,\mathrm{OP}\left(\frac{(\mathbf{x}_i(t)-\mathbf{m}(t))}{\sigma(t)}\right) \qquad (3.4)
\end{aligned}$$

where $\mathrm{OP}(\mathbf{x}) = \mathbf{x}\mathbf{x}^{\mathrm{T}}$ is outer product, $0 < c_{\text{cov}} \leq 1$ is a learning rate and $\sigma(t)$ is step size control parameter which will be, at the moment, regarded as to be equal to 1 and explained closely later.

Rank-one-update $\mathbf{p}_{\mathrm{c}}(t)\mathbf{p}_{\mathrm{c}}(t)^{\mathrm{T}}$ - Evolution path

In Eq. (3.4), the rank-μ-update does not take advantage of the information about the sign of the step because $\mathrm{OP}(\mathbf{x}) = \mathbf{x}\mathbf{x}^{\mathrm{T}} = \mathrm{OP}(-\mathbf{x})$. In order to improve this, the evolution path $\mathbf{p}_{\mathrm{c}}$ is introduced. Evolution path is a sum of consecutive steps with exponential weighing:

$$\mathbf{p}_{\mathrm{c}}(t) = (1-c_{\mathrm{c}})\mathbf{p}_{\mathrm{c}}(t-1) + \sqrt{c_{\mathrm{c}}(2-c_{\mathrm{c}})\mu_{\text{eff}}}\left(\frac{\mathbf{m}(t+1)-\mathbf{m}(t)}{\sigma(t)}\right) \qquad (3.5)$$

where c_{c} is a learning rate and $\sqrt{c_{\mathrm{c}}(2-c_{\mathrm{c}})\mu_{\text{eff}}}$ is a normalization factor explained closer in [7].

Using evolution path, the rank-one-update has form:

$$\begin{aligned}
\mathbf{C}(t+1) &= (1-\mu_{\text{cov}})\mathbf{C}(t) + \mu_{\text{cov}}\,\mathbf{p}_{\mathrm{c}}(t)\mathbf{p}_{\mathrm{c}}(t)^{\mathrm{T}} \\
&= (1-\mu_{\text{cov}})\mathbf{C}(t) + \mu_{\text{cov}}\,\mathrm{OP}(\mathbf{p}_{\mathrm{c}}(t)) \qquad (3.6)
\end{aligned}$$

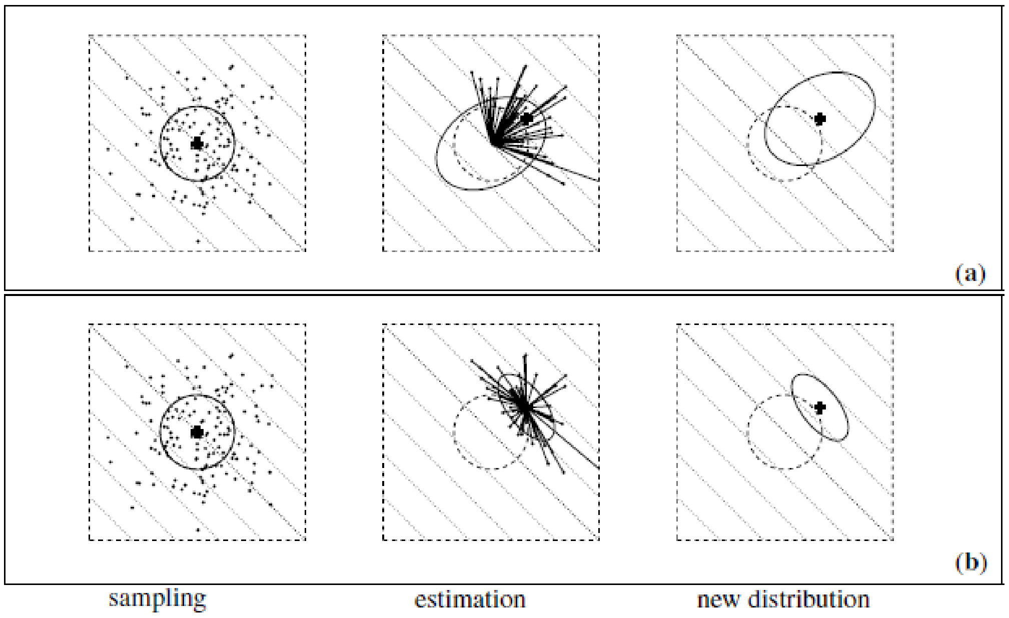

Figure 3.2.: Estimation of the covariance matrix on $f_{\text{linear}}(\mathbf{x}) = -\sum_{i=1}^{2} \mathbf{x}_i$ is to be minimized. Contour lines (dotted) indicate that the strategy should move toward the upper right corner.
(a) Estimation of $\mathbf{C}_\mu(t)$ according to Eq. (3.3), where $\mu = 50$, $w_i = \frac{1}{\mu}$ for $i = 1, ..., \mu$ and $\sigma(t) = 1$.
(b) Estimation used in EMNA featuring $\mathbf{C}_{\text{emp}}(t)$ update, Eq. (3.2), with only 50 best samples considered.
Left: sample of $\lambda = 150\ \mathcal{N}(0, \mathbf{I})$ distributed points. **Middle:** the $\mu = 50$ selected points (dots) determining the entries for the estimation equation (solid straight lines), and the estimated covariance matrix (ellipsoid). **Right:** search distribution of the next generation. Given $w_i = \frac{1}{\mu}$, (a) increases the expected variance in gradient direction for all $\mu < \frac{\lambda}{2}$, while (b) decreases this variance for any $\mu < \lambda$.
Nikolaus Hansen. The CMA evolution strategy: A comparing review, 2006. [8]

Combining rank-one-update and rank-μ-update
The rule for covariance matrix update in CMA-ES exploits both principles captured in Eq. (3.4) and Eq. (3.6) giving the resulting equation:

$$\begin{aligned}\mathbf{C}(t+1) &= (1-c_{\mathrm{cov}})\mathbf{C}(t)+ \\ &+c_{\mathrm{cov}}\left(1-\frac{1}{\mu_{\mathrm{cov}}}\right)\sum_{i=1}^{\mu} w_i \mathrm{OP}\left(\frac{(\mathbf{x}_i(t)-\mathbf{m}(t))}{\sigma(t)}\right)+\frac{c_{\mathrm{cov}}}{\mu_{\mathrm{cov}}}\mathrm{OP}\left(\mathbf{p}_{\mathrm{c}}(t)\right)\end{aligned} \tag{3.7}$$

where c_{cov} and μ_{cov} are learning rates used in coefficients which sum to 1:

$$(1-c_{\mathrm{cov}})+c_{\mathrm{cov}}\left(1-\frac{1}{\mu_{\mathrm{cov}}}\right)+\frac{c_{\mathrm{cov}}}{\mu_{\mathrm{cov}}}=1$$

.

Cumulative step-size control σ

The last part which completes description of CMA-ES defines the step-size control parameter $\sigma(t)$ and its purpose. The motivation for using this parameter comes from the cause, that the covariance matrix adaptation rule in Eq. (3.7) does not explicitly control the “overall scale” of the distribution. In Eq. (3.7), the scale is increased for each selected step while decreasing of the scale is done only implicitly by exponential declination of old information via the factor c_{cov}. [9] A closer view to this goes beyond the purpose of this work and is given in [8, 9].

Fig. 3.3 shows three different evolution paths composed of six steps. It is obvious that the path on the right could be substituted by one longer step. In the contrary, the steps on the left part of the Fig. 3.3 cancel each other out because the algorithm is trying to search within the area adjacent to the initial point. In this case the step-size should be decreased so the algorithm can preform a finer search. In ideal case, consecutive steps of the algorithm are not correlated. The situation in the middle part of Fig. 3.3 is close to the ideal case because successive steps are almost perpendicular.

The evolution path used for step-size control $\mathbf{p}_\sigma$ Eq. (3.8) is called conjugate evolution path.

$$\mathbf{p}_\sigma(t+1)=(1-c_\sigma)\mathbf{p}_\sigma(t-1)+\sqrt{c_\sigma(2-c_\sigma)\mu_{\mathrm{eff}}}\,\mathbf{C}(t)^{-\frac{1}{2}}\frac{\mathbf{m}(t+1)-\mathbf{m}(t)}{\sigma(t)} \tag{3.8}$$

where $\mathbf{C}(t)^{-\frac{1}{2}} \overset{def}{=} \mathbf{B}(t)\mathbf{D}(t)^{-1}\mathbf{B}(t)^{\mathrm{T}}$. Matrix $\mathbf{B}(t)$ is orthogonal basis of eigenvectors, $\mathbf{D}(t)$ is diagonal matrix of square roots of the corresponding positive eigenvalues. $\mathbf{B}(t)$ and $\mathbf{D}(t)$ are obtained by eigendecomposition of covariance matrix $\mathbf{C}(t)=\mathbf{B}(t)\mathbf{D}(t)\mathbf{D}(t)\mathbf{B}(t)^{\mathrm{T}}$.

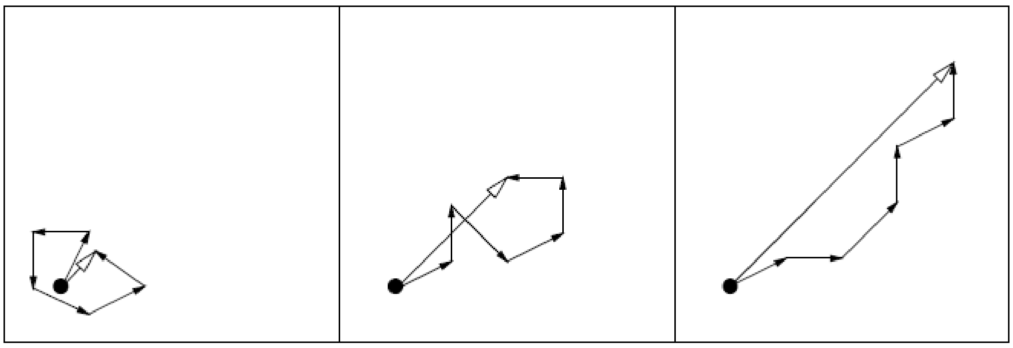

Figure 3.3.: Three evolution paths of respectively six steps from different selection situations (idealized). The lengths of the single steps are all comparable. The length of the evolution paths (sum of steps) is remarkably different and is exploited for step-size control.
Nikolaus Hansen, The CMA Evolution Strategy: A Tutorial, 2008. [9]

The construction of conjugate evolution path $\mathbf{p}_\sigma$ differs from the rank-one-update $\mathbf{p}_c$, Eq. (3.5) by additional transformation factor $\mathbf{C}(t)^{-\frac{1}{2}}$ which scales the size of evolution path in the coordinated system given by $\mathbf{B}(t)$. As a result, expected length of $\mathbf{p}_\sigma$ does not depend on its direction.

The updating rule for $\sigma(t+1)$, Eq. (3.9), compares the length of $\mathbf{p}_\sigma$ with *expected length under random selection* $\mathsf{E}\left(\|\mathscr{N}(0,\mathbf{I})\|\right) = \sqrt{2}\Gamma\left(\frac{n+1}{2}\right)/\Gamma\left(\frac{n}{2}\right) \approx \sqrt{n}\left(1-\frac{1}{4n}+\frac{1}{21n^2}\right)$ [10].

$$\sigma(t+1) = \sigma(t)\exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma(t)\|}{\mathsf{E}\left(\|\mathscr{N}(0,\mathbf{I})\|\right)}-1\right)\right) \tag{3.9}$$

Where factor c_σ/d_σ control the speed of change of σ.

The pseudo-code of CMA-ES algorithm is depicted in Alg. 3.3.

Parameter setting and initial values

CMA-ES is ready-to-use optimization algorithm. Its parameters are determined by empirical studies and users are not supposed to change them. The recommended setting with extended description of parameters is available in [8].

3.2.2. Principle of AE$_{\text{CMA}}$-Update

The AE$_{\text{CMA}}$-Update procedure is in Alg. 3.4. It is based on principles of rank-one-update and rank-μ-update of covariance matrix $\mathbf{C}$ originated in CMA-ES.

Algorithm 3.3 CMA-ES optimization algorithm

```
1. initialize $\lambda$, $\mu$, $w_{i=1,...,\mu}$, $\mu_{\text{eff},}$, $c_\sigma$, $d_\sigma$, $c_c$, $\mu_{\text{cov}}$, $c_{\text{cov}}$
2. initialize $\mathbf{C}(t) \in \mathbf{I}^n$, $\mathbf{m}(t) =$ vector of ones $(n \times 1)$, $\mathbf{p}(t) =$ vector of zeros $(n \times 1)$
3. repeat:
4.    $\mathbf{x}_i(t) \sim \mathcal{N}(\mathbf{m}(t), \sigma(t)^2\mathbf{C}(t))$ for $i = 1, ..., \lambda$
5.    $\mathbf{m}(t+1) = \sum_{i=1}^{\mu} w_i \mathbf{x}_i(t)$
6.    $\mathbf{p}_c(t) = (1-c_c)\mathbf{p}_c(t-1) + \sqrt{c_c(2-c_c)\mu_{\text{eff}}} \left(\frac{\mathbf{m}(t+1)-\mathbf{m}(t)}{\sigma(t)}\right)$
7.    $\mathbf{C}(t+1) = (1-c_{\text{cov}})\mathbf{C}(t) + c_{\text{cov}}\left(1-\frac{1}{\mu_{\text{cov}}}\right)\sum_{i=1}^{\mu} w_i \,\text{OP}\left(\frac{(\mathbf{x}_i(t)-\mathbf{m}(t))}{\sigma(t)}\right) +$
          $+\frac{c_{\text{cov}}}{\mu_{\text{cov}}}\,\text{OP}(\mathbf{p}_c(t))$
8.    $\mathbf{p}_\sigma(t) = (1-c_\sigma)\mathbf{p}_\sigma(t-1) + \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}\,\mathbf{C}(t)^{-\frac{1}{2}}\,\frac{\mathbf{m}(t+1)-\mathbf{m}(t)}{\sigma(t)}$
9.    $\sigma(t+1) = \sigma(t)\,\exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma(t)\|}{\text{E}(\|\mathcal{N}(0,\mathbf{I})\|)} - 1\right)\right)$
10. until *stopping criteria is met*
```

The cumulative step-size control σ is omitted.

The covariance matrix $\mathbf{C}$ is not used for drawing new samples from a multivariate normal distribution. Instead, the eigendecomposition is used to extract eigenvectors and eigenvalues of $\mathbf{C}$. The eigenvectors compose columns of matrix $\mathbf{B}^\circ$ and corresponding eigenvalues are featured in diagonal matrix $\mathbf{D}$.

The eigenvectors are directions in which the problem space is stretched by respective scaling factors given by eigenvalues. The desired transformation of representation space is given by matrix $\mathbf{B}$ obtained as $\mathbf{B} = \mathbf{B}^\circ\mathbf{D}$.

By application of $\mathbf{B}$, vectors in original search space are rotated and scaled in a manner that fits them to the space in which the optimization problem given by fitness function is proposed.

Parameter setting and initial values

For experiments in Chap. 4.3, the recommended values of parameters were chosen. When adjusting the parameters, the weights w_i must sum to 1 and $0 < c_1 + c_\mu \leq 1$. The recommended default setting of AE$_{\text{CMA}}$-Update provided below as well as extensive information on choice of parameters and it's meaning is available in [7].

The recommended default setting:

$$\mu = \lfloor \lambda/2 \rfloor$$

$$w_{i=1,...,\mu} = \frac{\ln(\mu+1) - \ln(i)}{\sum_{j=1}^{\mu} \ln(\mu+1) - \ln(j)} \quad \text{for } i = 1, ..., \mu$$

Algorithm 3.4 AE$_{\text{CMA}}$-Update$\left(\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_\mu\}\right)$
Updates the encoding matrix $\mathbf{B}$ using the μ best-ranked candidate solutions.
(Nikolaus Hansen. Adaptive encoding for optimization. Technical report, Centre de recherche INRIA Saclay – Île-de-France, 2008. [7])

```
1. given parameters $w_i, c_p, c_1, c_\mu$
2. given $\mathbf{m} \in \mathbb{R}^n$, $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{C} \in \mathbb{R}^{n\times n}$ from last iteration
3. let be matrix $\mathbf{B}^\circ$ orthogonal and matrix $\mathbf{D}$ diagonal with diagonal elements sorted in ascending order
4. $\mathbf{m}^- = \mathbf{m}$
5. $\mathbf{m} \leftarrow \sum_{i=1}^{\mu} w_i \mathbf{x}_i$
6. set scalars $\alpha_i \geq 0$, for $i = 1,...,\mu$ cf. Sect.
7. $\mathbf{p} \leftarrow (1-c_p)\mathbf{p} + \sqrt{c_p(2-c_p)}\alpha_0(\mathbf{m}-\mathbf{m}^-)$
8. $\mathbf{C}_\mu = \sum_{i=1}^{\mu} w_i \alpha_i^2 (\mathbf{x}_i - \mathbf{m}^-)(\mathbf{x}_i - \mathbf{m}^-)^T$
9. set scalar $\alpha_p \geq 0$, cf. Sect.
10. $\mathbf{C} \leftarrow \left(1-c_1-c_\mu\right)\mathbf{C} + c_1\alpha_p \mathbf{p}\mathbf{p}^T + c_\mu \mathbf{C}_\mu$
11. $\mathbf{B}^\circ\mathbf{D}\mathbf{D}\mathbf{B}^\circ \leftarrow \mathbf{C}$          (eigendecomposition)
12. optionally normalize $\mathbf{D}$
13. $\mathbf{B} \leftarrow \mathbf{B}^\circ\mathbf{D}$
```

$$\mu_{\text{eff}} = \sum_{i=1}^{\mu} \left(w_i^2\right)^{-1}$$

$$\alpha_c = 1,\ \alpha_p = 1$$

$$\beta = 2$$

$$\alpha_0 = \frac{\sqrt{n}}{||\mathbf{B}^{-1}(\mathbf{m}-\mathbf{m}^-)||}$$

$$\alpha_i = \frac{\sqrt{n}}{\max\left(\frac{l_i}{\beta}, \underset{\text{j}=1,...,\mu}{\text{median}}\left(l_j\right)\right)} \quad \text{for } i = 1,...,\mu$$

$$c_p = \frac{1}{\sqrt{n}}$$

$$c_1 = \frac{0.2\, c_p}{(n+1.3)^2 + \mu_{\text{eff}}}$$

$$\alpha_\mu = 0.2$$

$$c_\mu = \frac{0.2\,\alpha_c\,\mu_{\text{eff}} - 2 + \frac{1}{\mu_{\text{eff}}}}{(n+2)^2 + \alpha_\mu \mu_{\text{eff}}}$$

Initial values for variables are:

$$\mathbf{m} = \textstyle\sum_{i=1}^{\mu} w_i \mathbf{x}_i,\ \mathbf{p} = \text{vector of zeros}\,(n\times 1) \text{ and } \mathbf{C} = \mathbf{I}^n$$

CHAPTER

FOUR

EXPERIMENTS

In this chapter, tests and experiments done within this thesis are described. All tests were performed using the platform called COmparing Continuous Optimisers (COCO) [1] which is described at first section of the chapter. Following sections contain description of experiments and measured results with discussion.

4.1. COmparing Continuous Optimizers platform

Sensible performance assessment of optimization algorithm is a difficult and tedious task. COCO [1] is a platform created to facilitate sound performance quantification and comparison of real-parameter global optimizers. It contains a set of benchmark functions divided into groups according to their characteristics [6]:

- **Separable fcts** - separable functions
- **Moderate fcts** - Functions with low or moderate conditioning
- **Ill-conditioned [6] fcts** - Functions with high conditioning and uni-modal
- **Multi-modal fcts** - Multi-modal functions with adequate global structure
- **Weak structure fcts** - Multi-modal functions with weak global structure

Both noiseless and noisy alternative of the testbed is provided. Any optimization algorithm under review can be implemented using a framework given in included examples. The framework is available in several languages (C, C++, Java and Matlab/Octave). To create an output, gathered results can be further processed (using Python) and visualized. LaTeX templates are available to generate a report reviewing and comparing performance of one, two or more optimizers.

Two GECCO Workshops for Real-Parameter Optimization called Black-Box Optimization Benchmarking (BBOB) 2009 and 2010 has been held based on COCO platform [1] thus the competence of COCO is validated.

4.1.1. COCO parameters setting and restarting of tested algorithms

The preferred setting of COCO was used in order to preserve comparability with other algorithms tested in BBOB. All tests were done on complete set of noiseless functions. Every function was optimized for dimensions $n = 2, 3, 5, 10, 20, 40$. The number of trials on every function and dimension was 15.

Following restarting criteria are implemented inside every tested algorithm. The search is restarted if:

- the algorithm does not improve best found solution f_{best} for 50 iterations
- the algorithm does not improve best found solution f_{best} for 30 iterations and population diversity is low. Specifically, when average variation counted individually for every dimension is lower then 10^{-10}.
  In Matlab: $(\texttt{sum}(\texttt{var}(\texttt{pop}_{\texttt{de}}, 1, 2))/\texttt{DIM} < \texttt{1e}-\texttt{10}$.

The search is restarted until the maximum number of evaluations $\#\text{FE}_{\max}$ is reached or the best found solution is better than the defined precision range of optimum fitness value. The desired fitness value f_{target} which terminates the algorithm is then given as $f_{\text{target}} = f_{\text{opt}} + \Delta f$, where f_{opt} is optimum fitness value for a given benchmark function and Δf is the precision to reach. In this thesis, the recommended final smallest $f_{\text{target}} = f_{\text{opt}} + 10^{-8}$ was used.

The maximum number of evaluations $\#\text{FE}_{\max}$ determines running time of the experiment. $\#\text{FE}_{\max} = 10^4$ was chosen as trade-off value since the running time of most experiments is shorter than 12 hours for this $\#\text{FE}_{\max}$ and obtained results show convincingly the fundamental features of studied algorithms.

Starting populations are sampled uniformly within the recommended range $[-5, 5]$ for every dimension [5].

Tests were done for many different settings of respective algorithm. Every setting had to be tested separately as though it was a self-contained algorithm.

An abbreviation scheme for restart criteria

In order to shorten the description of a restarting rule, a simple abbreviation is used. The abbreviation has form XXiYYv, where XX is first restart criterion (maximum number of iterations when algorithm does not improve f_{best}) and YY is the second restart criterion (maximum number of iteration when f_{best} is not improved and diversity in population is small). When XX=YY, the second restarting criterion does not have influence.

4.1.2. Performance measure

In optimization, the running time is usually measured in number of fitness function evaluations #FE since evaluations of fitness function are usually most costly operation in a single iteration of optimization algorithm.

The performance measure used in COCO is based on #FE. It is called *expected running time* (ERT) [5]:

$$\begin{aligned} \mathrm{ERT}(f_{\mathrm{target}}) &= \frac{\#\mathrm{FEs}(f_{\mathrm{best}} \geq f_{\mathrm{target}})}{\#\mathrm{succ}} && \text{for } \#\mathrm{succ} > 0 \\ &= \infty && \text{otherwise} \end{aligned} \tag{4.1}$$

where $\#\mathrm{FEs}(f_{\mathrm{best}} \geq f_{\mathrm{target}})$ is number of function evaluations summed over all trials such that the best found solution up to now f_{best} has fitness higher than f_{target} and #succ is number of successful trials i.e. trials which reached f_{target}. Thus, it is the average (expected) number of fitness function evaluations before the desired target precision of fitness value is reached for the first time.

4.1.3. Visualization

Results of the experiments described in Sec. 4.3 are visualized in a variety of different graphs and tables generated by COCO. In this subsection, examples of used graphs are given and their meaning are explained. Thus, the flow of the experiment's description is not disrupted by side comments on the visualization means.

Empirical Cumulative Distribution Functions graphs [5]

Fig. 4.1 is an example of Empirical Cumulative Distribution Functions (ECDF)[1] graph for two algorithms ALG0 (+) and ALG1 (○). It exploits the principles of fixed-target (horizontal view) Fig. 4.2.

The left subgraph on Fig. 4.1 is based on fixed-target approach. The target values are given as the tolerance to the optimal fitness value $\Delta f = 10^{\{+1,-1,-4,-8\}}$

[1]Let $x_1, ..., x_n \in \Phi \subseteq \mathbb{R}$ be realizations of random variables. The empirical cumulative distribution function $F_n(x)$ based on $x_1, ..., x_n$ is

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \chi_{A_i}(x)$$

where A_i is subset $A_i = \{x \,|\, x_i \leq x\}$ and $\chi_{A_i}(x)$ is the indicator function:

$$\chi_{A_i}(x) = \begin{cases} 1, & \mathit{when}\; x \in A_i \\ 0, & \mathit{when}\; x \in \Phi \setminus A_i \end{cases}$$

Source: www.planetmath.org [2]

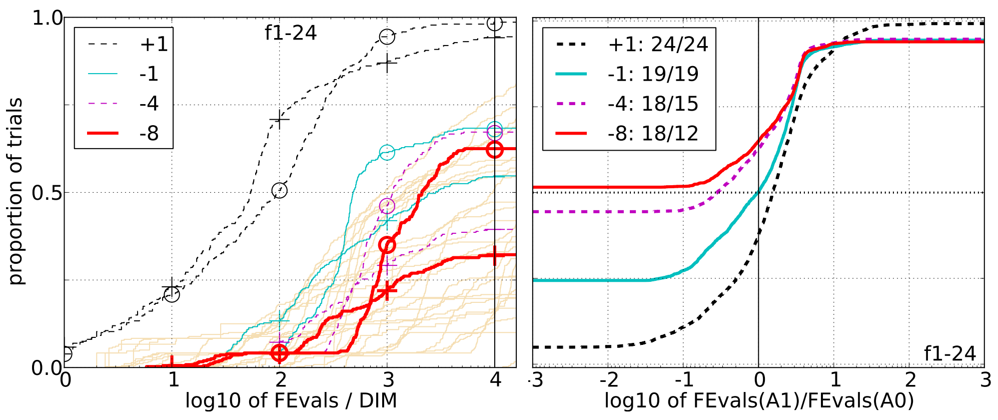

Figure 4.1.: Example of ECDF. The numbers in legend denote how many of testbed functions where optimized to given level of precision Δf in at least one trial.
Generated by COCO [1].

- in graph marked by colour of the line. On y-axis, proportion of trials which reached target value Δf with cost lower than the cost given on x-axis is marked. The cost on x-axis is given as $\log_{10}(\#\text{FE})/n$ [2][3], where #FE is variable. The beige lines on background belong to other algorithms in BBOB 2009 and $\Delta f = 10^{-8}$ (equivalent to red line).

Efficient algorithms optimize functions with smaller #FE. Thus, the more upper-left a line is situated the better performing algorithm it depicts.

The right subgraph on Fig. 4.1 is ECDF of ERT of ALG1 over ERT of ALG0 for different Δf. The more line rises on left half of the graph, the more efficient is the algorithm ALG1 in comparison with ALG0 and vice versa. The difference in performance is smaller when the rise is situated close to the separation border at value 0.

ECDF of the bootstrapped ERT [5]

Fig. 4.3 shows an example graph where several algorithms (marked on the right side of the figure) are compared. For this purposes, ECDF of ERT computed over given dimension for 50 target levels f_{target} in $10^{[-8,2]}$ is used. This computation can be done for all or a subgroup of testbed functions. The beige line corresponds to the algorithms from BBOB 2009 with the best ERT for each of the targets considered. [5]

The actual results were measured on $\#\text{FE}_{\text{max}}$ marked with $\times$. From this mark

[2]#FEs is total number of functions evaluation summed over all trials while #FE is used as number of evaluations in single run.

[3]A notation dissimilarity occurs here. On graph, the dimension of optimization problem is denoted as DIM while, in the text, the letter n is used.

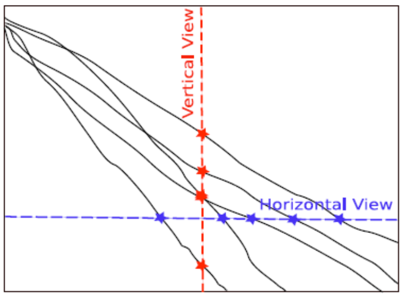

Figure 4.2.: Illustration of fixed-cost (vertical view) and fixed-target (horizontal view). Black lines depict best function value f_{best} versus number of fitness evaluations in several trials.
Real-Parameter Black-Box Optimization Benchmarking BBOB-2010: Experimental Setup [5]

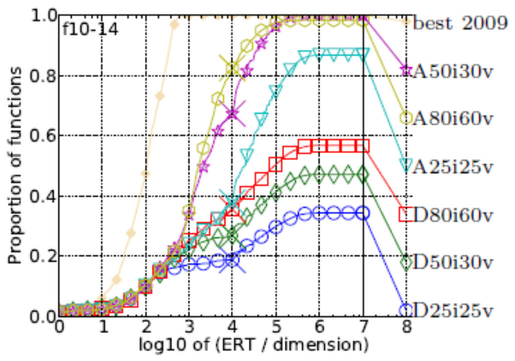

Figure 4.3.: Example graph: Comparison of several algorithm's ECDF using bootstrapping.
Generated by COCO [1].

to the right, estimates of ERT are used. The estimates are computed using bootstrapping method and is claimed to be a good estimate of real ERT [5].

When the #FE$_{\text{max}}$ is reached, bootstrapping method randomly picks trials which were already performed until it chooses one which reached $f_{\text{target}} = f_{\text{opt}} + 10^{-8}$. The ERT is then computed considering #FE which is sum of all function evaluation over all performed and randomly drawn trial. If there is at least one trial which reaches the $f_{\text{target}} = f_{\text{opt}} + 10^{-8}$, the ECDF of ERT estimated in this way reaches the top of the graph. In other case, the line converges to the proportion of target levels which where successfully reached.

4.2. Implementation

The tested algorithms are: DE, DE+AE and CMA-ES. All tests were performed in Matlab using COCO framework. DE algorithm was implemented with respect to easy application of AE. Entire AE is encapsulated in class AdaptiveEncoding.m. The CMA-ES algorithm's implementation was taken from [11] and modified for using with COCO. In order to obtain comparative results to DE and DE+AE, the same restart rules were applied to CMA-ES as well.

4.2.1. Notes on $\mathrm{AE}_{\mathbf{CMA}}$-Update implementation

Several problems appeared during implementation of $\mathrm{AE}_{\mathrm{CMA}}$-Update procedure.

The transformation matrix $\mathbf{B} = \mathbf{B}^{\circ}\mathbf{D}$ needs to be composed of real numbers. For covariance matrixes, the eigenvectors and eigenvalues are real-valued since covariance matrixes are Hermitian. Due to numerical inaccuracy in Matlab, updated matrix $\mathbf{C}$ is not every time symmetric and numbers opposite to each other over diagonal can differ slightly. This occurs mainly for higher dimensions n. Therefore, the symmetry of matrix $\mathbf{C}$ needs to be enforced. In Matlab, this can be done as:

$$\mathtt{C} = \mathtt{tril}(\mathtt{C},0) + \mathtt{tril}(\mathtt{C},-1)' \quad \text{or} \quad \mathtt{C} = 0.5 * (\mathtt{C} + \mathtt{C}')$$

Other problem with numerical instability occurred when inverse of matrix $\mathbf{B}$ is computed. In order to avoid this problem, condition number of matrix $\mathbf{C}$ needs to be limited. The solution used to resolve this problem is coincident with the one in CMA-ES implementation [11]:

```
if max(diag(D)) > 1e14*min(diag(D))
 tmp = max(diag(D))/1e14 - min(diag(D));
 C = C + tmp*eye(DIM);
 D = D + tmp*eye(DIM);
end
```

4.3. Experiments and results

In this subsection, experiments comparing performance of DE, DE+AE and CMA-ES are described.

The default setting for DE and DE+AE is as follows: crossover parameter $cr = 0.5$, and mutation parameter g is drawn from uniform distribution $g = \mathscr{U}(0.5, 1)$ for every generation[17]. Other parameters are set respectively to every group of experiments and are stated in boxes within appropriate subsections.

For most of the tests, the restarting criterion 50i30v is used. The notation convention is described at the end of Sec. 4.1.1. The default values of c_1, c_μ and c_p are given in Sec. 3.2.1.

Datasets with results of the experiments are available at attached CD.

4.3.1. Experiments with population size

Population size is the most important parameter of any population-based algorithm. Fig. 4.4 and Fig. 4.5 show graphs comparing performance of DE and DE+AE with different population sizes.

According to [17], the recommended, most robust population size for DE is $\varepsilon = 10n$ but population bigger then 40 does not substantially improve the convergence. However, smaller values of ε appear to be more efficient during the experiments. This can be a result of more frequent restarts, since smaller population inclines to faster convergence.

The tested sizes of population are $\varepsilon = 6n$, and $3n$. In order to simulate conditions of CMA-ES algorithm, strategy of choosing population size $\varepsilon = 4 + \lfloor 3\ \ln(n) \rfloor$ was taken over from the algorithm and tested as well.

Experimental setting:

$\#\mathrm{FE}_\mathrm{max} = 10^4 n$; $\varepsilon = \{\min(6n, 80), \min(3n, 80), \min(4 + \lfloor 3\ \ln(n) \rfloor, 80)\}$;
mutation:rand/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_\mathrm{p} = default$;

The performance of algorithms with biggest population size $\varepsilon = 6n$ is usually the worst. In small dimensions, $\varepsilon = 3n$ is the best option while the smaller population sizes work better for higher dimensions. This suggests that restarting of algorithm has bigger effect on performance for higher dimensional optimization problems.

Interestingly, the biggest population $\varepsilon = \min(6n, 80)$ does not perform the best on multi-modal functions. This can be caused by the fact, that the population size is not big enough (only the limit value of population size $\varepsilon = 80$ is chosen in case of $n = 20$), or more frequent restarts are more beneficial to performance than big population.

For DE+AE, good performance of small populations is connected with another observation. For estimating covariance matrix, AE needs information obtained from $\mu = \lfloor \lambda/2 \rfloor$ best individuals entering AE_CMA-Update. The more individuals are provided the better estimate of covariance matrix is obtained. Bigger populations provide more information in every generation, therefore AE should work better with them. This could be a reason, why the smallest population size $\varepsilon = \min(4 + \lfloor 3\ \ln(n) \rfloor, 80)\}$ is performing the worst on moderate and ill-conditioned function for dimension 5. However, for dimension 20, the smallest population performs the best. The most likely explanation seems to be that, the benefits of frequent restarting dominate the robustness of bigger population.

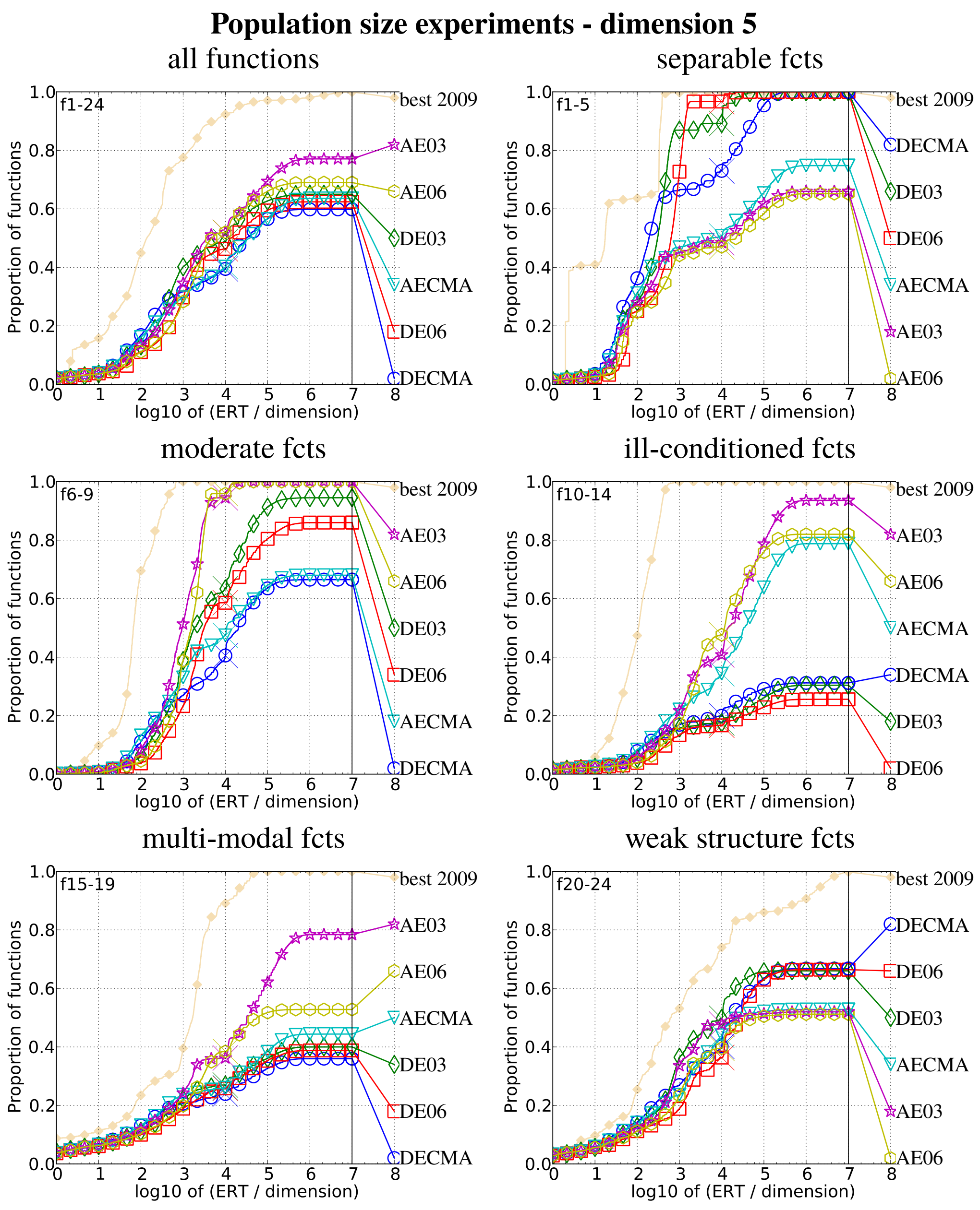

Figure 4.4.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#\mathrm{FE}_{\mathrm{max}} = 10^4 n$;
$\varepsilon = \{\min(6n, 80), \min(3n, 80), \min(3n, 4 + \lfloor 3\ln(n) \rfloor, 80)\}$;
mutation:rand/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_p = default$;
Abbreviations:
DE: DE algorithms; AE: DE+AE;
06: $\varepsilon = 6n$;
CMA: CMA-ES population size strategy $\varepsilon = 4 + \lfloor 3\ln(n) \rfloor$;

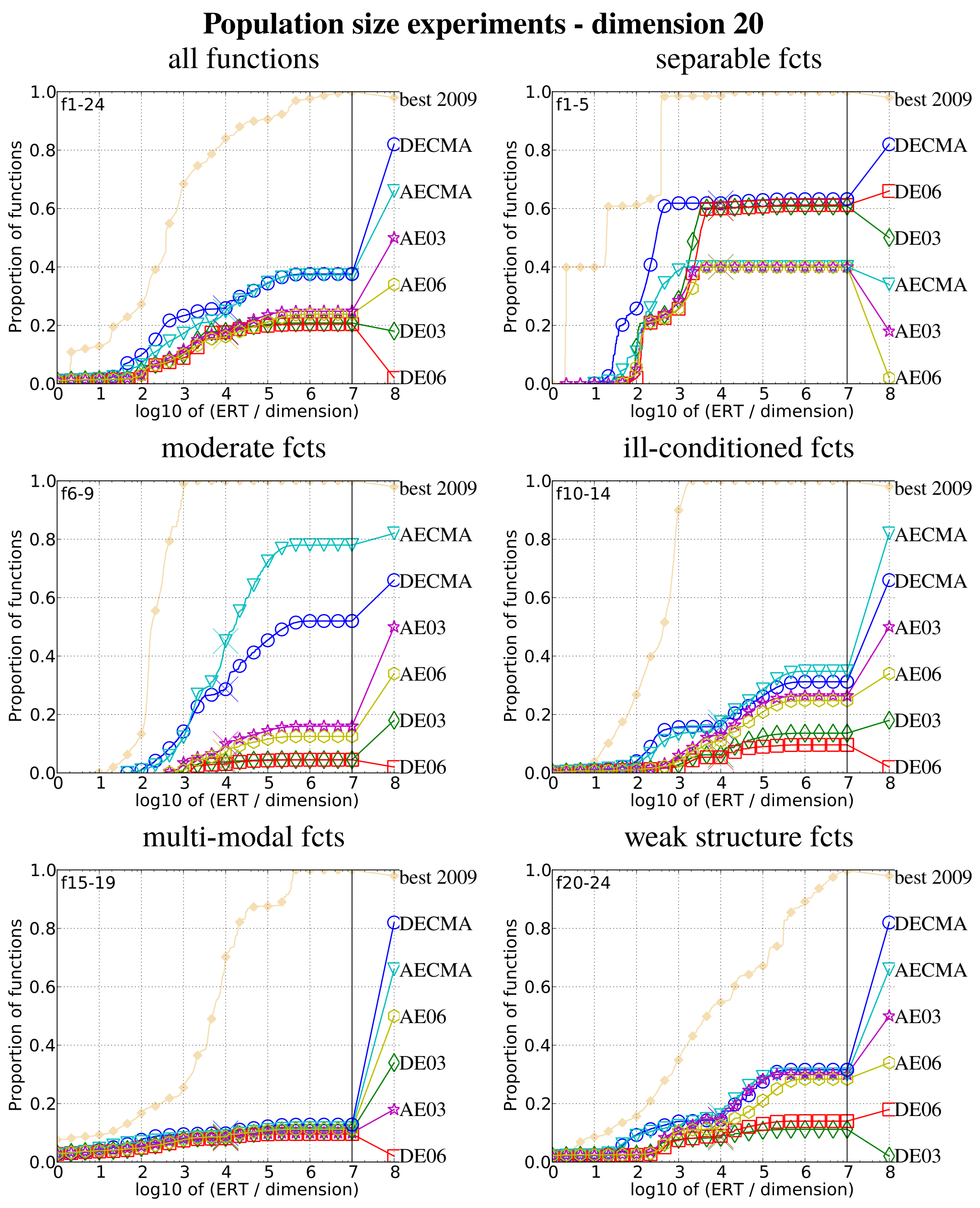

Figure 4.5.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#\mathrm{FE}_{\max} = 10^4 n$;
$\varepsilon = \{\min(6n, 80), \min(3n, 80), \min(3n, 4 + \lfloor 3\ln(n) \rfloor, 80)\}$;
mutation:rand/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_\mathrm{p} = default$;
Abbreviations:
DE: DE algorithms; AE: DE+AE;
06: $\varepsilon = 6n$;
CMA: CMA-ES population size strategy $\varepsilon = 4 + \lfloor 3\ln(n) \rfloor$;

4.3.2. Experiments with crossover operator

A set of experiments was executed in order to compare the performance of the two commonly used crossover operators: binomial and exponential, see. Sec. 2.1. Although, *"The crossover method is not so important"* [17] and the binomial crossover is usually first-option choice.

The results of the experiments for dimensions 5 and 20 are in Fig. 4.6 and Fig. 4.7 respectively.

Experimental setting:

$\#\text{FE}_{\text{max}} = 10^4 n$; $\varepsilon = \min(6n, 80)$
mutation:rand/1 **crossover:**{bin, exp}; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_\text{p} = default$;

As expected, the experiments does not show a significant difference in performance between binomial and exponential crossover. However, for all subgroups of functions, binomial crossover is the best choice for DE+AE. Exponential crossover performs better for DE on separable problems with higher dimension.

Since crossover operation is rotation dependent, DE algorithm can not rely on this operation when optimizing non-separable functions [19]. Significance of crossover operator for separable function comes obvious on higher dimensions where DE/rand/1/exp reached better solution in some trials (the bootstrapped estimate reaches top of the graph).

For DE algorithm, the biggest difference between binomial and exponential crossover appears on functions with low or moderate conditioning and small dimensions. Obviously, DE can still benefit from crossover procedure on those functions since they possess "remains" of separability. In such case, binomial crossover is more suitable. It replaces elements of candidate solution $\mathbf{x}$ by respective individual elements of donor vector (not by blocks elements, see Chap. 2), which leaves more randomness in the process and depends less on particular coordinate system.

The same reason can also explain overall better performance of DE+AE with binomial crossover.

4.3.3. Experiments with mutation operator

Unlike crossover, mutation operator is rotation independent [19]. Therefore, in particular for DE, the choice of mutation operator poses a bigger challenge and supposedly has bigger influence on performance than crossover operator.

Three different mutation strategies were tested: rand/1, best/1, avg/1. Closer explanation is provided in Sec. 2.1. The results of experiments with mutation operators are shown in figures Fig. 4.8 and Fig. 4.9.

Experimental setting:

#FE$_{\max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:{rand, best, avg}/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_\mathrm{p} = default$;

In general, mutation operator rand/1 proves as the worst option. avg/1 mutation performs slightly better than best/1. However, avg/1 mutation does not bring very convincing improvement in comparison to best/1 and those two mutation can be considered to perform similarly well.

The mutation operators best/1 and avg/1 create bigger selection pressure on population and support faster convergence than rand/1 mutation. Thus, the performed search is more local. Interestingly, this not appears to be a disadvantage on multi-modal functions where premature convergence can cause stucking of algorithm in local optima easily.

4.3.4. Experiments with adaptation learning rates

The adaptation learning rates c_1, c_μ and c_p in CMA-ES and AE$_\mathrm{CMA}$-Update adjust the speed of accepting new information into covariance matrix **C**, see Sec. 3.2.1. Therefore, those parameters control the speed of the main function of AE - the ability to adjust coordinate system.

Tuning c_1 and c_μ

The first set of experiments adjusts parameters c_1 and c_μ in DE+AE algorithm (Fig. 4.10 and Fig. 4.11). The default values of those parameters are [7]:

$$c_1 = \frac{0.2\, c_\mathrm{p}}{(n+1.3)^2 + \mu_\mathrm{eff}}; \quad c_\mu = \frac{0.2\, \alpha_c\, \mu_\mathrm{eff} - 2 + \frac{1}{\mu_\mathrm{eff}}}{(n+2)^2 + \alpha_\mu \mu_\mathrm{eff}}$$

The default values are relatively small and prove to be feasible for small dimension. However, in higher dimension, setting bigger c_1 and c_μ brings appreciable improvement to performance.

The merit of increasing value c_1 and c_μ from the default setting is distinct mainly for ill-conditioned functions. This is expectable result since ill-conditioned functions are “highly” non-separable. In such case, the capability of AE to adjust the coordinate system is crucial and increased value of c_1 and $c_\mu$allows AE to do that more promptly.

Experimental setting:

#FE$_{\max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = \{default, 0.1, 0.2, 0.3, 0.4\}$; $c_\mu = \{default, 0.1, 0.2, 0.3, 0.4\}$;
$c_\mathrm{p} = default$;

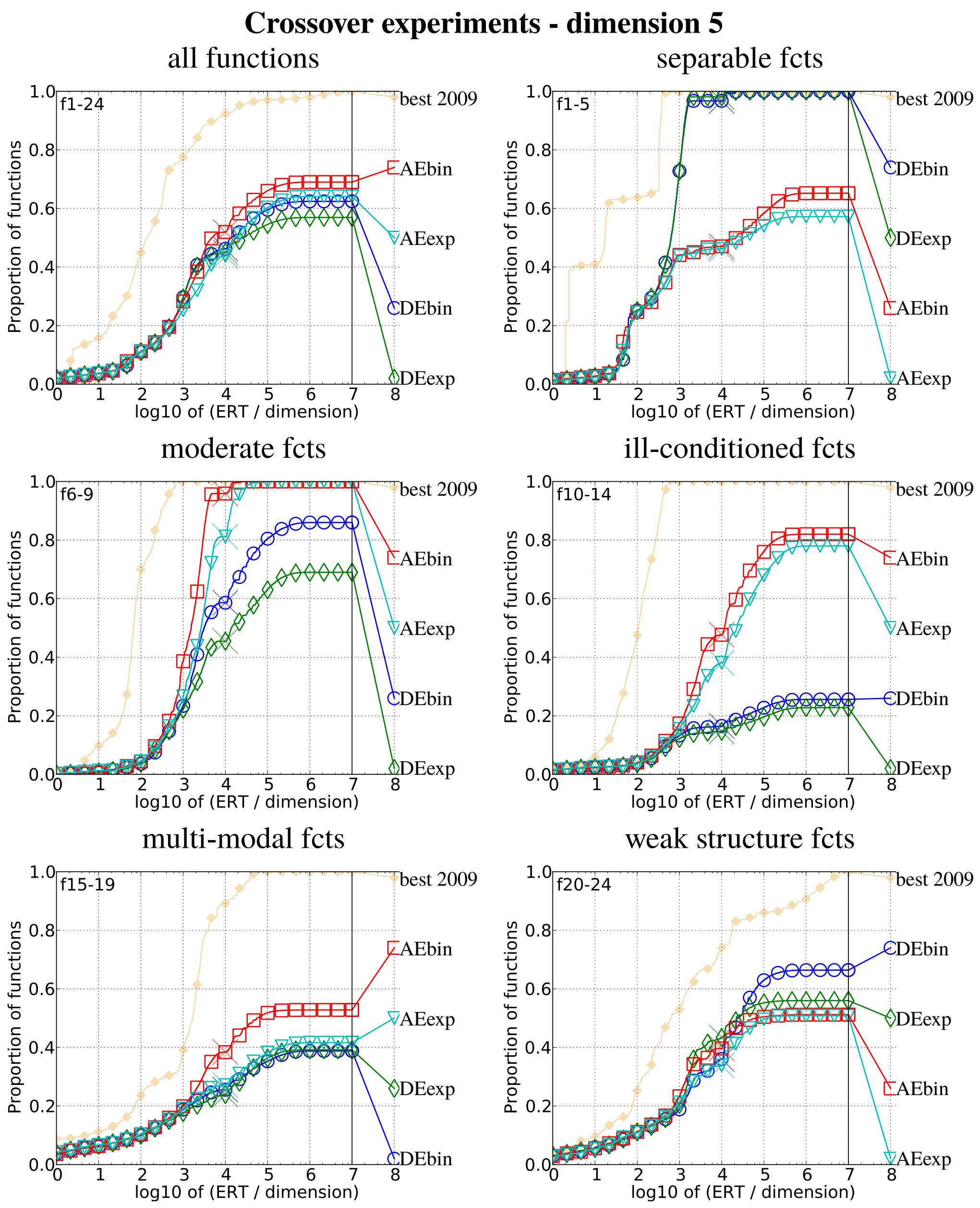

Figure 4.6.: ECDF of the bootstrapped ERT**,** see Sec. 4.1.3
Experimental setting:
$\#FE_{max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:rand/1; **crossover:**{bin, exp}; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_p = default$;
Abbreviations:
DE: DE algorithms; AE: DE+AE;
bin: binomial mutation; exp: exponential mutation;

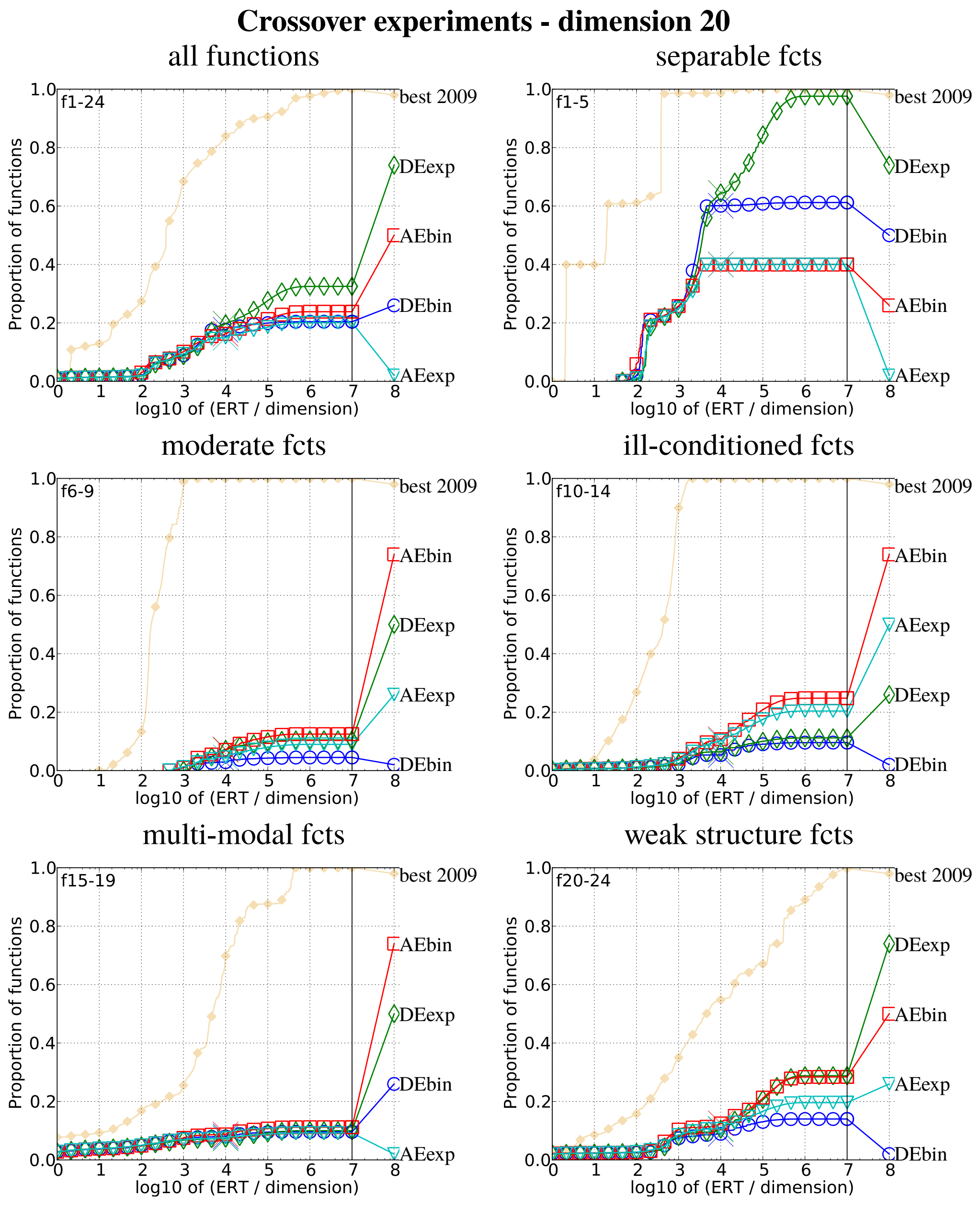

Figure 4.7.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#\text{FE}_{\text{max}} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:rand/1; **crossover:**{bin, exp}; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_\text{p} = default$;
Abbreviations:
DE: DE algorithms; AE: DE+AE; bin: binomial mutation; exp: exponential mutation;

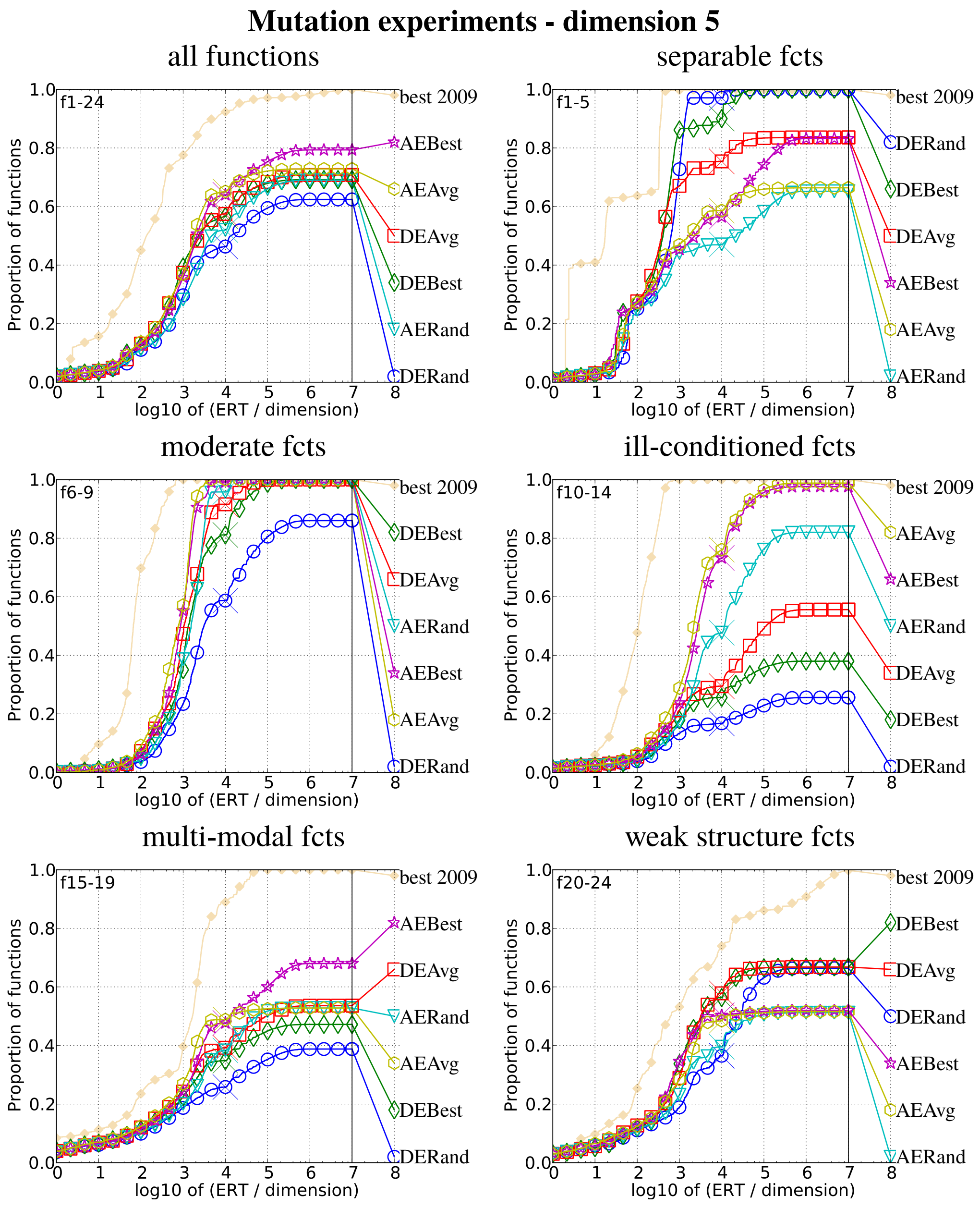

Figure 4.8.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#FE_{max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:{rand, best, avg}/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_p = default$;
Abbreviations:
DE: DE algorithms. AE: DE+AE.
rand: rand/1 mutation. best: best/1 mutation. avg: avg/1 mutation

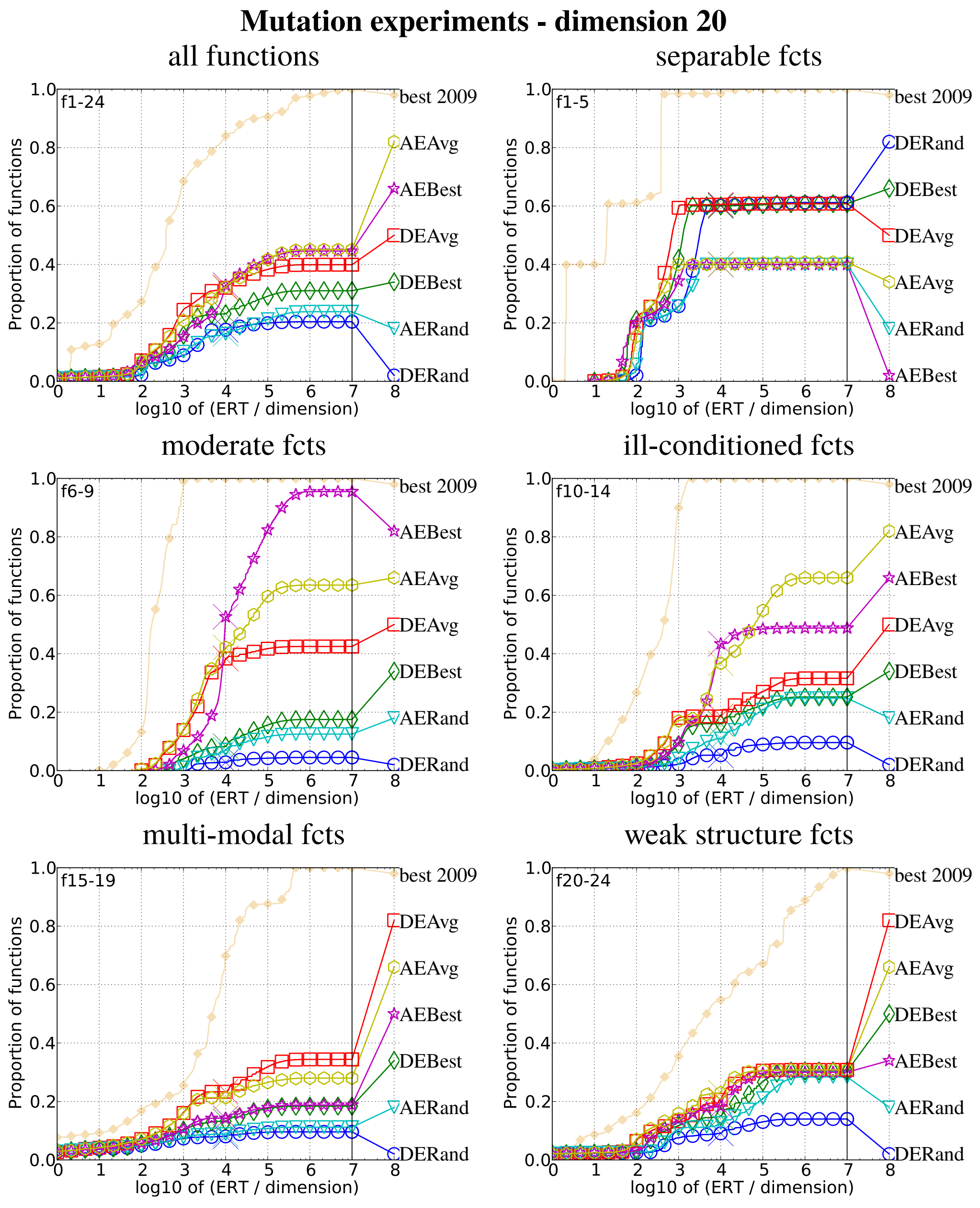

Figure 4.9.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
#FE$_{\max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:{rand, best, avg}/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = default$; $c_\mu = default$; $c_p = default$;
Abbreviations:
DE: DE algorithms. AE: DE+AE.
rand: rand/1 mutation. best: best/1 mutation. avg: avg/1 mutation

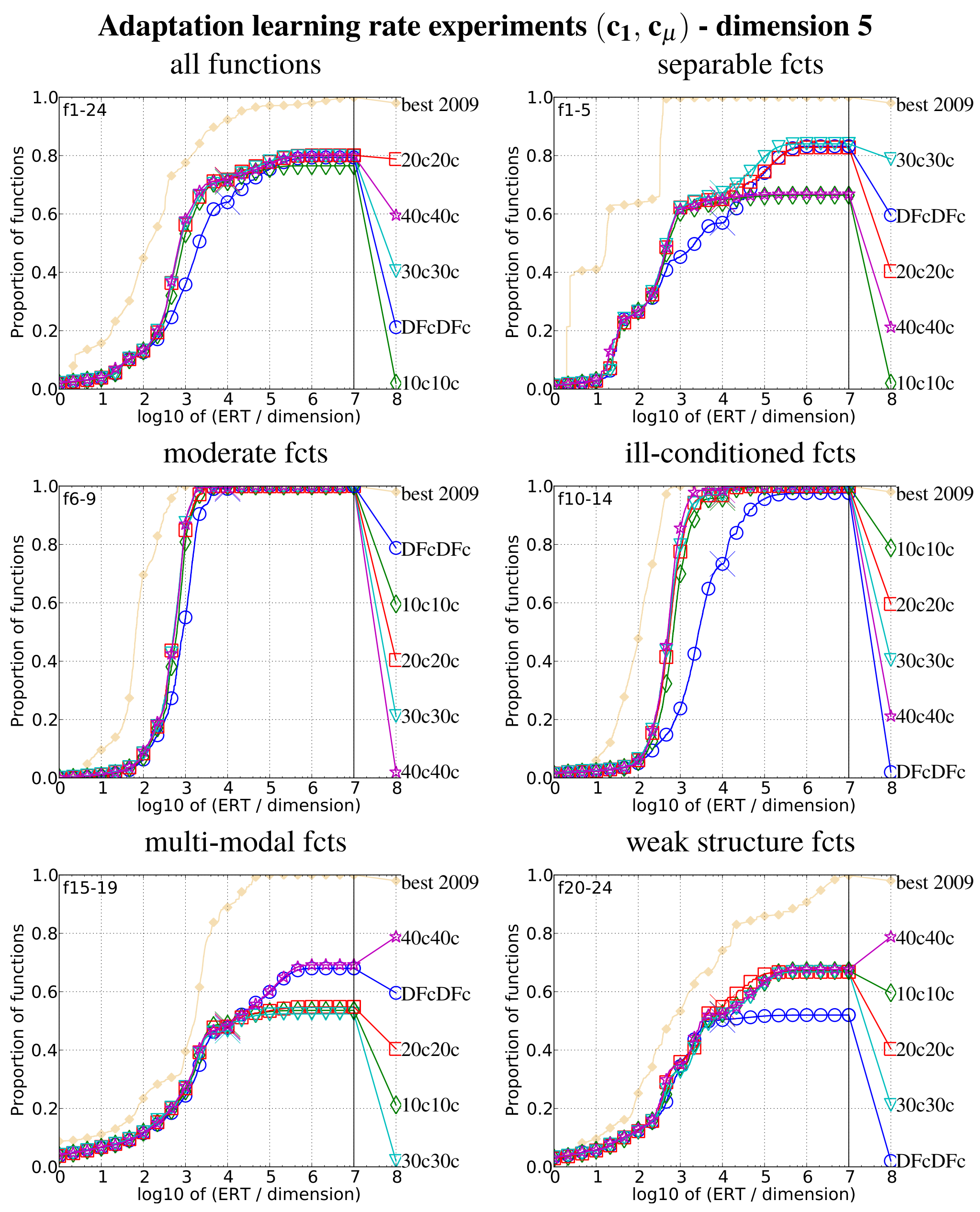

Figure 4.10.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
#FE$_{\text{max}} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = \{default, 0.1, 0.2, 0.3, 0.4\}$;
$c_\mu = \{default, 0.1, 0.2, 0.3, 0.4\}$;
$c_p = default$;
Abbreviations:
D: DE algorithms. A: DE+AE.
XXcYYc: $c_1 = 0.XX$ $c_\mu = 0.YY$
DFcDFc: $c_1 = default$ $c_\mu = default$

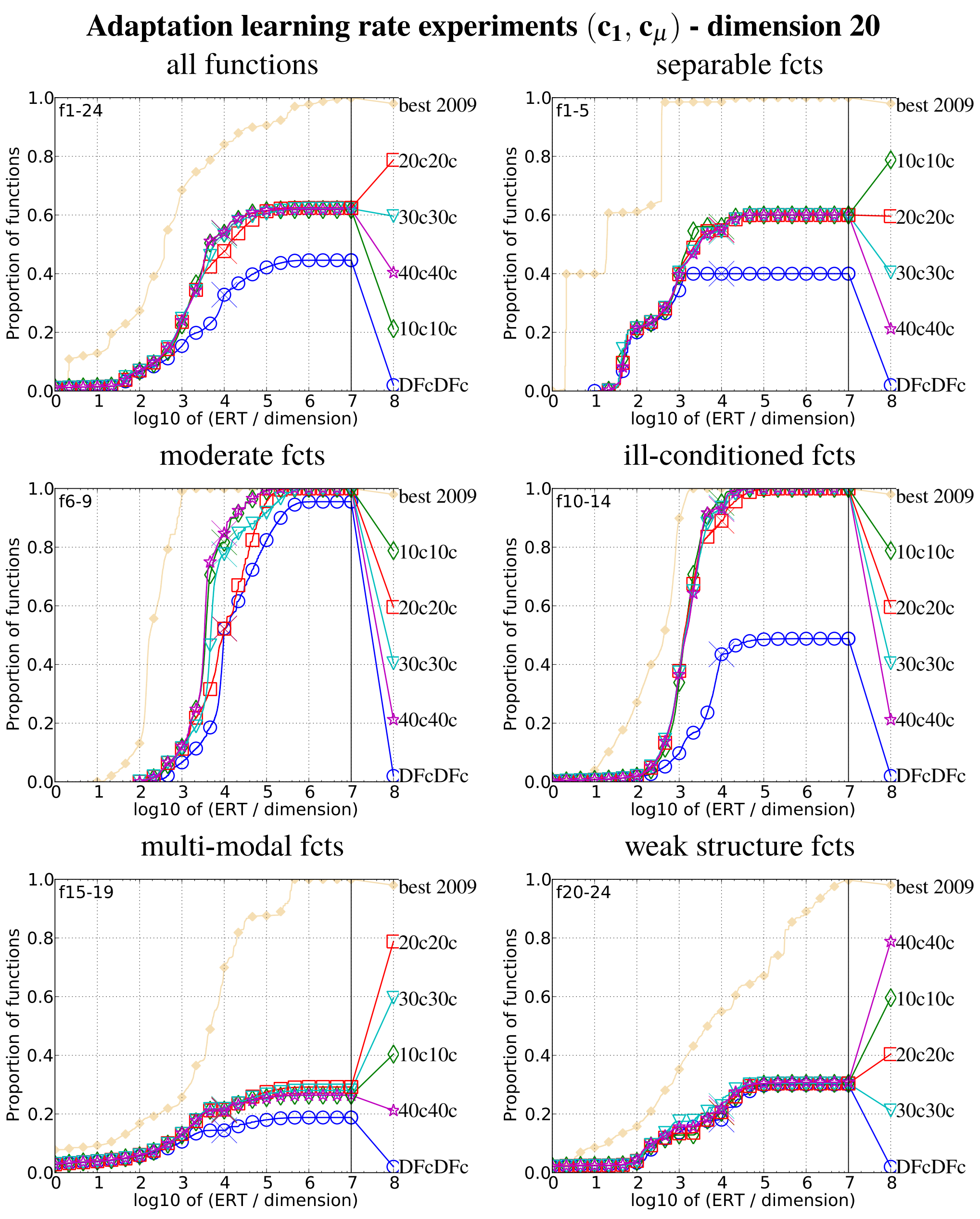

Figure 4.11.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#FE_{max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = \{default, 0.1, 0.2, 0.3, 0.4\}$;
$c_\mu = \{default, 0.1, 0.2, 0.3, 0.4\}$;
$c_p = default$;
Abbreviations:
D: DE algorithms. A: DE+AE.
XXcYYc: $c_1 = 0.XX$ $c_\mu = 0.YY$
DFcDFc: $c_1 = default$ $c_\mu = default$

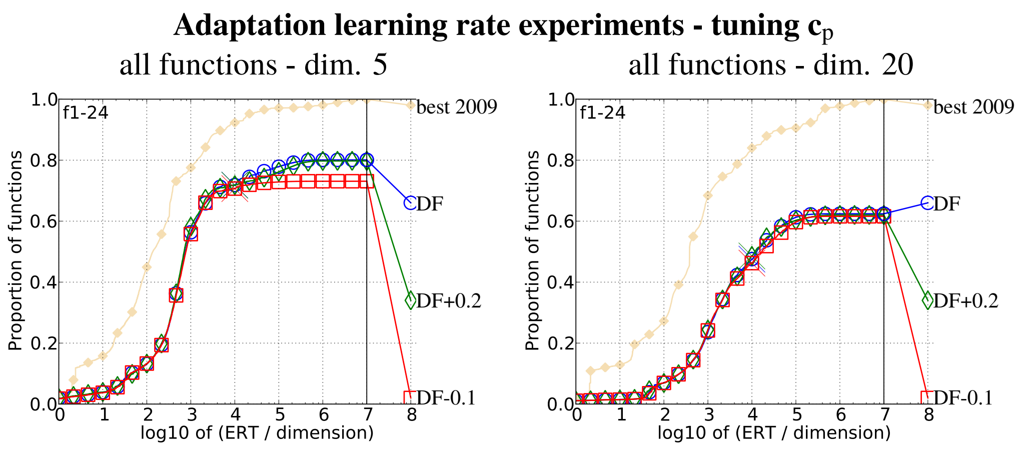

Figure 4.12.: ECDF of the bootstrapped ERT, see Sec. 4.1.3
Experimental setting:
$\#FE_{max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = 0.2$; $c_\mu = 0.2$; $c_p = \{default, default + 0.2, default - 0.1\}$;
Abbreviations:
DF±0.X: $c_p = default + 0.X$

Tuning c_p

Other experiments were done for different values of c_p which controls the learning rate of evolution path **p**, see Sec. 3.2.2. The default value of the parameter is $c_p = \frac{1}{\sqrt{n}}$. In this case setting value of c_p lower or higher than the default value does not have impact on performance of DE+AE algorithm. The result of experiments is visualized in Fig. 4.12.

Experimental setting:

$\#FE_{max} = 10^4 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = 0.2$; $c_\mu = 0.2$; $c_p = \{default, default + 0.2, default - 0.1\}$;

4.3.5. Experiments with restart criterion

An important part of tested algorithms is also the restarting mechanism. In order to assess the influence of restarting on performance, a series of tests was executed. Results of this experiments are in Fig. 4.13 and Fig. 4.14. The notation convention is described at the end of Sec. 4.1.1.

The worst performing restart criterion is 25i25v since it probably does not allow population to converge and the procedure is restarted prematurely. The best performing criterion is 80i60v. However, the performance of restarting criteria 50i30v used for most of the experiments is comparable with 80i60v.

In particular, the difference between 50i30v and 80i60v is small in case of DE+AE. However, in case of ill-conditioned function the criteria 80i60v is obviously superior to 50i60v also for DE+AE. This could be caused by the fact, that AE needs some time to adjust the coordinate system correctly before being beneficial for the search.

This hypothesis is supported also by experiments with value of parameters c_1 and c_μ in Sec. 4.3.4. DE+AE with bigger values of c_1 and c_μ performs better for restart criterion 50i30v (Sec. 4.3.4) than with restart criterion 80i30v and default values of c_1 and c_μ (in this section).

Experimental setting:

$\#\text{FE}_{\text{max}} = 10^4 n$; $\varepsilon = \min(3n, 4 + \lfloor 3\ln(n) \rfloor, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**{25i25v, 50i30v, 80i60v};
$c_1 = default$; $c_\mu = default$; $c_\text{p} = default$;

4.3.6. Comparison of DE, DE+AE and CMA-ES

In order to make a final comparison and conclusion on benefits of applying AE to DE. The algorithm DE and DE+AE were run for $\#\text{FE}_{\text{max}} = 10^5 n$ evaluations. The setting for this long-run experiments was chosen using experience from previous tests. The combination of best performing parameters was chosen for DE algorithm. In case of DE+AE, smaller population sizes than $\varepsilon = \min(6n, 80)$ and restart criteria 80i50v performed better in separated tests. However, the settings do not show good results when used together. Therefore, a tested well-performing combination of population size $\varepsilon = \min(6n, 80)$ and restart criteria 50i30v was chosen instead.

Experimental setup: DE:

$\#\text{FE}_{\text{max}} = 10^5 n$; $\varepsilon = \min(4 + \lfloor 3\ln(n) \rfloor, 80)$ **;**
mutation:best/1; **crossover:**bin; **restarts:**80i60v;

Experimental setup: DE+AE

$\#\text{FE}_{\text{max}} = 10^5 n$; $\varepsilon = \min(6n, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**50i30v;
$c_1 = 0.2$; $c_\mu = 0.2$; $c_\text{p} = default$;

Experimental setup: CMA-ES

$\#\text{FE}_{\text{max}} = 10^5 n$; $\varepsilon = \min(4 + \lfloor 3\ln(n) \rfloor, 80)$; **restarts:**80i60v;

On Fig. 4.15 and Fig. 4.16 is ECDF of ERT computed over all functions for dimensions 5 and 20. For both dimensions the DE+AE algorithm ($\circ$) reaches in average better fitness values than DE (+).

In more detail view, the results are depicted in Fig. 4.17 and Fig. 4.16 where ECDF of ERT is computed over subgroups of functions.

On separable function DE algorithm performs better than DE+AE. Although, in higher dimensions the difference is small. On separable function, adding AE rather confuses DE algorithm than contributes to its better performance, because

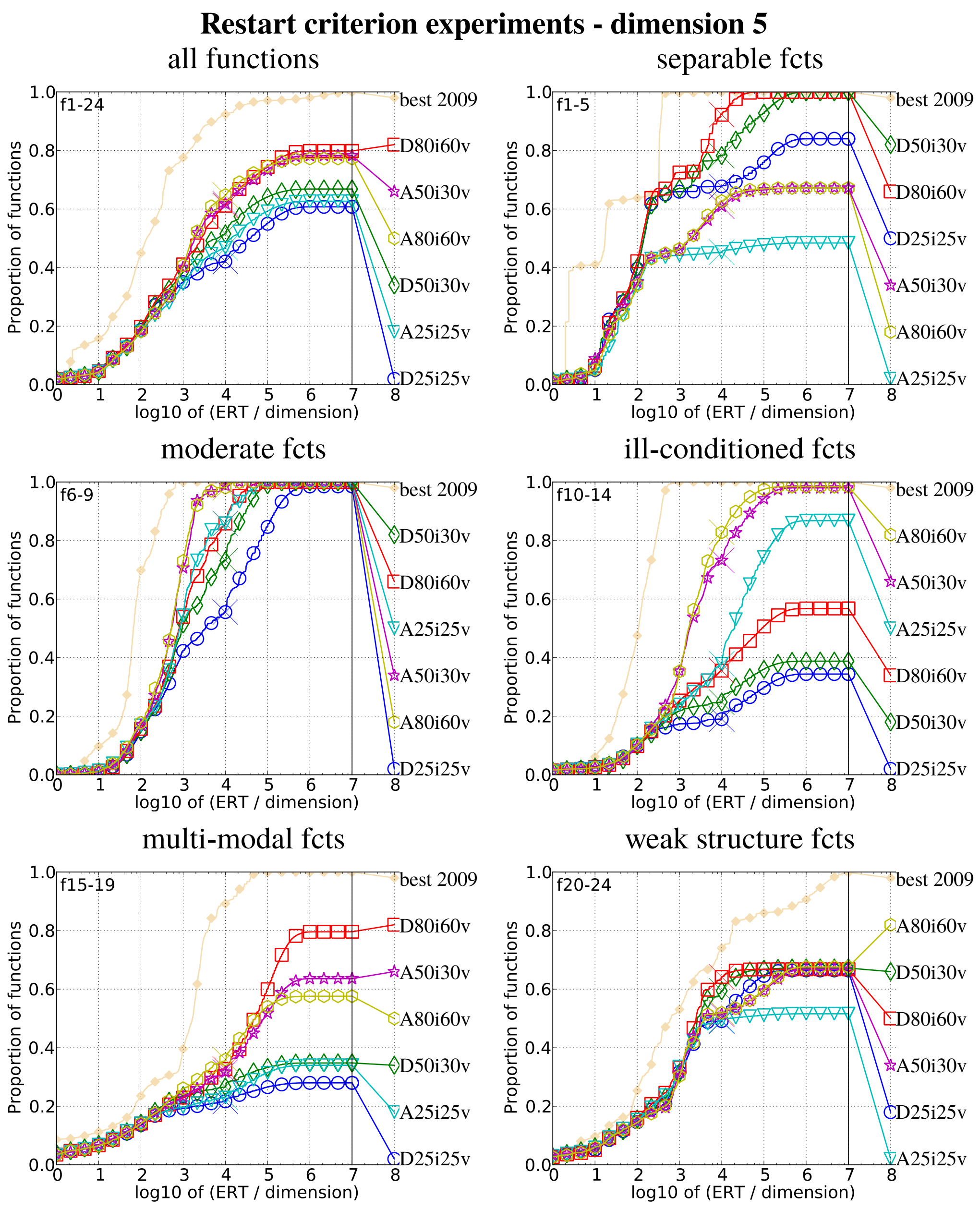

Figure 4.13.: ECDF of the bootstrapped ERT, see Sec. 4.1.3

Experimental setting:

#FE$_{\max} = 10^4 n$; $\varepsilon = \min\left(3n, 4 + \lfloor 3 \ln(n) \rfloor, 80\right)$;

mutation:best/1; **crossover:**bin; **restarts:**{25i25v, 50i30v, 80i60v};

$c_1 = default$; $c_\mu = default$; $c_\mathrm{p} = default$;

Abbreviations:

D: DE. A: AE.

XXiYYv: The notation convention is described at the end of Sec. 4.1.1.

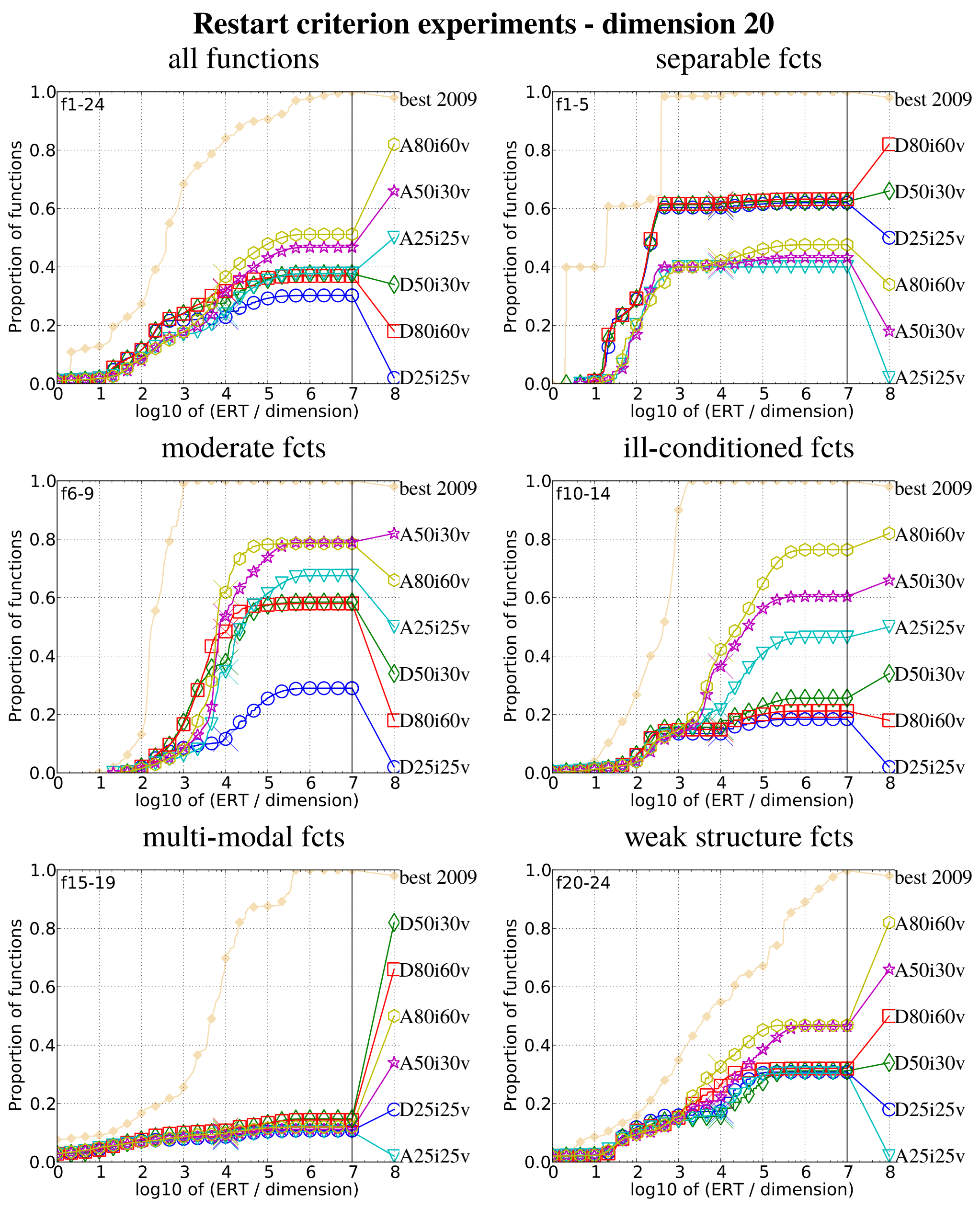

Figure 4.14.: ECDF of the bootstrapped ERT**,** see Sec. 4.1.3
Experimental setting:
$\#\mathrm{FE}_{\max} = 10^4 n$; $\varepsilon = \min(3n, 4 + \lfloor 3\ln(n) \rfloor, 80)$;
mutation:best/1; **crossover:**bin; **restarts:**{25i25v, 50i30v, 80i60v};
$c_1 = default$; $c_\mu = default$; $c_\mathrm{p} = default$;
Abbreviations:
D: DE. A: AE.
XXiYYv: The notation convention is described at the end of Sec. 4.1.1.

the fitness function is already rotated correctly and additional transformation is undesirable.

Situation changes for non-separable functions. On moderate, ill-conditioned and multi-modal function, DE+AE is superior to DE since adaptive encoding is helping with finding correct representation. In particular, DE+AE outperforms DE on ill-conditioned (i.e. "highly" non-separable) uni-modal functions.

On weakly structured function, DE reaches better results and proves to be more suitable for global optimization than DE+AE. However, the difference is negligible for higher dimension.

Fig. 4.19 and Fig. 4.20 provide comparison of DE, DE+AE and CMA-ES algorithms on ECDF of ERT using bootstrapping to estimate ERT for #EF up to 10^7.

In general view, CMA-ES performs the best on lower dimension problems while DE+AE slightly outperforms CMA-ES on problems with dimension 20. When considering type of fitness function, DE is the best option for separable functions, in particular is superior on lower dimensions. However, DE performs the worst on non-separable uni-modal functions. On those functions, DE+AE and CMA-ES perform almost the same on lower dimensions. For dimension 20, DE+AE reaches the best results. However, on moderate functions, CMA-ES is faster then DE+AE at the beginning of the search.

On multi-modal functions neither DE nor DE+AE is better than CMA-ES.

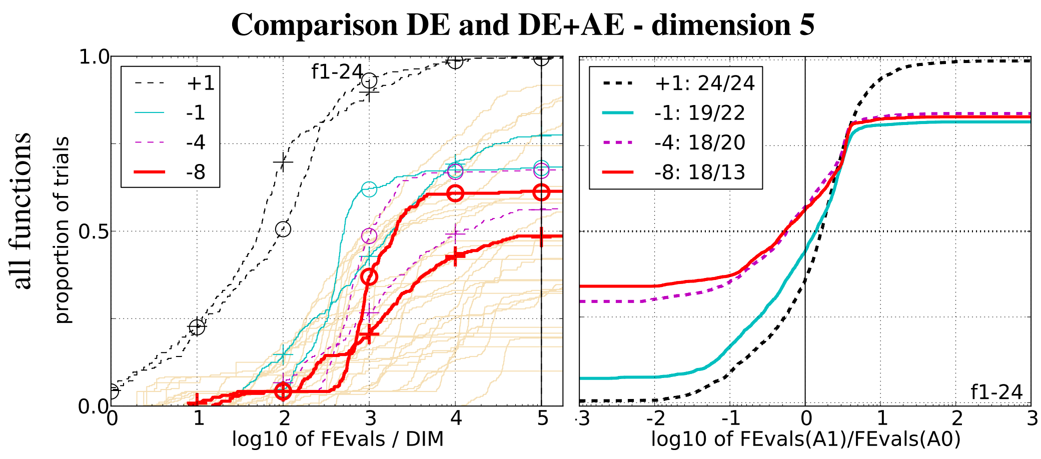

Figure 4.15.: ECDF of ERT, see Sec. 4.1.3 of DE (+) and DE+AE (○) Computed over all functions in noiseless testbed.

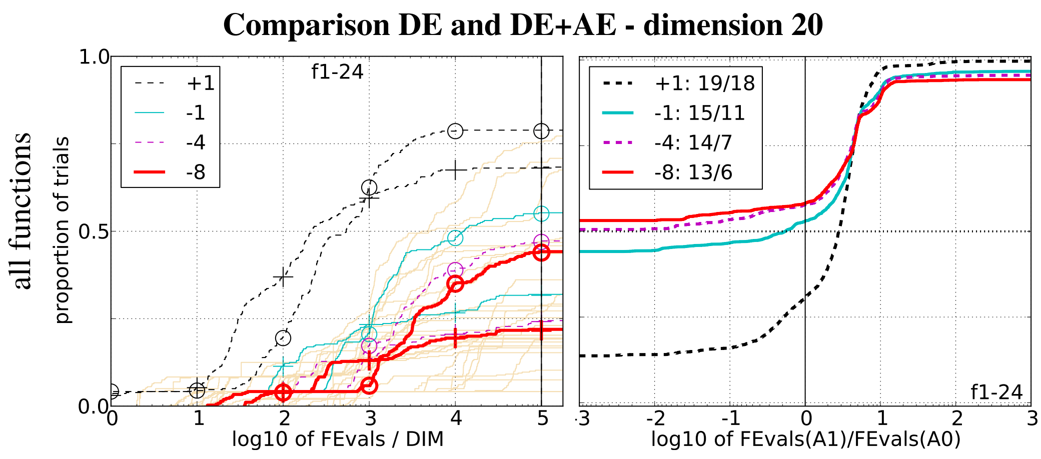

Figure 4.16.: ECDF of ERT, see Sec. 4.1.3 of DE (+) and DE+AE (○) Computed over all functions in noiseless testbed.

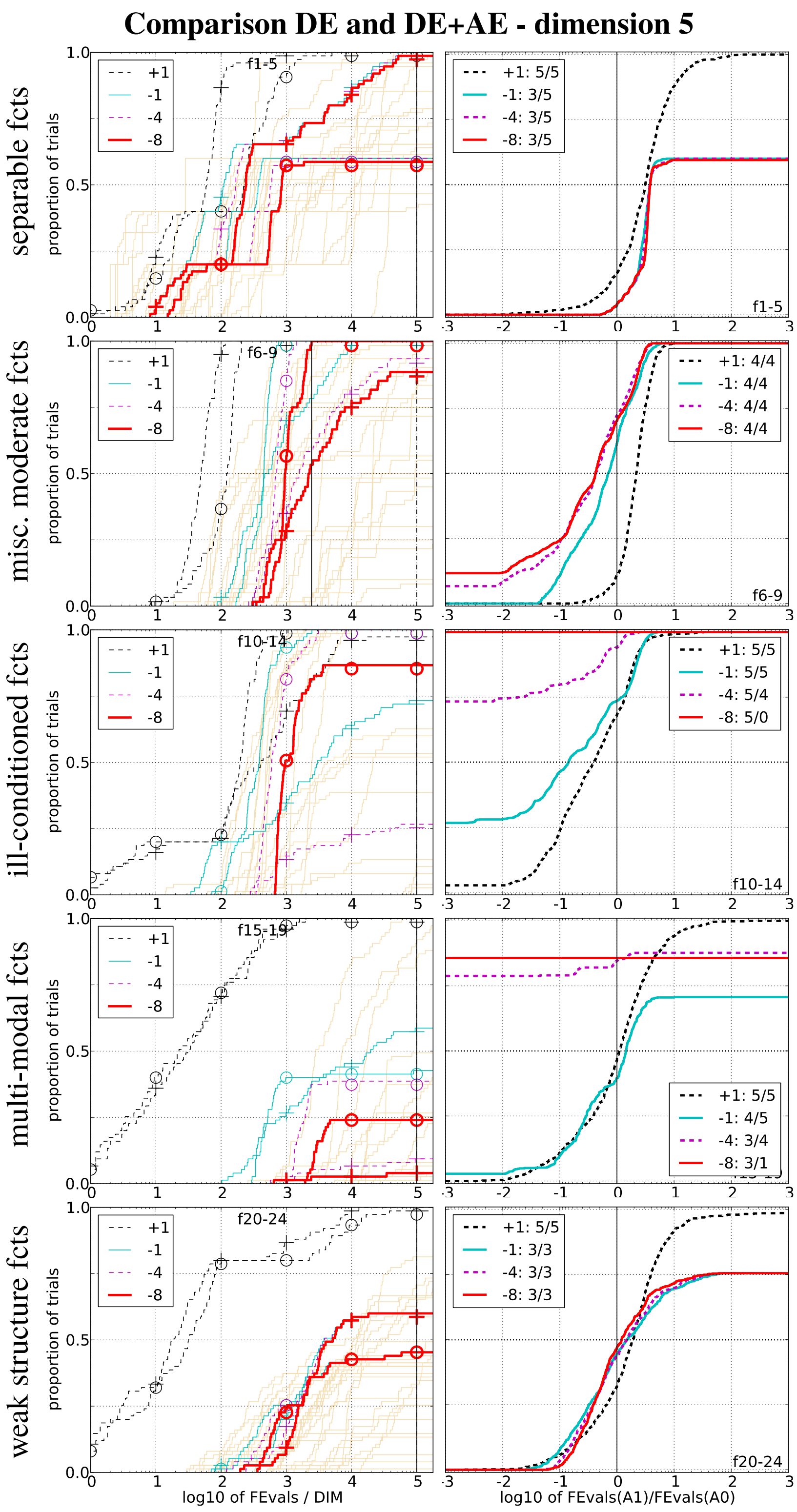

Figure 4.17.: ECDF of ERT, see Sec. 4.1.3 of DE (+) and DE+AE (○) Computed over subgroups of functions in noiseless testbed.

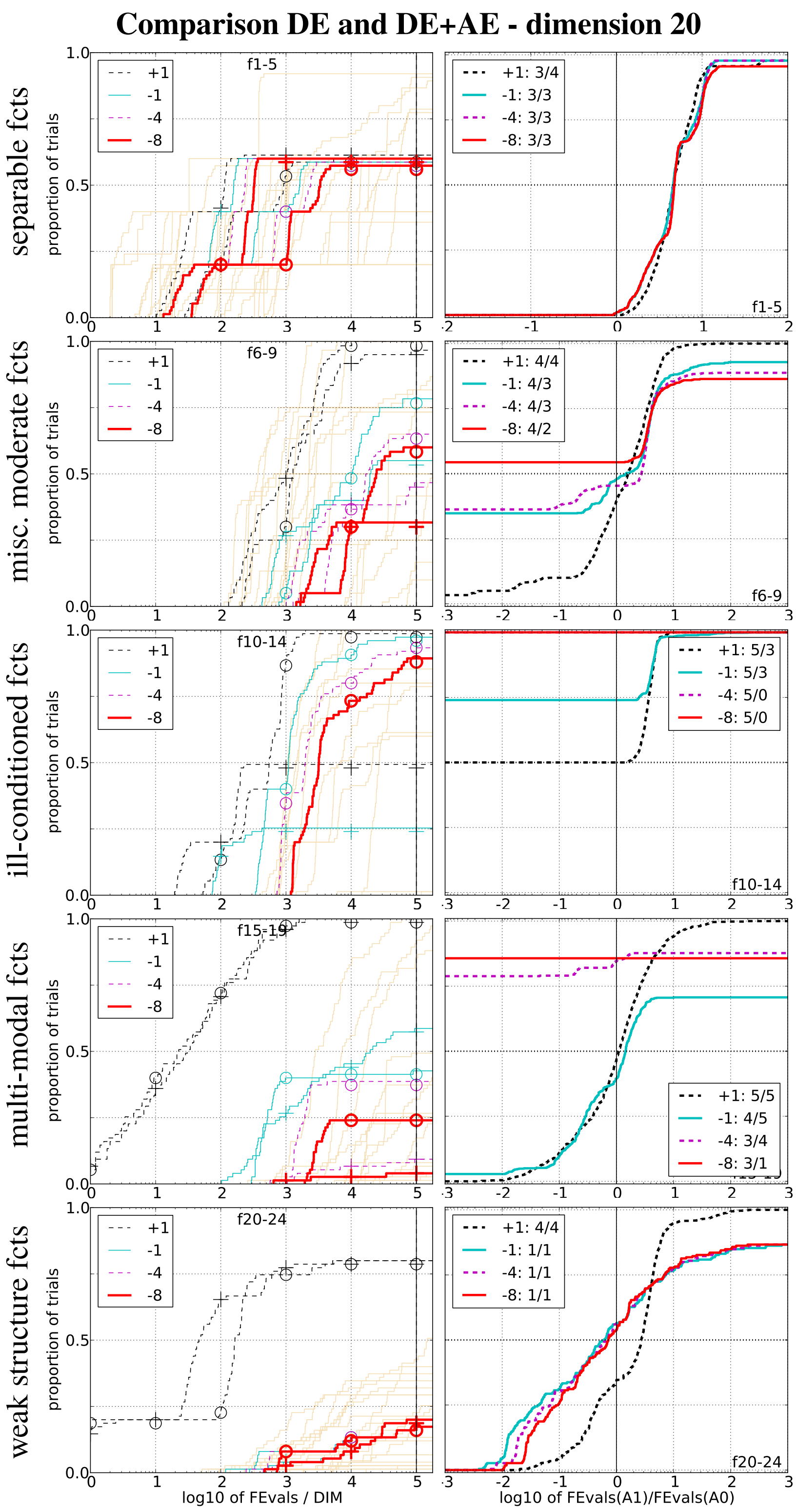

Figure 4.18.: ECDF of ERT, see Sec. 4.1.3 of DE (+) and DE+AE (○) Computed over subgroups of functions in noiseless testbed.

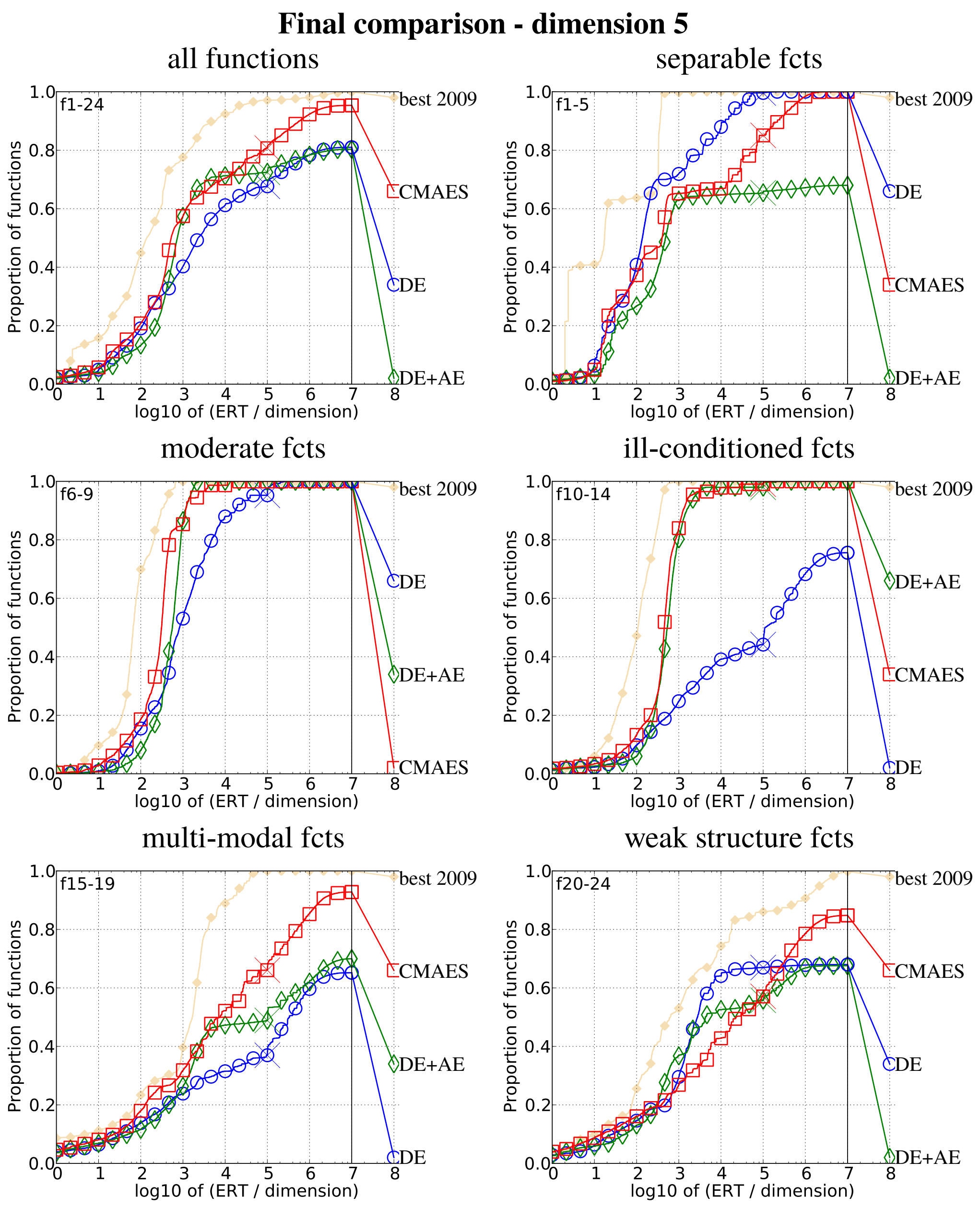

Figure 4.19.: ECDF of the bootstrapped ERT, see Sec. 4.1.3

Final comparison - dimension 20

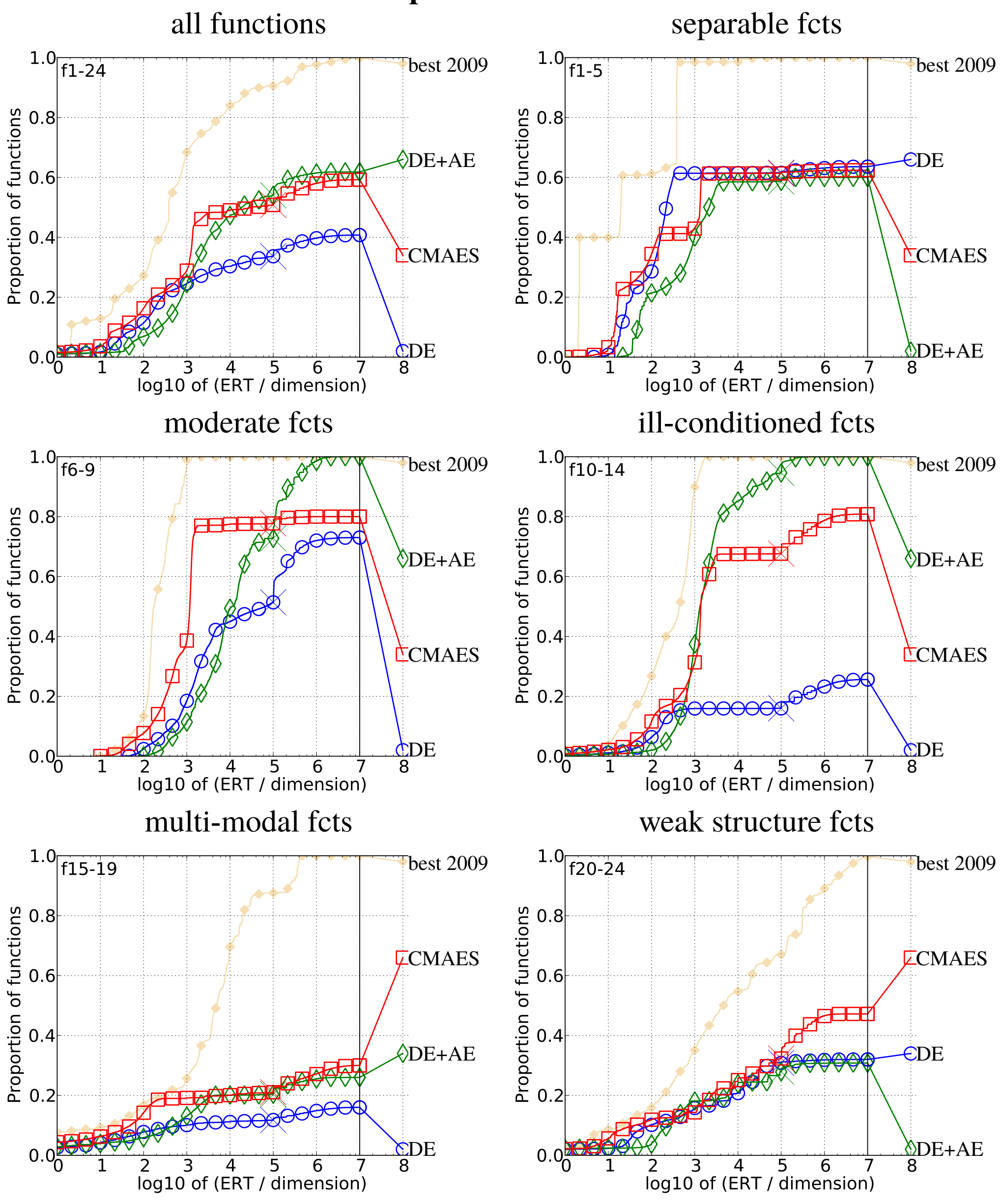

Figure 4.20.: ECDF of the bootstrapped ERT, see Sec. 4.1.3

CHAPTER

FIVE

CONCLUSIONS

Space representation is a key issue when designing well performing optimization algorithm. In this work, a system for adaptive encoding introduced by Hansen in [7] is applied to DE algorithm [14, 17, 19, 20]. The resulting novel algorithm DE+AE was implemented and tested on testbed of noiseless functions implemented in platform COCO [1, 6]. Performance of DE+AE was compared with original DE algorithm and CMA-ES.

Application of AE is beneficial primarily in case, that original optimization algorithm is not efficient on non-separable functions. Applying AE significantly improved performance of DE algorithm at ill-conditioned functions, where DE+AE evens up or outperforms the basic version of CMA-ES algorithms. However, when using AE, performance of DE on separable function is worst. Also, using AE does not bring significant improvement to DE on multi-modal functions.

The superiority of best/1 and avg/1 mutation operators suggests that AE will be more efficient when applied to optimizers performing search more locally.

Future work

In this work, the AE algorithm was applied to basic version of DE. However, there are variety of improvements increasing the performance of DE on non-separable functions (e.g. [19]) or equipping DE algorithm with automatic mechanisms for parameter adjustment [4, 21]. Applying AE to them could result in even better algorithm.

Also, further work should be focused on search of parameters for $\mathrm{AE}_{\mathrm{CMA}}$-Update. Depending on given optimization problem, experiments done within this thesis show that adjusting learning rate parameters c_1 and c_μ can significantly improve performance of DE+AE.

Regarding experiments done within this work, the tests could be executed on noisy functions as well.

BIBLIOGRAPHY

[1] *COmparing Continuous Optimisers: COCO* [online]. [cited 2. 5. 2011]. URL: http://coco.gforge.inria.fr/doku.php?id=start.

[2] *PlanetMath.org - Empirical distribution function* [online]. [cited 19. 7. 2011]. URL: http://planetmath.org/encyclopedia/EmpiricalDistributionFunction.html.

[3] BEYER, H.-G. – SCHWEFEL, H.-P. Evolution strategies: A comprehensive introduction. *Natural Computing: an international journal*. 2002, 1.

[4] BREST, J. et al. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on evolutionary computation*. December 2006, 10, 6, s. 646–657.

[5] FINCK, S. et al. *Real-Parameter Black-Box Optimization Benchmarking BBOB-2010: Experimental Setup*, 2009. URL: http://coco.lri.fr/BBOB-downloads/download10.2/bbobdocexperiment.pdf.

[6] FINCK, S. et al. *Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions*, 2009. URL: http://coco.lri.fr/downloads/download10.61/bbobdocfunctions.pdf.

[7] HANSEN, N. Adaptive Encoding for Optimization. Technical report, Centre de recherche INRIA Saclay – Île-de-France, 2008. URL: http://hal.inria.fr/docs/00/27/64/76/PDF/hansen-INRIA-RR-6518.pdf.

[8] HANSEN, N. *The CMA Evolution Strategy: A Comparing Review* [online]. 2006. [cited 13. 6. 2011]. URL: http://www.lri.fr/~hansen/hansenedacomparing.pdf.

[9] HANSEN, N. *The CMA Evolution Strategy: A Tutorial* [online]. 2008. [cited 20. 6. 2011]. URL: `http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf`.

[10] HANSEN, N. – KERN, S. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *Parallel Problem Solving from Nature*, 2004.

[11] HANSEN, N. – OSTERMEIER, A. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*. 2001, 9, s. 159–195. URL: `http://www.cs.colostate.edu/~whitley/CS640/cmaartic.pdf`.

[12] LARRAñAGA, P. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, A review on estimation of distribution algorithms, s. 80–90. Kluwer Academic Publishers, 2002.

[13] LOSHCHILOV, I. – SCHOENAUER, M. – SEBAG, M. Adaptive Coordinate Descent. *GECCO'11*. 2011, s. 885 – 892. URL: `www.lri.fr/~ilya/publications/GECCO2011_AdaptiveCoordinateDescent.pdf`.

[14] RONKKONEN, J. – KUKKONEN, S. – PRICE, K. Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, 1, s. 506 – 513, 2005.

[15] SPALL, J. C. *Introduction to Stochastic Search and Optimization*. Willey-Interscience, 2003.

[16] STORN, R. On the Usage of Differential Evolution for Function Optimization. In *Biennial Conference of the North American*, s. 519 – 523. Fuzzy Information Processing Society, 1996.

[17] STORN, R. *Differential Evolution (DE)* [online]. [cited 11. 5. 2011]. URL: `http://www.icsi.berkeley.edu/~storn/code.html`.

[18] STORN, R. – PRICE, K. Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, 1947 Center Street, Berkeley. URL: `http://www.icsi.berkeley.edu/~storn/TR-95-012.pdf`.

[19] SUTTON, A. M. – LUNACEK, M. – WHITLEY, L. D. Differential Evolution and Non-separability: Using selective pressure to focus search. GECCO '07, s. 1428–1435. ACM, 2007. URL: `http://doi.acm.org/10.1145/1276958.1277221`.

[20] SWAGATAM, D. – ABRAHAM, A. – KONAR, A. Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. *Studies in Computational Intelligence*. 2008, 116, s. 1–38.

[21] YAMAGUCHI, S. An Automatic Control Parameter Tuning Method for Differential Evolution. *Electrical Engineering in Japan*. November 2011, 174, 3, s. 1696–1703.

Appendix

APPENDIX

A

ABBREVIATIONS

AE Adaptive encoding

AE_{CMA}**-Update** Updating procedure based on CMA-ES algorithm.

CMA-ES Covariance Matrix Adaptation - Evolution Strategy

COCO COmparing Continuous Optimizers

DE Differential Evolution

DE+AE Differential Evolution equipped with mechanism for adaptive encoding

GAEF General Adaptive-Encoding Framework

APPENDIX
B

NOTATION

n	Dimension of optimization problem (scalar). In graphs and code snips also marked as DIM.
ε	Population size in DE (scalar)
λ	Number of samples generated every generation in CMA-ES (scalar)
μ	Number of best-ranked candidate solutions used for update (scalar)
$\mathbf{x}, \mathbf{m}, \mathbf{p}$	Vectors
$\mathbf{x}_i$	$i-$th individual of population (vector)
$x_{i,j}$	$j-$th element of $i-$th individual (scalar)
$\mathbf{B}$	Matrix
$\mathbf{B}^{\mathrm{T}}$	Transpose of a matrix
$\mathbf{I}^n$	Identity matrix $n \times n$
Θ	Vector space
$f()$	Objective function $f : \mathbb{R}^n \to \mathbb{R}$
f_{opt}	Minimum value of a fitness function
Δf	Desired precision to which an algorithm is optimizing an optimization problem
f_{target}	The target fitnessvalue to reach in optimization (before the algorithm is restarted) $f_{\text{target}} = f_{\text{opt}} + \Delta f$
f_{best}	Best (lowest) fitness value reached so far in a trial

#FE	Number of function evaluations
$\#\text{FEs}(f_{\text{best}} \geq f_{\text{target}})$	Number of function evaluations summed over all trials such that the best found solution up to now f_{best} has fitness higer than f_{target}
$\#\text{FE}_{\text{max}}$	Maximum number of fitness evaluation in one trial
$\text{OP}(\mathbf{x})$	Outer product $\text{OP}(\mathbf{x}) = \mathbf{x}\mathbf{x}^{\text{T}}$
$\mathscr{N}(\mathbf{m}, \mathbf{C})$	Multi-variate normal distribution with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. The matrix $\mathbf{C}$ is symmetric and positive definite; $\mathscr{N}(\mathbf{m}, \mathbf{C}) \sim \mathbf{m} + \mathscr{N}(0, \mathbf{C})$
$\mathscr{U}(a, b)$	Uniform distribution. Random numbers are drawn from range $[a, b]$

APPENDIX
C

CONTENTS OF SUPPLEMENT CD

Diploma_Thesis.pdf	This document in .pdf format
/Tests	Data containing results of performed tests
/Matlab	Matlab code of all programs used in this thesis