April 2014

Universiteit Leiden
Opleiding Informatica

Analyzing and Improving
Differential Evolution

Martin Wimmers

MASTER'S THESIS

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

Contents

Chapter 1

Introduction

Evolutionary algorithms are used in several fields to find optimal solutions for a given optimization problem. The power of these algorithms lies in the fact that no information on the underlying problem is required in order for them to work, and that the same algorithm can be used to optimize vastly differing functions [3]. The only requirement is that the problem has a continuous search space.[1] Whilst there are many types of evolutionary algorithms, most of them work on a similar principle; they usually mutate existing solutions in a specific way, and keep track of all the improvements they encounter. In the end, these algorithms provide the optimal solution, or a set of (Pareto-)optimal solutions that they encounter while running.
Several types of evolutionary algorithms exist. This paper will mainly focus on one type, namely Differential Evolution (DE) [16], and will briefly examine another one, called Evolutionary Strategy [4], against which DE is compared. However, there are many other types of evolutionary algorithms. Moreover, evolutionary algorithms have a multitude of variations each. Some variations implement slightly different parameter settings, or mutation rules. Other variations are hybridized algorithms, where two algorithms with different performances are combined to try and get the best of both worlds.

1.1 Applications

Evolutionary Algorithms are often used in practice to look for improvements or optimizations in designs [6]. Applications include, but are not limited to: optimizing airfoils, aiding in nuclear reactor design, and car design. These algorithms have been proven useful in seeking improvements in complicated designs. In order to predict whether an Evolutionary Algorithm will perform well in a real-life application, a method needs to be used to predict how well it will perform, and how fast it will converge. If an algorithm performs well, it can avoid local optima, and find the global optimum of a given function, and fast convergence is desired whenever the maximum amount of solution evaluations is limited. Moreover, different algorithms need to be compared, to analyze them for their strengths and weaknesses. For this, standardized benchmarks are needed.

1.2 Benchmarks

In order to compare different competing algorithms, at least one benchmark of functions is needed. It needs to have a variety of different functions that represent various classes of functions (for instance: separable and non-separable), and it should represent possible real life applications. In this paper, two benchmarks are used. One is called Black Box Optimization Benchmark [8] (or BBOB), and the other is called Large Scale Global Optimization Benchmark, or LSGO [11].
BBOB is used at the Genetic and evolutionary computation conference (GECCO) [15], and is used to compare algorithm performance. The strength of BBOB is that it provides a multitude of functions (see figure 1), some separable and some non-separable, and these functions can be scaled to any desired higher dimension. Since the functions of BBOB are all very different, an algorithm that performs well on all of the functions can be expected to perform well in real life applications.

[1]There are workarounds for problems that do not have a continuous search space; e.g. penalize all solutions that violate a constraint, or are not part of the search space.

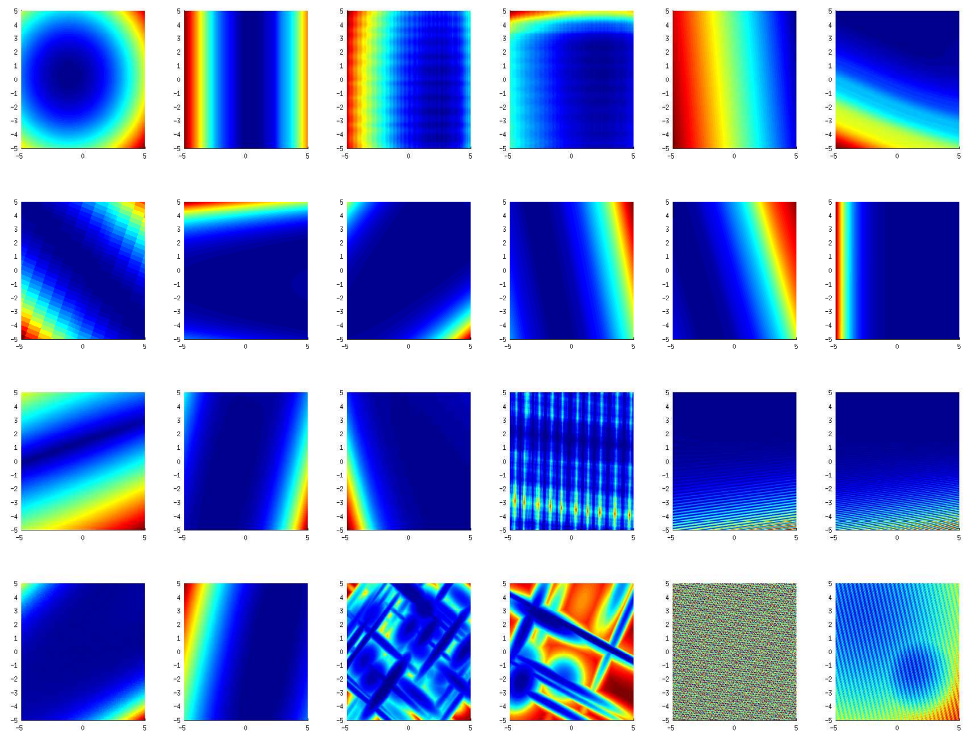

Figure 1.1: Surface plots of the 24 functions used in BBOB, in 2D.

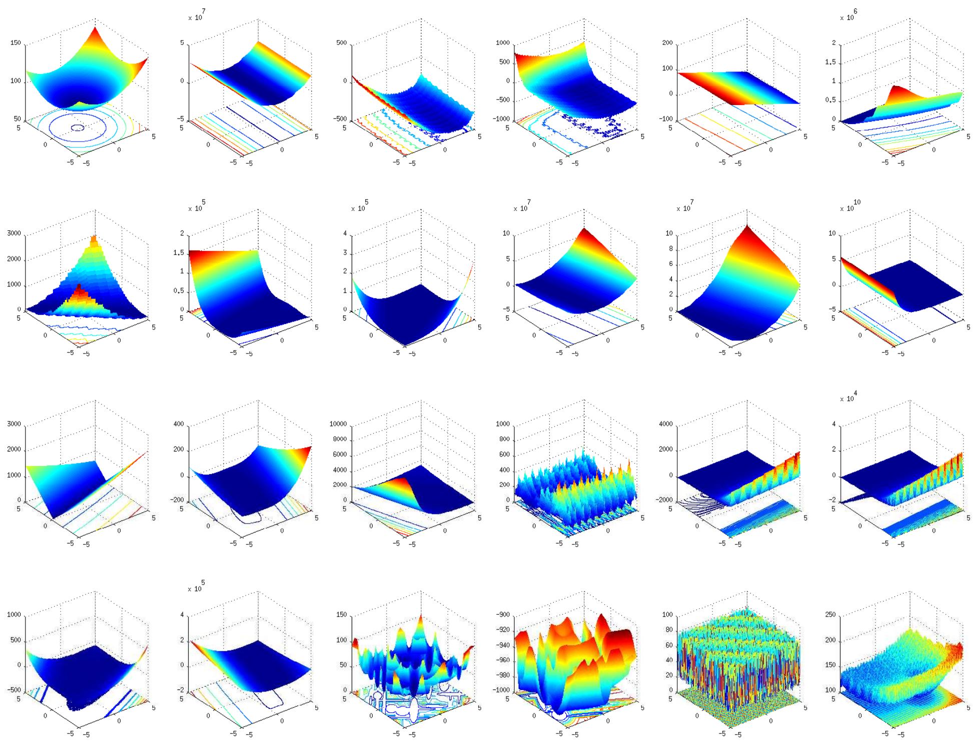

Figure 1.2: Surface plots from the side of the 24 functions used in BBOB, in 2D.

LSGO tests the performance of an optimization algorithm for problems of about 1000 variables. This benchmark cannot be used to see how an algorithm fares with respect to an increase in dimensions. Rather, it is designed to compare optimization algorithms in high dimensions. Since high-dimensioned problems are difficult to solve, this is an interesting area for comparing and developing algorithms. In the experiments described in this thesis, each function of the LSGO is run ten times, in order to compare relative performance.

1.3 Algorithms

Whilst there are several types of evolutionary algorithms, this paper will focus primarily on one variation, called Differential Evolution (DE). This algorithm has very simple mutation rules, yet it performs quite well. The performance of DE, as well as variations thereof, will be analyzed. The performance will also be compared with that of other state of the art evolutionary algorithms. A special look is taken at jaDE[13], a DE variant that has performed very well, and seems to be the current state of the art DE variant. Finally, a new DE based algorithm will be introduced, with several variations, including one based on DE and jaDE. This new algorithm attempts to overcome some of the drawbacks of currently existing DE variants.

1.4 Thesis Goals

An evolutionary algorithm that performs better than other evolutionary algorithms is good to have, but it is also important to understand why an algorithm performs better in some conditions, and worse in others. This thesis attempts to analyze performance of the state of the art algorithms on several functions that represent various difficulties, and attempts to understand why some optimization algorithms perform better than others, and under which conditions. This thesis will try to answer the following questions: *what are the strengths and weaknesses of Differential Evolution, how can its drawbacks be overcome, and why does it have its strengths and weaknesses?*

After understanding the basic problems of the DE based algorithms, some improvements to DE/jaDE are suggested, and this algorithm will also be compared to the same state of the art algorithms. Finally, the thesis will summarize the findings, and propose further research questions.

This thesis is structured as follows: In Chapter 2, Differential Evolution is discussed in detail. Also, some variants of Differential Evolution are shown. In Chapter 3, the performance of DE is compared to some state of the art Evolution Strategies (a different class of optimizing algorithms). In Chapter 4, a new variant of DE is proposed, explained and analyzed. Finally, this thesis' conclusions are presented in Chapter 5, and the appendices show some analysis of the algorithms' performance.

Chapter 2

Differential Evolution

2.1 Description of Differential Evolution

Differential Evolution [16] is an algorithm, proposed in 1997, for finding optima of functions in a continuous search space. It finds new solutions by selecting some already existing solutions, taking the difference between them, and adding this difference to another solution, in order to find an improvement. The algorithm first starts with a set of NP candidate solutions, also known as the population. Each candidate solution is a vector $\vec{v_i} \in \mathbb{R}^n$, where $0 < i \leq NP$, and n is the dimensionality of the vector, or the dimension of the function. The name Differential Evolution stems from the exploration method in the search space: when trying to find an improvement for the vector $\vec{v_i}$, the *difference* between two distinct vectors is scaled by a factor F, and added to $\vec{v_i}$ (the vectors used to create the difference vector are distinct, and none of them may be $\vec{v_i}$), to create a solution vector $\vec{x_i}$. Then, if $f(\vec{x_i}) < f(\vec{v_i})$, $\vec{v_i}$ replaces $\vec{x_i}$[1]. DE requires only a few parameters to be set beforehand, is simple to use, yet robust, and can easily be used in parallel computing [16]. The algorithm for Differential Evolution is shown in algorithm 1. Since this is the most basic and the first proposed DE algorithm, this algorithm will be called canonical DE when compared to its variants.

[1]This assumes a minimization problem. If the goal is to maximize a function value, $\vec{v_i}$ will be replaced if $f(\vec{x_i}) > f(\vec{v_i})$

Algorithm 1 The canonical Differential Evolution algorithm, in an instance where a minimum value of the evaluation function is sought.

```
for $i = 1 \to NP$ do
    for $j = 1 \to D$ do
        $v_{i,j} \leftarrow rand_{number} \in$ Search Space ▷ Set the initial population, giving each vector values within the desired search space of the problem
    end for
end for
while not terminate do
    for $i = 1 \to NP$ do
        $j = rand_{integer} \in [1, D] \setminus \{i\}$
        $k = rand_{integer} \in [1, D] \setminus \{i, j\}$
        $x_i \leftarrow v_i + F(v_j - v_k)$
        $d = rand_{integer} \in [1, D]$
        for $l = 1 \to D$ do
            if $rand \in [0, 1] < CR$ or $l == d$ then
                $x_{i,l} \leftarrow v_{i,l}$                                ▷ Applying crossover
            end if
        end for
    end for
    for $i = 1 \to NP$ do
        if $f(x_i) < f(v_i)$ then                          ▷ Assuming a minimization problem
            $v_i \leftarrow x_i$
        end if
    end for
end while
```

Canonical DE works by first generating NP different initial solutions. Then, for a number of times, this amount being specified by the user, the algorithm creates NP new individuals, using the formula $x_i \leftarrow v_i + F(v_j - v_k)$ for $0 < i, j, k \leq NP, i \neq j, i \neq k, j \neq k$. So, for each existing individual x_i, the difference between 2 other distinct existing solutions, multiplied by the difference factor F, is added. Afterwards, for each dimension $d \in D$, the probability of inheriting the value of the previous is CR. Moreover, one dimension is randomly chosen, and that dimension will also be inherited. Finally, after generating all new individuals, the individuals that produced better vectors are replaced by the vectors that they generated.
A few parameters must be set by the user: F is the difference factor; it scales the difference vector that is added to the target vector, as used in the mutation step: $x_i \leftarrow v_i + F(v_j - v_k)$. CR is the crossover rate, which determines how much new information the target vector will receive; as the value of CR gets close to 1, $\vec{x_i}$ is more likely to resemble $\vec{v_i}$. Finally, there is NP, which determines how many vectors there are in the population. It must be noted that the population size should be at least 4, because the target vector, and the two difference vectors must all be distinct.

2.1.1 Setting the parameters

As noted, Canonical Differential Evolution requires three parameters to be set. What the optimal settings are depends on the characteristics of the underlying function that is to be optimized. However, the only part of the algorithm that depends on the underlying function is the selection process that determines which individuals go to the next generation. The process of creating new vectors, and crossover is independent of the function, and only depends on the parameters given. Hence, by analyzing the effect of setting the parameters, the user can make a wise decision, based on knowledge of the function that is to be optimized.

The new solutions that can be generated by Differential Evolution in two dimensions are illustrated in fig. 2.1 . Both figures have set $F = 0.7$, and they show all possible outcomes in one generation, given a specific population. It becomes evident that a higher crossover rate works better for separable functions, whereas a low crossover rate is better on ill-conditioned functions. This seems to stem from the fact that for a separable function, one can optimize each dimension independently of the others. Since there is no relationship between the dimensions with respect to the optimum, by having a high crossover rate, dimensions can be explored more quickly. However, for non-separable functions, it's better to explore in more dimensions, as dependencies can exist between dimensions. Also, a bigger value of F increases the distance that can be explored from each already existing solution. However, for relatively low population sizes, a large F means that large areas are left unexplored, since the gaps between the parents and potential offspring increase. In this plot, $F = 0.7$ was chosen, because according to [16], values between 0.4 and 1.0 for F work best. As a compromise, the mean value of 0.7 was chosen.

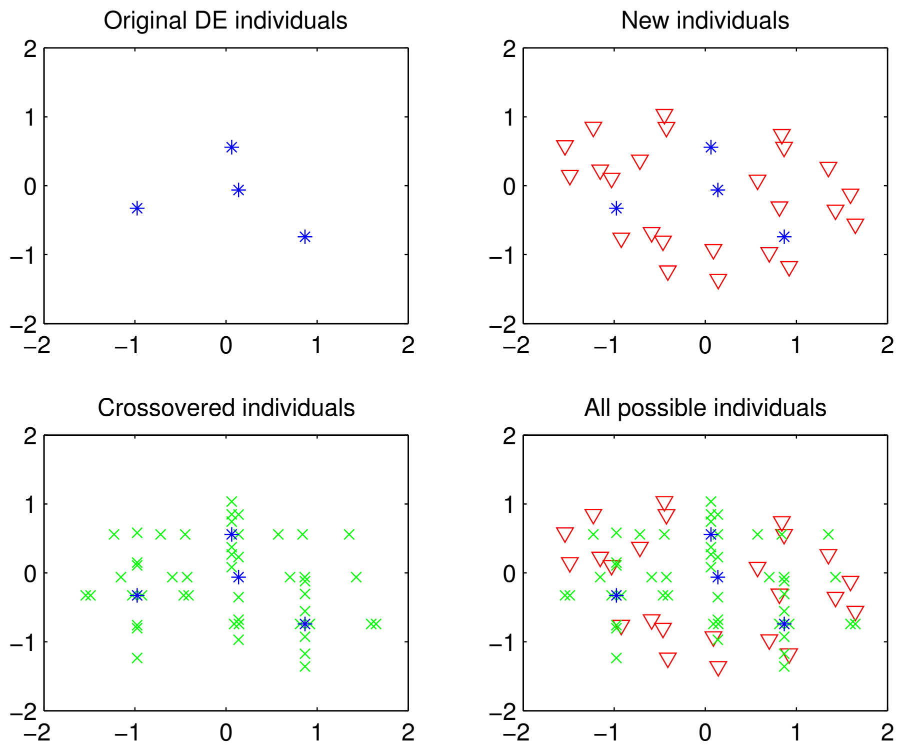

Figure 2.1: This figure shows all possible offspring from a given DE configuration with a population size of 4 and F = 0.7, in 2D.

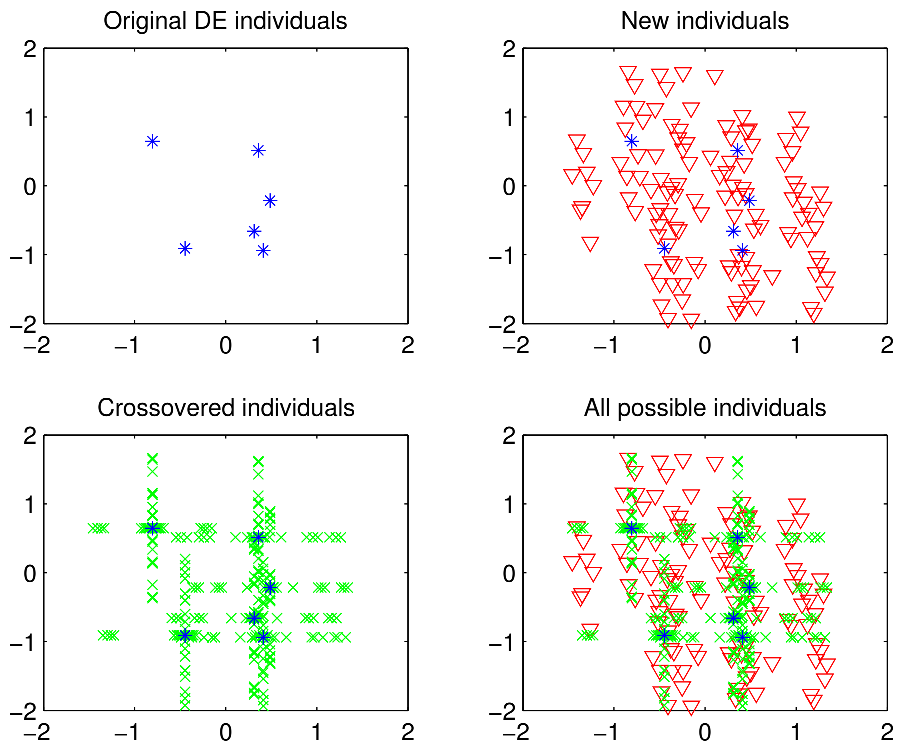

Figure 2.2: This figure shows all possible offspring from a given DE configuration with a population size of 6 and F = 0.7, in 2D.

2.1.2 Differential Evolution Variations

There are several strategies possible for mutating vectors in differential evolution [16]. Algorithm 1 shows the basic, or canonical differential evolution strategy. This strategy works by taking a random vector and adding to it a random difference vector, crossing it over with a parent, and comparing both. However, a parent can be mutated using similar, but other rules. For instance, instead of adding a random vector to a random difference vector, the random difference vector can be added to the vector with the best fitness value. This idea is expressed in equation 2.1.

$$\vec{x}_i \leftarrow \vec{x}_{best} + F(\vec{x}_j - \vec{x}_k) \text{ for } 0 < j,k \leq m,\ best \neq j, best \neq k, j \neq k \tag{2.1}$$

The update rule for DE/best/1

Instead of expressing mutation strategies with words, a standardized notation, proposed in [16], is introduced. It takes the form of $DE/x/y$, with x being the strategy, and y being the number of difference vectors. Using this form, canonical DE is $DE/rand/1$ and the newly proposed strategy is $DE/best/1$. Other strategies, proposed in [16, 7], are shown in eqs. (2.2) to (2.4). $DE/rand/2$ creates two difference vectors, rather than one, and adds them to another existing individual, $DE/best/2$ adds two difference vectors to the best individual and $DE/target-to-best/1$ adds the difference vector of the best vector to an existing individual, as well as adding another random vector to the result.

$$\begin{gathered}\vec{x_i} \leftarrow \vec{x_i} + F(\vec{x_h} - \vec{x_j}) + F(\vec{x_k} - \vec{x_l}) \\ \text{for } h,j,k,l\ rand_{integer} \in [1,D],\ h,i,j,k,l \text{ pairwise distinct}\end{gathered} \tag{2.2}$$

The update rule for DE/rand/2; two random vectors are added to an individual

$$\begin{gathered}\vec{x_i} \leftarrow \vec{x}_{best} + F(\vec{x_h} - \vec{x_j}) + F(\vec{x_k} - \vec{x_l}) \\ \text{for } h,j,k,l\ rand_{integer} \in [1,D],\ h,i,j,k,l \text{ pairwise distinct}\end{gathered} \tag{2.3}$$

The update rule for DE/best/2, two random vectors are added to the best individual

$$\begin{gathered}\vec{x_i} \leftarrow \vec{x_i} + F(\vec{x}_{best} - \vec{x_i}) + F(\vec{x_h} - \vec{x_j}) \\ \text{for } h,j\ rand_{integer} \in [1,D],\ h,i,j \text{ pairwise distinct}\end{gathered} \tag{2.4}$$

The update rule for DE/target-to-best/1, a random difference vector is added to the difference vector between the best and the mutated individual

2.1.3 Differential Evolution Performance

To compare the performance of various Differential Evolution Strategies, the Black Box Optimization Benchmark [1] is used, with problems from dimensionality $D = 2$ to $D = 40$. To evaluate the effects of parameters, and versions, the following strategies, suggested in [7], are tested:

- DE/rand/1 (Canonical DE)
- DE/rand/2
- DE/best/1
- DE/best/2
- DE/target-to-best/1

Since the versions with two vectors require at least six elements in the population (target and 4 other vectors), The following populations sizes are tried: $3 \cdot D$(the absolute minimum, because for DE/rand/2, 5 distinct vectors are needed), $6 \cdot D$ and $9 \cdot D$, to see whether more diversity or more iterations improve overall performance. The maximum number of function evaluations is kept at $D \cdot 10^4$, D being the number of variables or dimensions.
Since the crossover rate can also affect the performance, three settings are used: $CR = 0.2$, 0.5 and 0.8. 0.5 is the most uniform crossover, 0.2 lets the algorithm search closer to the target vector, and 0.8 focuses the search closer to the initially generated new vector. Finally, as F is recommended to be in the range $[0.4, 1]$, the values of $F = 0.4$, 0.7 and 1 are chosen.
To compare performance across the DE variations, all variable settings are tested on DE/rand/1. The variable settings that perform best are then set and used to test the other versions.

Canonical Differential Evolution Performance

To determine the parameters that can be used to compare Canonical DE
($DE/rand/1$) to other DE strategies, several experiments are run. The first series of experiments run Canonical DE, using all combinations of the previously mentioned values for F, CR, and NP. Each of these experiments is run on the standard BBOB test suite, for $D \in 2, 3, 5, 10, 20, 40$. Since these experiments yield a large amount of data, the parameters F and CR are compared with the same population sizes. The graphs that show the performance of Canonical DE with various parameter settings are provided in Appendix A. The following analysis is based on the performance in $D = 5$ and $D = 20$, to simplify the analysis of trying to find a good parameter setting.

For $P = 3 \cdot D$, it seems that the best choice is to have $F = 0.4$. In the overall best scoring section, $F = 0.4$ yields the best performance. The higher the crossover rate for $F = 0.4$, the better it performs. So it seems that exploration in the nearby area works best for a small population size. It is also noticeable that for 20 dimensions, regardless of the population size, setting $F = 0.4$ and $CR = 0.8$ (80% of the new vector comes from the difference vector, rather than the parent) works best overall.

In 5 dimensions, for $P = 6 \cdot D$ and $P = 9 \cdot D$, the settings $CR = 0.8$ and $F = 0.7$ worked best. In $P = 3 \cdot D$, this setting was second best, only slightly behind $CR = 0.5$ and $F = 0.7$. The difference in average performance between the population sizes is very small. Moreover, it must be remarked that *the best performing settings for 20D, yield the worst performance in 5D, for $P = 3 * D$, $C = 0.5$ and $F = 0.4$.*

Seeing that the best parameters depend on the dimensionality, and that a parameter setting can vary greatly in performance, a stable parameter setting is needed. Fortunately, one combination of parameter settings showed consistently good behavior within its population size. These settings are $CR = 0.5$ and $F = 0.4$. For $P = 3 * D$ and $P = 6 * D$, this combination performed second best in 5D and third best in 20D, which showed the most consistent performance. Moreover, for $P = 9 * D$, these settings for F and CR performed second best within their population sizes, much more consistent than any of the other settings.

Since the settings for $F = 0.4$ and $CR = 0.5$ perform consistently well within their population sizes, the figures of Appendix A illustrate the performance of these settings for $P \in 3 \cdot D, 6 \cdot D, 9 \cdot D$. Also included is the setting $CR = 0.8$, $F = 0.4$ and $P = 3 * D$, because in 5D problems it performed best within its population size, and in 20D it performed worst within its population size.

The results suggest that smaller populations perform better than large populations in large dimensions, and the opposite is true for small dimensions. A large population corresponds with a large diversity, but the number of generations is limited. A small population can explore more generations. Moreover, in high-dimensional problems, setting CR to lower values increases the exploration rate, as opposed to the exploitation rate, as new solutions would tend to differ more, and hence explore in more dimensions simultaneously. This is especially useful for non-separable functions.

When looking at the results, different problem types have different optimal parameters. For moderate facts, DE performs better in lower dimensions with a high population size. A relatively low population size (which allows for more generations of solutions) performs better in higher dimensions. A similar

observation can be made for ill-conditioned and multi-modal problems. Moreover, each experiment uses a population size, as well as an evaluation budget that grows linearly with respect to the problem dimensionality. This suggests that the optimal population size is either a fixed, or increases less than linearly with respect to the number of dimensions. This may be the case because higher dimensions require much more exploration compared to lower dimensions, and a smaller population size automatically means more generations, hence increasing the potentially explored distance.
To compare performance across the multiple DE strategies, the settings are set to: $P = 6 \cdot D$, $F = 0.4$ and $CR = 0.5$. The population size $P = 6 \cdot creD$ is chosen, because its performance was either average, or very similar to $P = 9 \cdot D$ or $P = 3 \cdot D$, so it seems to be a good representation of the Canonical DE algorithm across multiple dimensions.

Performance of Different Differential Evolution Strategies

The performance graphs of the various DE variants are shown in Appendix B. In all cases, the values for all parameters are kept identical, as outlined above. It should be noted that the $DE/rand/1$ strategy is the Canonical DE strategy as described before.

When looking at overall performance, the DE/best/2 strategy works best, except for $D = 2$, where it performs worst. However, since the performance difference for $D = 2$ between the strategies is minimal, it can be said that DE/best/2 is the best strategy overall.

Canonical DE ($DE/rand/1$) also performs relatively well across the benchmark, except for $D = 5$. When scaling the problems to higher dimensions, it is clear that both the $DE/best/2$ and $DE/rand/1$ strategies outperform all the other strategies.

In higher dimensions, the problem of all DE strategies becomes apparent: the strategies only scale up well for separable problems.

2.2 jaDE

The results of the experiments show that using a canonical DE, or variations thereof that employ different vector mutation strategies, different settings for P, F and CR perform better in different types of functions. The big drawback of standard DE is that these parameters are fixed, rather than learned. To overcome this problem, a variant, called jaDE [17], was made. jaDE differs from standard DE in several ways. It employs a *current-to-pbest strategy.* This strategy adds a difference vector containing a random vector from the $100 \cdot p\%$ best vectors, where the user has to set p, and a completely random vector to an existing individual. This strategy is less greedy than the current-to-best strategy, and this reduces the likelihood of jaDE to get stuck in a local optimum.

Algorithm 2 The jaDE algorithm, in an instance where a minimum value of the evaluation function is sought.

```
$\mu_{CR} = 0.5$
$\mu_F = 0.5$
$A = \emptyset$
for $i = 1 \rightarrow NP$ do
    for $j = 1 \rightarrow D$ do
        $v_{i,j} \leftarrow rand$
    end for
end for
while $not\ terminate$ do
    $S_F = \emptyset$
    $S_{CR} = \emptyset$
    for $i = 1 \rightarrow NP$ do
        $CR_i = randn_i(\mu_{CR}, 0.1)$
        $F_i = randc_i(\mu_F, 0.1)$
        $x^p_{best} = rand \in 100 \cdot p$ best vectors
        $x_1 = rand \in X \backslash x_i$
        $x_2 = rand \in X \cup A \backslash \{x_i \cup x_1\}$
        $x_i \leftarrow v_i + F_i(x^p_{best} - v_i) + F_i(x_1 - x2)$
        $d = rand_{integer} \in [1, D]$
        for $j = 1 \rightarrow D$ do
            if $rand \in [0,1] < CR$ or $j == d$ then
                $x_{i,j} \leftarrow v_{i,j}$
            end if
        end for
        if $F(x_i) < F(v_i)$ then
            $v_i \rightarrow A$
            $CR_i \rightarrow S_{CR}$
            $F_i \rightarrow S_F$
            $v_i \leftarrow x_i$
        end if
    end for
    while $|A| > NP$ do
        Remove random element $\in A$
    end while
    $\mu_{CR} = (1 - c) \cdot \mu_{CR} + c \cdot meanA(S_{CR})$
    $\mu F = (1 - c) \cdot \mu_F + c \cdot meanL(S_F)$
end while
```

Moreover, jaDE does not use fixed values for F and CR. Instead, F and CR are now expected values, and each individual vector mutates these values, using a normal distribution for the crossover rate, and a Cauchy distribution for the difference factor. The crossover rate is generated with $\mu = S_{CR}$ and $\sigma = 0.1$, to keep mutation small. For the difference factor, it uses a Cauchy distribution with $\mu = S_F$ and $scale = 0.1$. Then, all successful values for F and CR are remembered, and for the next generation, new values for F

and CR are computed, using the mean value for CR and the Lehmer mean[2] for F. By using a Lehmer mean, rather than an arithmetic mean, the adaptation of μ_F places more weight on larger successful mutation factors.
Optionally, an archive can be used, to keep track of successful parents and values from previous generations. Since the update rule requires both two random vectors, and a vector containing a $p$best individual, some individuals from the archive may be used in choosing the $p$best (from both current population and archive) vector, to generate one of the random vectors. The archive was used for all experiments with jaDE that are described in this thesis.

2.2.1 jaDE Performance

In Appendix C, jaDE performance is compared to the performance of standard DE algorithms, namely the best performing Canonical Differential Evolution algorithm and the DE/best/2 strategy. As can be seen, jaDE performs poorly in lower dimensions, but in higher dimensions significantly outperforms the other algorithms. This shows that jaDE scales up better than standard DE algorithms.

[2]The Lehmer mean of a set X is defined as $mean_L(X) = \frac{\sum_{x\in X} x^2}{\sum_{x\in X} x}$

Chapter 3

Differential Evolution and Evolution Strategies

It is important to compare the performance of Differential Evolution with the performance of other optimization algorithms, in order to uncover its merits and weaknesses as an optimization algorithm. Besides Differential Evolution, there are other algorithms designed to find optima of continuous functions. Differential Evolution and its variation are a subset of Evolutionary Algorithms, a whole class of algorithms designed for optimization [3, 4].
There are several types of Evolutionary Algorithms. each with their own merits and weaknesses. In this thesis, Differential Evolution will be compared to Evolution Strategies. These are also a type of Evolutionary Algorithms, and they too performed well on BBOB. Hence, a comparison between Differential Evolution and Evolution Strategies can yield an interesting performance analysis.

3.1 Evolution Strategies

An Evolution Strategy works on the principle of mutating an existing solution. Unlike Differential Evolution, the mutation does not use the values of other solutions. Rather, it uses a normal distribution to generate a vector, with each element being generated using the same mean and variance values. This strategy allows for a population size as low as 1.
Just like with Differential Evolution, there are several types of Evolution Strategies. The different types differ in their method of mutation. Some basic strategies only mutate the variance of the mean distribution used for mutation (e.g. 1+1 ES [4]), whereas other methods use more sophisticated methods to try and improve their learning and convergence rate.

3.2 Basic Evolution Strategies

A basic Evolution Strategy is controlled by three parameters: μ, λ, σ. Moreover, there are two types of strategies: plus (+) and comma(,). μ is the number of solutions that are evaluated at the beginning of each iteration, called parents, λ is the number of new solutions generated from the μ already existing individuals, called offspring. σ is the step size, which determines how much each offspring differs from its original parent. σ should be set, and adapted, in such a way to maximize convergence rate. For a 1+1 ES, the optimal rate is achieved if approximately 1 in every 5 offspring generated is an improvement, as this theoretically maximizes convergence[14].
Finally, when the next set of individuals must be chosen, the plus (+) strategy selects the μ best new individuals from both the parents and offspring, whereas the comma (,) strategy only selects the μ best offspring generated in the last generation. This means that a comma strategy can deteriorate in a generation. However, a temporary deterioration can be useful to overcome local optima.

Algorithm 3 The basic Evolution Strategy algorithm

```
initialize $x, \sigma, \mu, \lambda$
while Not terminate do
    $\sigma' \leftarrow \sigma \cdot exp(\tau_0 \cdot \mathcal{N}(0,1))$
    for $i = 1 \rightarrow \mu$ do
        for $j = 1 \rightarrow \lambda$ do
            $x'_{i,j} \leftarrow x_i + \sigma' \cdot \mathcal{N}(0,1)$
        end for
    end for
    for $i = 1 \rightarrow \mu$ do
        for $j = 1 \rightarrow \lambda$ do
            if $f(x_i) < f(x'_{i,j})$ then
                $x_i \leftarrow x'_{i,j}$
            end if
        end for
    end for
end while
```

3.3 Covariance Matrix Adaptation

The basic Evolution Strategies do not take into account whether a function is separable, or non-separable. Since dependencies between variables may exist, learning these dependencies can greatly improve the convergence rate. CMA-ES uses a covariance matrix C, that is learned over time, to map dependencies between variables, and uses this information to mutate new vectors. This matrix is updated each iteration, using improvements discovered through the search process.

Algorithm 4 The basic 1+1 CMA-ES algorithm

```
initialize $x_{parent}, \sigma, C = I, p_{succ} = p_{succ}^{target}, p_c = 0$
while Not terminate do
    Find $A$ such that $C = A \cdot A^T$
    $z \leftarrow \mathcal{N}(0, I)$
    $x_{offspring} \leftarrow x_{parent} + \sigma A z$
    updateStepSize($\sigma, \lambda_{succ}, p_{succ}$)
    if $f(x_{offspring}) \leq f(x_{parent})$ then
        $x_{parent} \leftarrow x_{offspring}$
        updateCov($C, Az, p_{succ}, p_c$)
    end if
end while
```

Algorithm 5 updateStepSize($\sigma, \lambda_{succ}, p_{succ}$)

```
$p_{succ} \leftarrow (1 - c_p) psucc + c_p \lambda_{succ}$
$\sigma \leftarrow \sigma \exp(\frac{1}{d}(p_{succ} - \frac{p_{succ}^{target}}{1 - p_{succ}^{target}}(1 - p_{succ})))$
```

Algorithm 6 updateCov(C, y, p_{succ}, p_c)

```
if $p_{succ} < p_{thresh}$ then
    $p_c \leftarrow (1 - c_c)p_c + \sqrt{c_c(2 - cc)}y$
    $C \leftarrow (1 - c_{cov}C + c_{cov}p_c pc^\top$
else
    $p_c \leftarrow (1 - c_c)p_c$
    $C \leftarrow (1 - c_{cov}C + c_{cov} \cdot (p_c pc^\top + c_c(2 - cc)C)$
end if
```

Here, x_{parent}, and $x_{offspring}$ are respectively the initial solution, and the newly generated solution. C is the covariance matrix, p_{succ} is the average rate in which improvements are found, p_{succ}^{target} is the target success rate that is desired, and p_c is the evolution path, used to update the covariance matrix. Finally, there are the constants $p_{thresh} < 0.5$, as well as c_c and c_{cov} such that $0 \leq c_{cov} < c_c \leq 1$
There are several variations of CMA-ES, which differ in the method used to compute the covariance matrix. Two variants of CMA-ES, and a variation on standard ES, are used in this thesis, and their performances are analyzed with the performance of the best DE variation, namely jaDE.

3.3.1 (1+1) Cholesky Covariance Matrix Adaptation

Since updating the Covariance matrix is computationally expensive, as factorizing a matrix requires $O(n^3)$ operations, a way to speed this process up using Cholesky factorization, in combination with a 1+1 strategy, is proposed [9]. The improved Cholesky factorization reduces time complexity to $O(n^2)$, by never explicitly computing the covariance matrix, but by only operating on Cholesky factors.

3.3.2 (1+1) Active Covariance Matrix Adaptation

Another way to update the covariance matrix is to add another update step, called the active update [2]. This update step is applied whenever a mutation yields an especially unsuccessful result. Especially unsuccessful is defined for the 1+1 strategy as having a lower fitness value than its k-th order ancestor. The effect is to make the mutations in the direction of failure rounder, reducing the likelihood to repeat the search in an unsuccessful place.

3.3.3 DR2

DR2 is an evolution strategy that adapts its mutation rate in a deterministic way, given a set of past mutations [12]. Whilst new individuals are still generated using a normal distribution, the step size for the next iteration is altered deterministically, given the fitness values of the original and newly mutated vector.

3.4 Comparison of Differential Evolution to Evolution Strategies

In the appendix E the performance comparison graphs of 1+1 Cholesky CMA-ES, 1+1 Active CMA-ES DR2, canonical DE and jaDE are shown. It is evident that DR2 does not perform particularly well in comparison. Both jaDE and 1+1 Cholesky CMA-ES perform well. The 1+1 Cholesky CMA-ES performs best of all strategies in 40 dimensions. However, the space complexity of the Cholesky CMA-ES is $O(N^2)$, whereas jaDE has a linear space complexity. Therefore, for large scale problems, jaDE might be the more suitable of the two algorithms.

Chapter 4

Mutation Oriented Differential Evolution and its Variations

After having compared some already existing optimization algorithms, a novel algorithm, based on DE, is proposed in this chapter. This algorithm is called moDE, and it works by skewing the mutation vector based on previously successful mutations. By keeping track of successful mutations, the preferential direction of mutation can be set, and this causes mutations to skew more towards the preferential direction. Compared to CMA-ES, this algorithm's space complexity is linear with respect to the function's dimensionality.

4.1 Mutation Oriented Differential Evolution

Whenever an individual is mutated in Differential Evolution, using the rule $x_i \leftarrow x_i + F * (x_j + x_k)$, the vector $F * (x_j + x_k)$ can be considered the mutation vector. The mutation vector is the vector that is added to an existing individual, to create a new individual. The idea of mutation oriented Differential Evolution (moDE) is to keep track of successful mutation vectors, and to shift the direction of each new mutation vector into the direction of previous successful mutations.
This is achieved, by keeping track of previous successful mutations for each solution in an orientation vector. Each solution's mutation vector x_μ contributes to the solution's local orientation vector *locorient* if the mutation vector results in an improvement. Hence, the orientation vector is the vector which represents the general direction of successful mutations. This idea of keeping track of the most promising areas in the search space is based on Particle Swarm Optimization [10], where individual solutions tend to converge towards the direction of the most successful solution. The skewing of a mutation vector is achieved as follows: first, the angle α between the orientation vector and mutation vector is computed. Then, the mutation vector x_μ is mutated as follows: $x_\mu \leftarrow x_\mu + (0.5\cos(\alpha) + 1) \cdot norm(locorient) \cdot x_\mu$, where $norm(v)$ is the Euclidean length of a vector v.
The only parameter that needs to be set here is the cooling factor c_{loc}, which determines the influence of past successful mutations as well as new successful mutations. c_{loc} determines the cooling factor for each individual's own orientation vector. A low cooling factor causes older mutations to become more quickly irrelevant. Conversely, a high cooling factor gives each subsequent successful mutation a smaller influence. In these experiments, the value chosen was $c_{loc} = 0.2$, as to prevent successful mutation directions from causing an individual to overshoot the global optimum.

Algorithm 7 The mutation oriented Differential Evolution algorithm

```
for $i = 1 \to NP$ do
    for $j = 1 \to D$ do
        $v_{i,j} \leftarrow rand$
        $mvec_{i,j} \leftarrow 0$
        $locorient_{i,j} \leftarrow 0$
        $xorient)_{i,j} \leftarrow 0$
    end for
end for
Set the parameter $c_{loc}$
while not terminate do
    for $i = 1 \to NP$ do
        $j = rand_{integer} \in [1, D] \setminus \{i\}$
        $k = rand_{integer} \in [1, D] \setminus \{i, j\}$
        $mvec_i \leftarrow v_i + F(v_j - v_k)$
        $k = rand_integer \in [1, D]$
        for $j = 1 \to D$ do
            if $rand \in [0, 1] < CRorj == k$ then
                $mvec_{i,j} \leftarrow v_{i,j}$
            end if
        end for
        $mvec_i \leftarrow mvec_i - v_i$
        if $norm(mvec_i) \neq 0$ then
            $mvec_i \leftarrow mvec_i + 0.5 \cdot (\frac{dot(mvec_i, locorient_i)}{norm(mvec_i) \cdot norm(locorient_i)} + 1) \cdot norm(0.5 * locorient_i) \cdot mvec_i$
        end if
        $x_i \leftarrow v_i + mvec_i$
    end for
    for $i = 1 \to NP$ do
        $locorient_i \leftarrow c_{loc} \cdot locorient_i$
        if $F(x_i) < F(v_i)$ then
            $v_i \leftarrow x_i$
            $locorient_i \leftarrow locorient_i + ((1 - c_{loc}) \cdot mvec_i)$
        end if
    end for
end while
```

The transformation of the mutation vector is based on the shape of a sphere. Firstly, the transformation vector is added to half of the orientation vector. The scale factor is based on the angle of the mutation vector with the orientation vector.

The angle θ between two vectors is computed using the norm and dot product. From [5], the formula $\cos\theta = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|}$, using the norm and the dot product of a vector. By taking the arccosine, we can retrieve the angle.

4.1.1 Target to best

One proposed variation to the standard moDE algorithm is to change the update rule. The update rule in standard moDE is moDE/rand/1, which corresponds to adding a random vector to an individual. Since moDE is designed to maximize the likelihood that a subsequent mutation will be directed towards a successful region, it is expected that changing the update rule to moDE/target-to-best/1 would cause functions to converge more quickly towards the optimum.

4.1.2 Combining Mutation Oriented Differential Evolution with jaDE

The main limitation of moDE is that the parameters are all fixed, and whilst one parameter setting may be optimal for some functions, it need not be the optimal for other functions. Recall that the strength of jaDE lies in the fact that all parameters used for mutation are learned, and thus become finetuned for each specific function. As a consequence, jaDE is a very powerful optimization algorithm.
To overcome the fixed parameters problem, moDE is modified to allow all parameters needed for mutation to be learned during run time, just like jaDE. From this idea, a hybrid algorithm of both jaDE and moDE is made. This algorithm uses the principles of jaDE to mutate its mutation parameters. This should cause the performance of the algorithm to be less dependent on its initial values, because the algorithm learns the optimal values for its mutation parameters during run time. Like jaDE, an archive of size NP is used to keep an additional list of potential vectors that can be added to the already existing vector, and the values for F and CR are also mutated in the same way as jaDE. Like moDE, the algorithm keeps track of successful directions of mutation, and will skew the mutation vector towards the direction of previously successful mutations. However, unlike moDE, the hybrid algorithm also mutates the influence that the orientation vector has. The variables that keeps track of the right amount of influence of the orientation vector is mutated and sampled using a Cauchy distribution, to prevent stagnation. Since this algorithm is a hybrid of moDE and jaDE, it will be called mojaDE in the remainder of this thesis.

4.2 Results

For the performance graphs of the three novel algorithms, as well as a comparison with jaDE, please see Appendix D. The performance graphs show that regular moDE is the best of the novel strategies, better than target to best, and mojaDE. Regular moDE was subsequently used in a comparison with jaDE in the LSGO experiments, the results of which are in Appendix F. However, it is clear that jaDE is the overall winner of the strategies in the BBOB benchmark, as it performed better throughout most experiments. It is also worth noting that mojaDE performed extremely poor. It seems that mojaDE converges prematurely, and gets stuck in local optima. Therefore, combining the ideas of jaDE and moDE did not turn out well in this case. Also, moDE does not scale well to higher dimensions, as it does not converge for some problems in LSGO, whereas jaDE converges for all LSGO problems.

Chapter 5

Results

This thesis has explained the algorithms Differential Evolution, its variants jaDE and moDE, which was conceived by the author, as well as Evolution Strategies and some variants, namely the (1+1) Cholesky CMA-ES, the Active (1+1) CMA-ES and DR2. In this Chapter, a comparison between the algorithms, as well as a discussion on the results, is made. Finally, conclusions drawn from the work are presented, and some topics for future research are proposed.

5.1 Performance Analysis

When looking at the data obtained from BBOB, it becomes clear that DE requires different initial settings to obtain maximum convergence and optimal results for problems with few dimensions (less than ten), than problems with higher dimensions (twenty or more). This is made clear when looking overall at the various different types of functions. For instance, in Appendix A, the best obtained results for $D = 40$ with canonical DE were obtained with different parameters than for $D = 2$; for all experiments run, population size of $3 \cdot D$ performed best in $D = 40$ whereas a population size of $9 \cdot D$ performed best for $D = 2$. Also, the crossover factor $F = 0.8$ performed better in higher dimensions, and worse in lower dimensions.

Appendix B shows the performance of different update rules for DE. Remarkably, two strategies performed consistently better than the others, namely Canonical DE and DE/best/2[1], where two random vectors are added to the best found individual. Possibly, DE/best/2 has this performance because it searches more in an area near the best solution found so far, yet because it has a larger degree of randomness than DE/best/1, it can avoid being stuck in a local optimum. It must also be noted that DE/best/1 performed badly, and in higher dimensions was the worst overall strategy.

After having chosen the best DE variants, namely Canonical and DE/best/2, its performance was compared to jaDE in Appendix C. In lower dimensions, jaDE performs worse. However, as the number of dimensions increases, so does the relative performance of jaDE over the other DE variants, and for $D = 40$, it is the best performing DE variation. Since problems with a larger number of dimensions are more difficult to optimize than problems with a smaller number of dimensions, jaDE is used as the benchmark for comparing the algorithms proposed by the author.

In Appendix D, four strategies are compared, namely jaDE and three novel strategies proposed by the author: moDE, mojaDE and moDEttbest. mo stands for mutation oriented, as the strategy increases the likelihood of a subsequent mutation to be skewed towards the direction of a successful previous mutation. These strategies seem to behave more like a greedy algorithm when looking for optima, as the results show. In higher dimensions jaDE is the best algorithm. Only for $D = 2$ and $D = 3$ does moDE outperform jaDE. The variant mojaDE, where the mutation of a vector obtained by jaDE is also transformed to appear more similar to previous successful mutations, performed terribly. This results of this experiment, clearly show that to tackle problems in higher dimensions, information obtained in the first few mutations must be applied gradually.

Appendix E compares the performance of Canonical DE, and jaDE to three Evolutionary Strategy types, namely Cholesky CMA-ES, Active CMA-ES and DR2. For $D = 40$, Cholesky CMA performs significantly

[1]The meaning of this notation is explained in Chapter 2

better than the other strategies. This advantage is lost in lower dimensions, and for $D = 2$, it is not even the best performing strategy. However, DR2 shows consistently poor performance.

5.1.1 DE compared with CMA-ES

As can be seen in Appendix E, Cholesky CMA-ES is the best performing algorithm tested in this paper using the BBOB benchmark for $D = 40$. However, it is worth noticing that jaDE is the second best performing algorithm of all algorithms analyzed. Moreover, jaDE has a space complexity of $O(D)$ whereas all CMA-ES algorithms have a space complexity of $O(D^2)$. This means that for extremely large scale programs, storage can become an issue. The DE algorithms were also tested using another benchmark, namely LSGO, or Large Scale Global Optimization. For this thesis, each algorithm was run on each LSGO function for ten runs. So whilst Cholesky CMA-ES does have a much better overall convergence rate, its space complexity can cause problems for problems with a larger dimensionality. Since jaDE also does perform really well, better than some other CMA-ES variants, but only has a linear space complexity, it was chosen to run in LSGO, alongside moDE. Hence, jaDE can be considered to be overall as good as Cholesky CMA-ES, due to its better scalability.

5.2 Conclusion

The attempt to improve jaDE failed. However, this does establish even more firmly that jaDE is a very powerful optimization algorithm. Moreover, jaDE was compared to some variants of CMA-ES. The Cholesky CMA-ES algorithm performed best of all, but jaDE was not much worse. Finally, since jaDE has a better space complexity than any CMA-ES algorithm, jaDE might be a more preferable algorithm to Cholesky CMA-ES, due to less overhead and space constraints.
Now the thesis will answer the main thesis question: *what are the strengths and weaknesses of Differential Evolution, how can its drawbacks be overcome, and why does it have its strengths and weaknesses?* The obvious weaknesses of DE is that, except for jaDE, most parameters are fixed during runtime. Moreover, none of the algorithms learn potentially existing dependencies between variables, unlike CMA-ES. However, the DE algorithms are all very easily parallelized, and their space complexity is linear in the problem dimensionality. Furthermore, the search space around a set of solutions in DE has a similar shape to the shape of the already existing solution set. This implicitly makes searches more likely into a promising direction. Moreover, jaDE overcomes most of the DE drawbacks, by learning all internal parameters, rather than keeping the fixed throughout runtime. This makes jaDE a good overall optimization algorithm, for problems with both low and high dimensionality.

5.3 Potential Future Research

Whilst a small subset of algorithms has been analyzed, several things that could be investigated in the future remain. For instance, is there a way to improve jaDE so that it performs similarly to Cholesky CMA-ES, if not better? Are there other algorithms with a linear space complexity with comparable performances and convergence rates, perhaps based on a different paradigm?
The GECCO conference that uses BBOB benchmark does not specify any constraints on space complexity. Perhaps an optimization algorithm contest where space complexity is capped could offer an interesting alternative to the current algorithm constraints imposed at GECCO.

Appendix A

Canonical Differential Evolution Performance Graphs

In this appendix, the performance graphs of all experiments run with Canonical Differential Evolution are shown. Three population sizes were chosen, ($p \in 0.3, 0.6, 0.9$) with the other parameters remaining the same ($CR = 0.5$ and $F = 0.4$, together with data from population $p = 0.3$ with a higher crossover rate $CR = 0.8$. Each group of six graphs shows the results for an identical dimension size, displayed in the top left corner of each graph.
The performance graphs show the success rate of a specific strategy, measured over a group of functions, the type being labeled on top of each graph. The horizontal axis shows the number of function evaluations divided by the number of dimensions, in log scale. The verical axis displays the proportion of functions that has converged. The bottom right graph shows overall performance, over all 24 BBoB function. Finally, beneath each group of graphs, the meaning of the labels behind each performance line on each graph is explained.
For most comparisons and conclusions, the performance of all functions for $D = 20$ and $D = 40$ were used, to assess overall performance, where the performance for $D = 40$ weighed more strongly than the performance in $D = 20$.

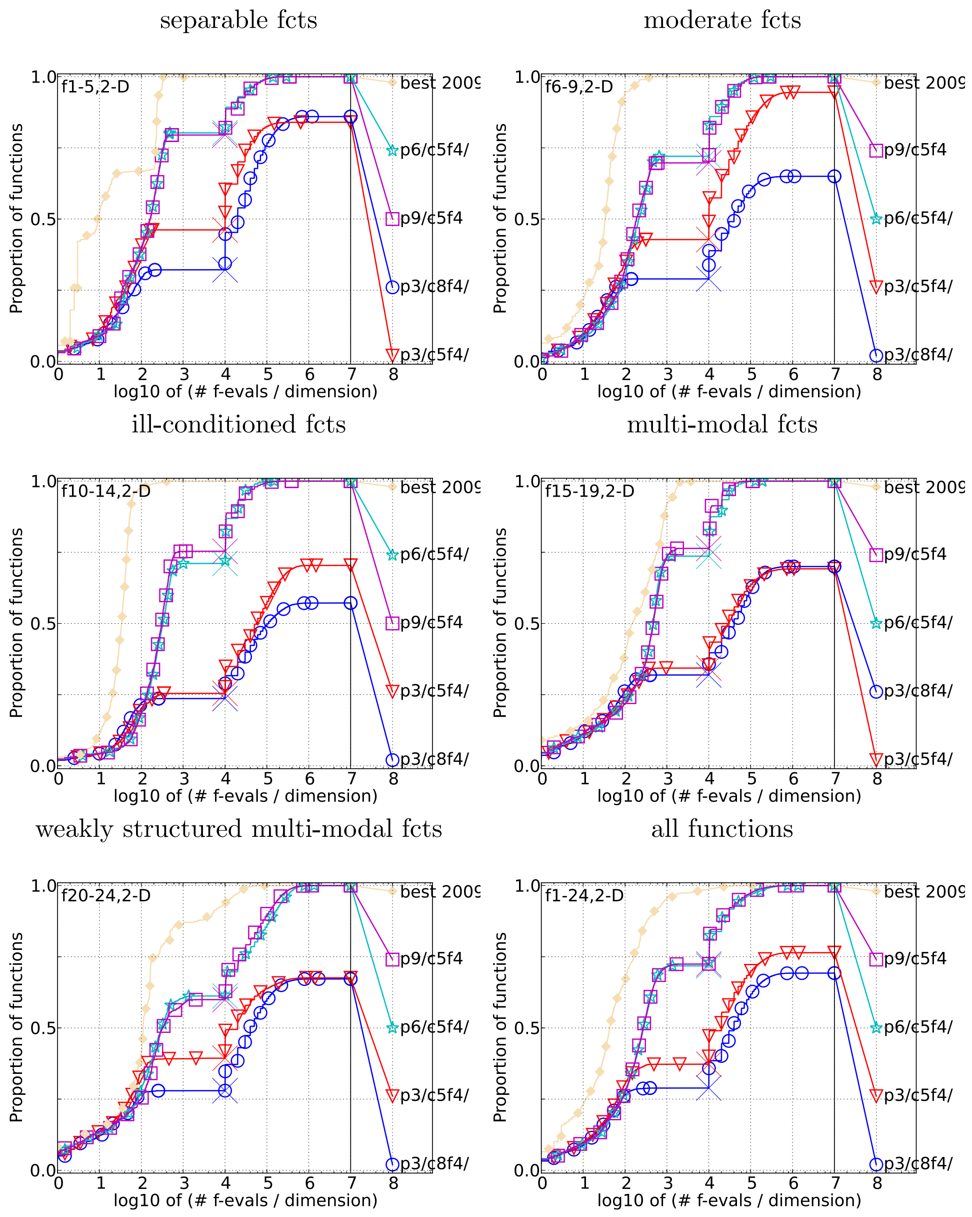

Figure A.1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\dots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

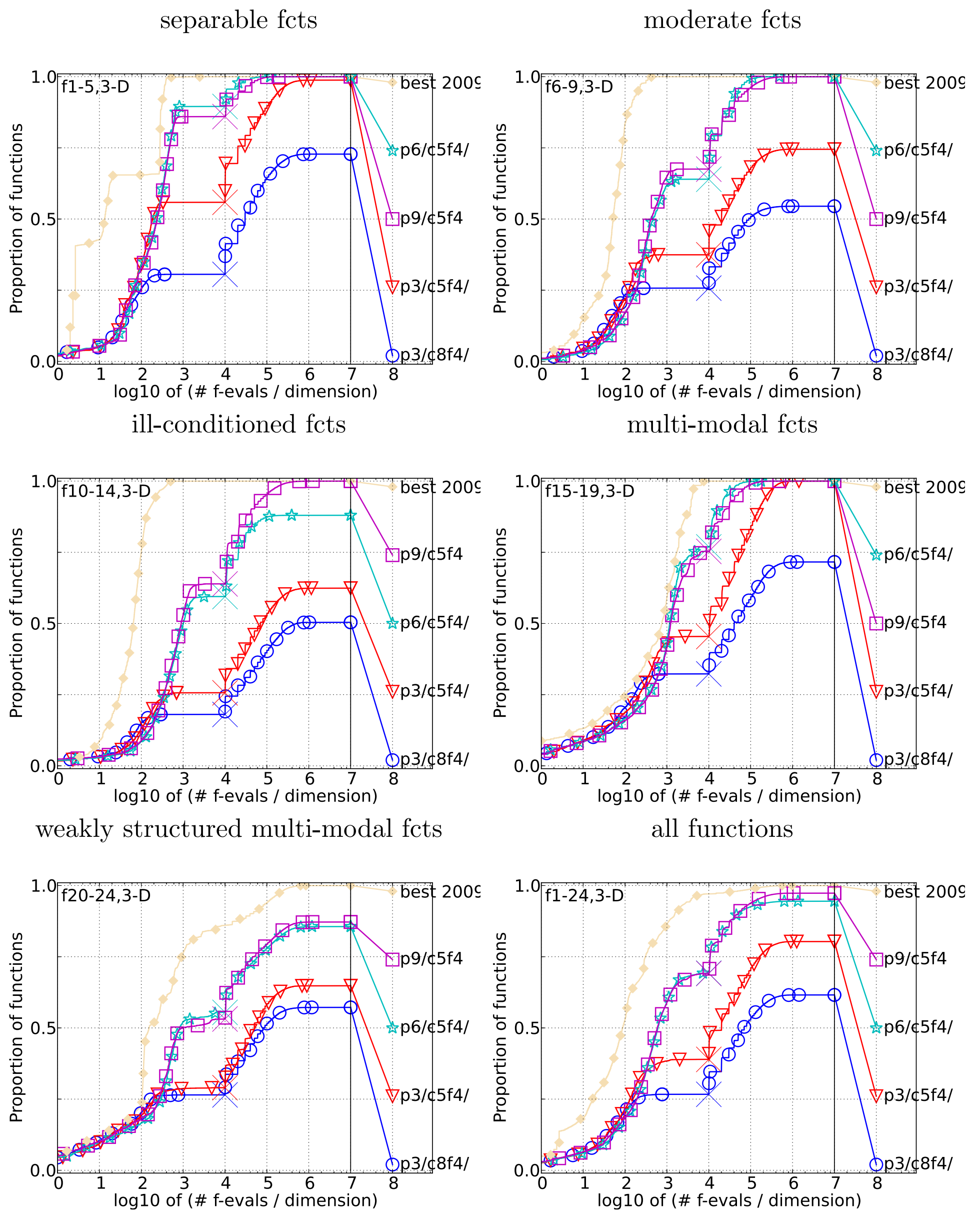

Figure A.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\ldots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

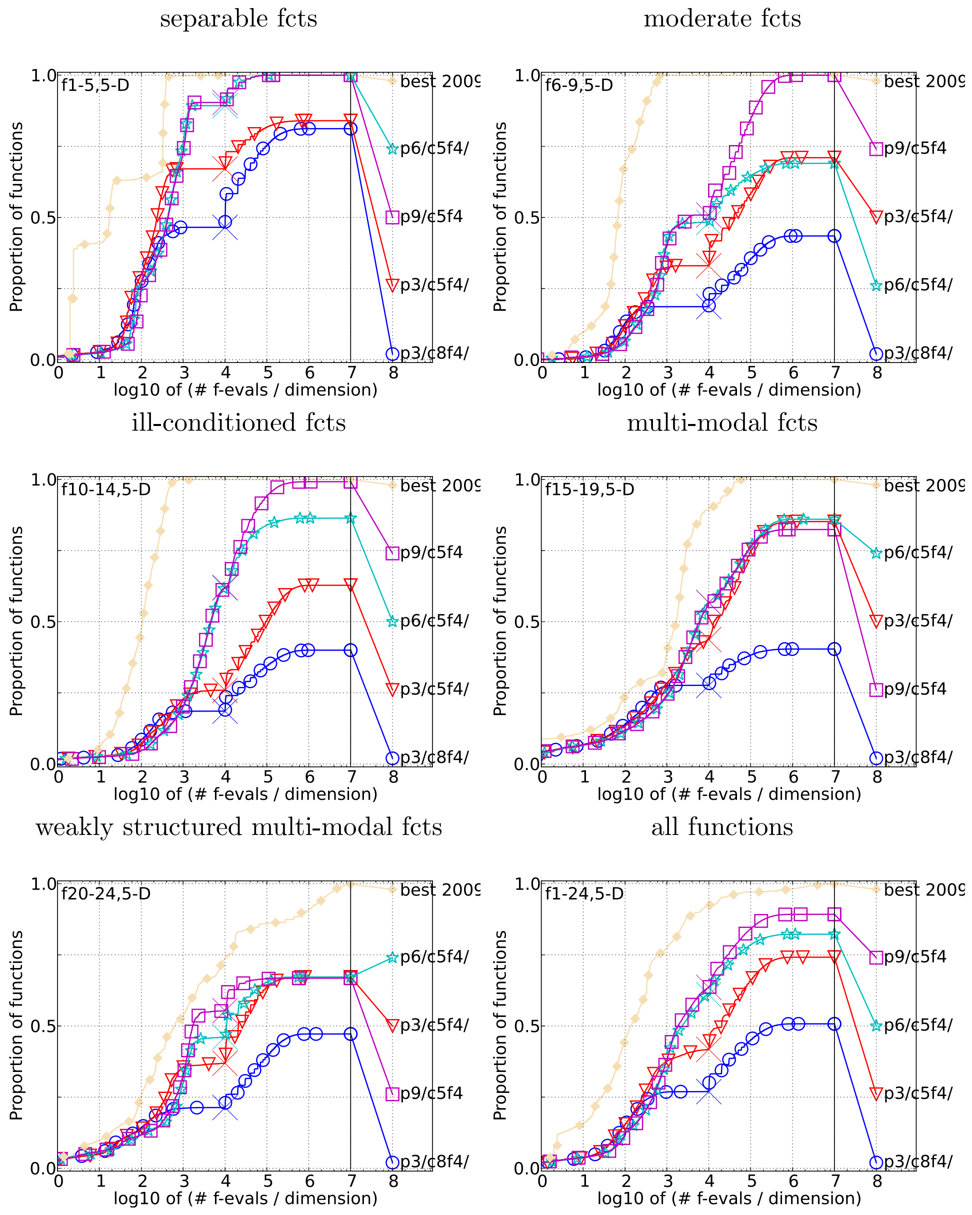

Figure A.3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\ldots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

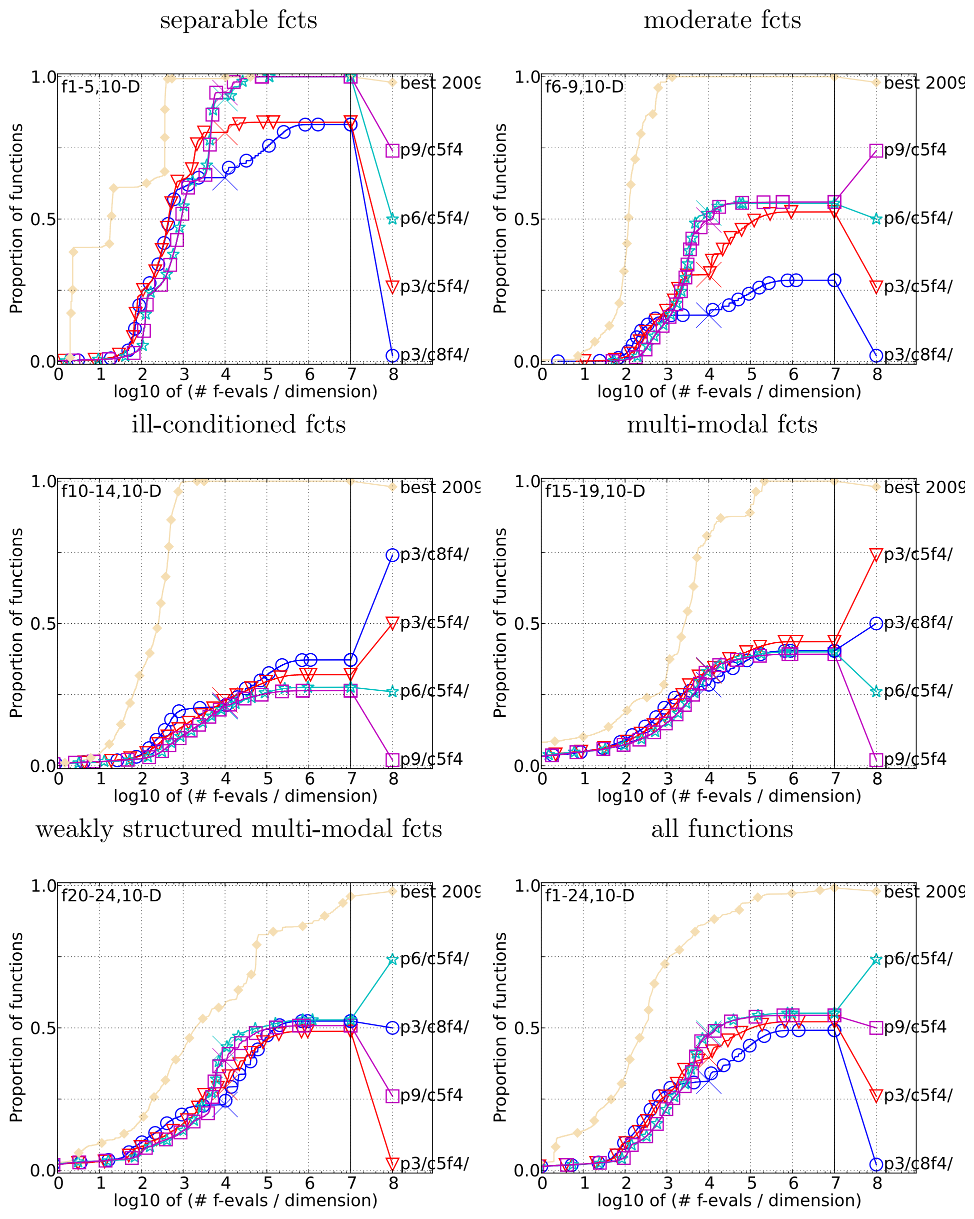

Figure A.4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\dots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

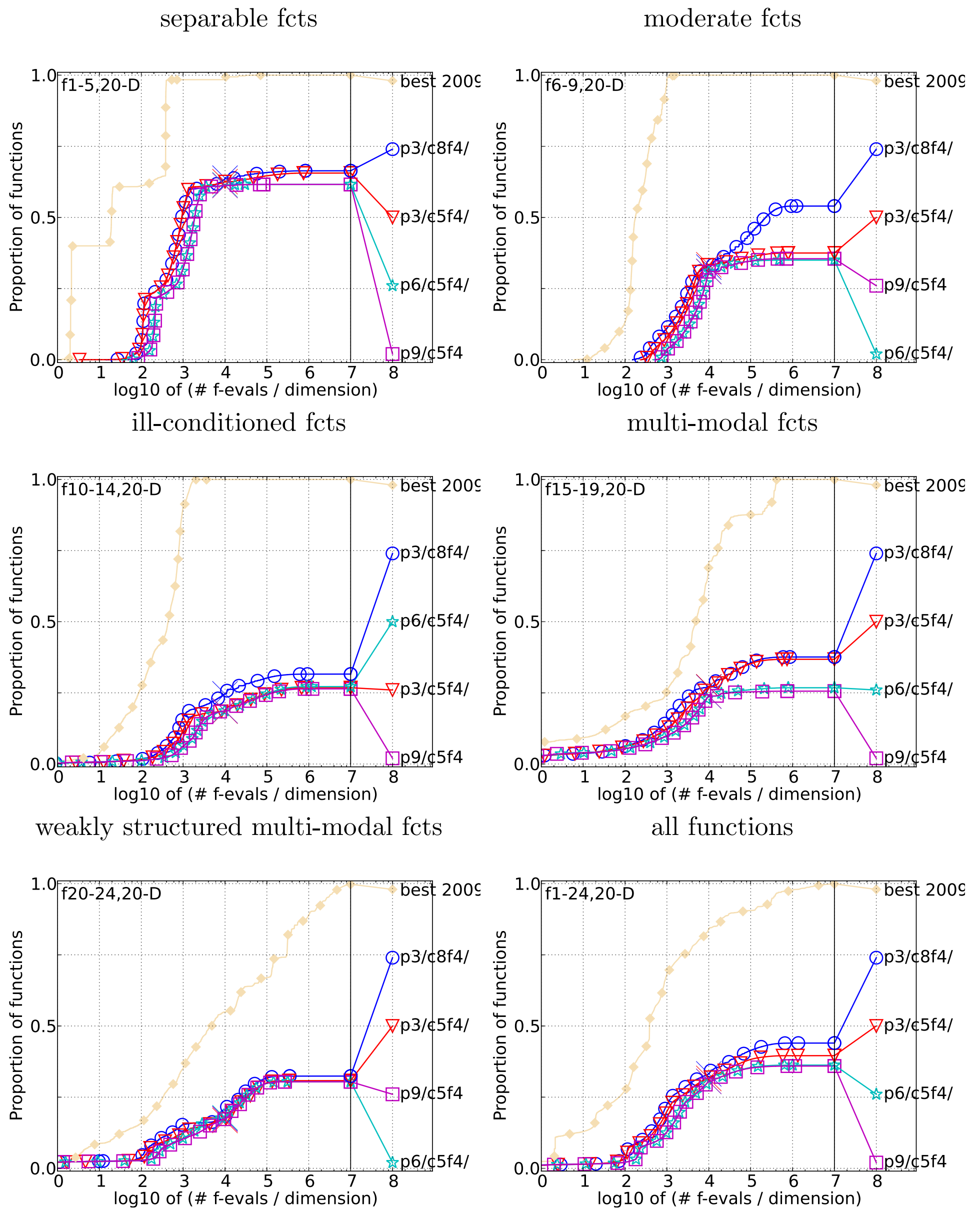

Figure A.5: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\ldots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

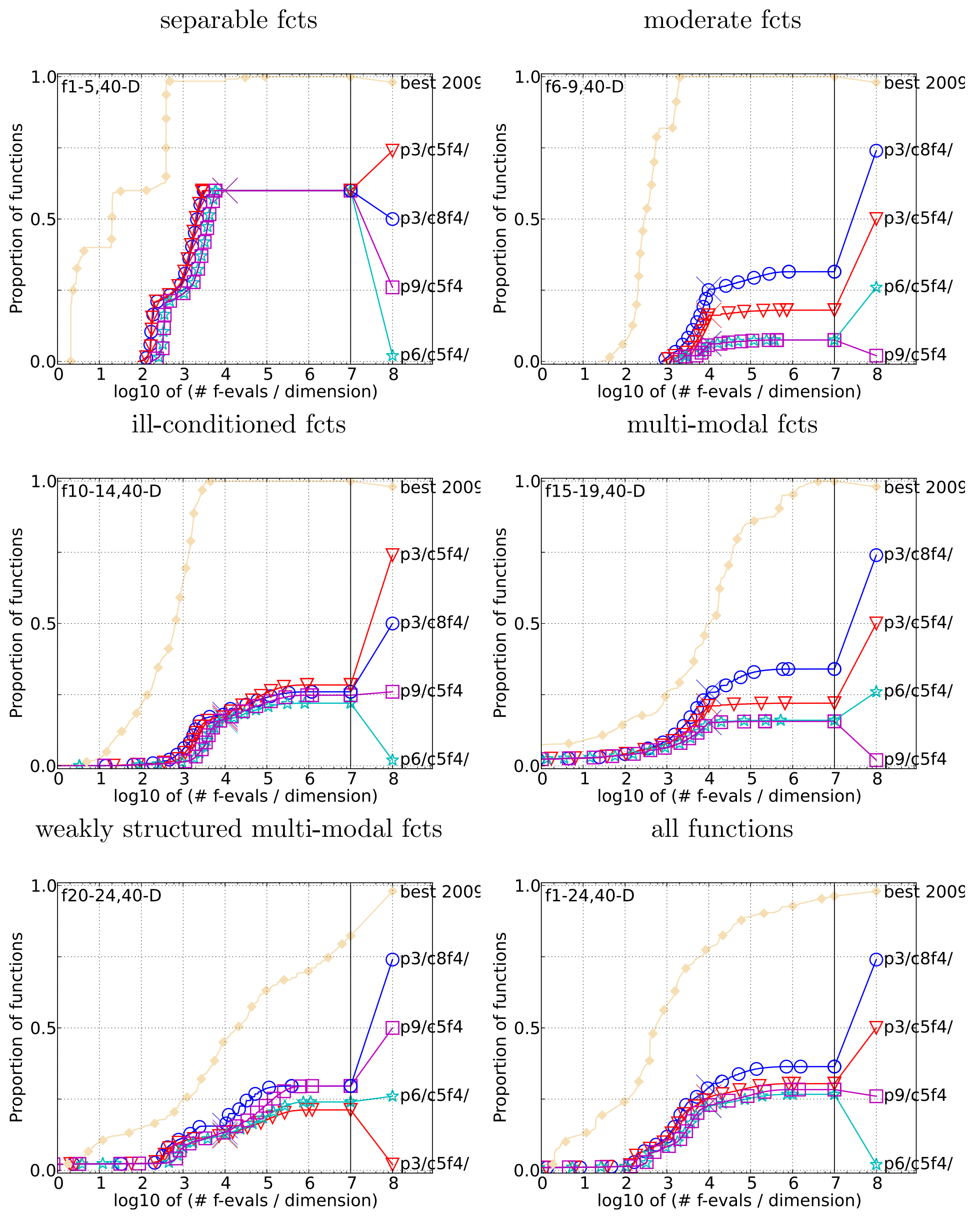

Figure A.6: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target. The number after p indicates the population size e.g. $P3/\dots$ corresponds with $P = 3 * D$. The number after f is the crossover factor, where 1 corresponds with 1, 4 with 0.4 and 7 with 0.7. c is the crossover factor CR multiplied by 10.

Appendix B

Performance Graphs of some Variations of Differential Evolution Strategies

The graphs in this appendix show the performance of various Differential Evolution strategies, as described in Chapter 2. In all strategies, the crossover rate, population sizes and difference factor are kept equal, as to provide a relative comparison between strategies, as opposed to parameter comparison. The parameters are chosen to be the best performing parameters of Canonical DE.
The performance graphs show the success rate of a specific strategy, measured over a group of functions, the type being labeled on top of each graph. The horizontal axis shows the number of function evaluations divided by the number of dimensions, in log scale. The verical axis displays the proportion of functions that has converged. The bottom right graph shows overall performance, over all 24 BBoB function. Finally, beneath each group of graphs, the meaning of the labels behind each performance line on each graph is explained.
For most comparisons and conclusions, the performance of all functions for $D = 20$ and $D = 40$ were used, to assess overall performance, where the performance for $D = 40$ weighed more strongly than the performance in $D = 20$.

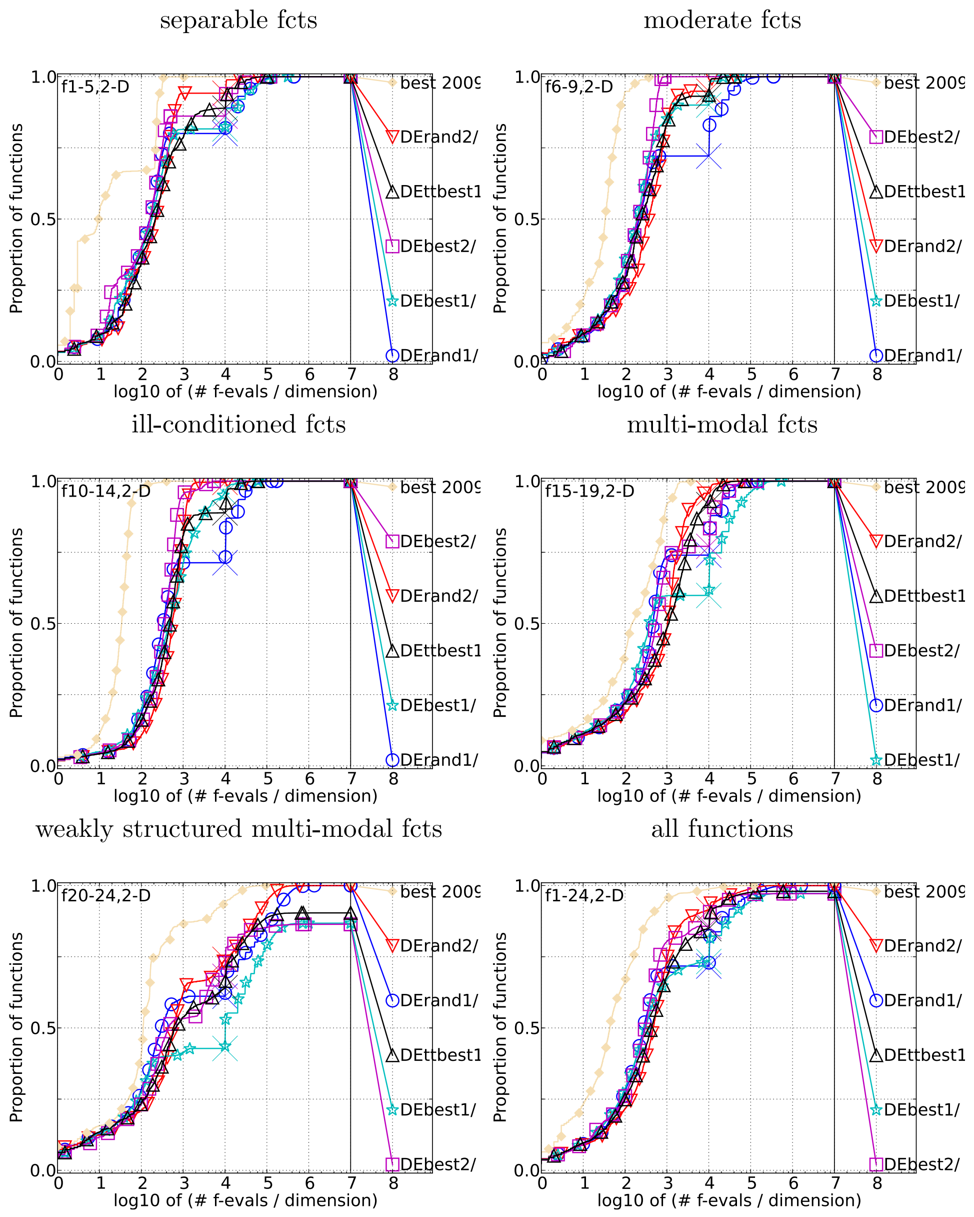

Figure B.1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

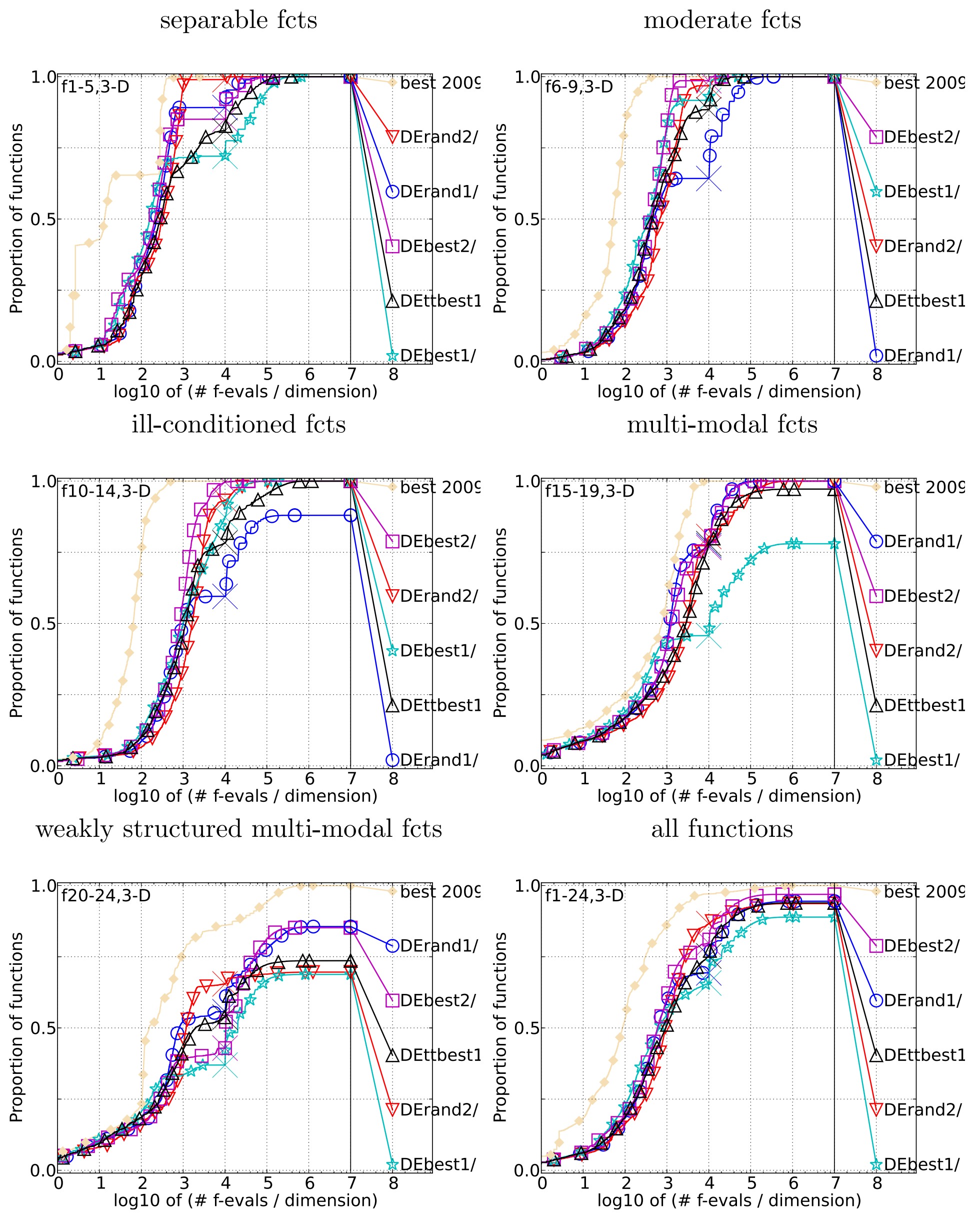

Figure B.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 3-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

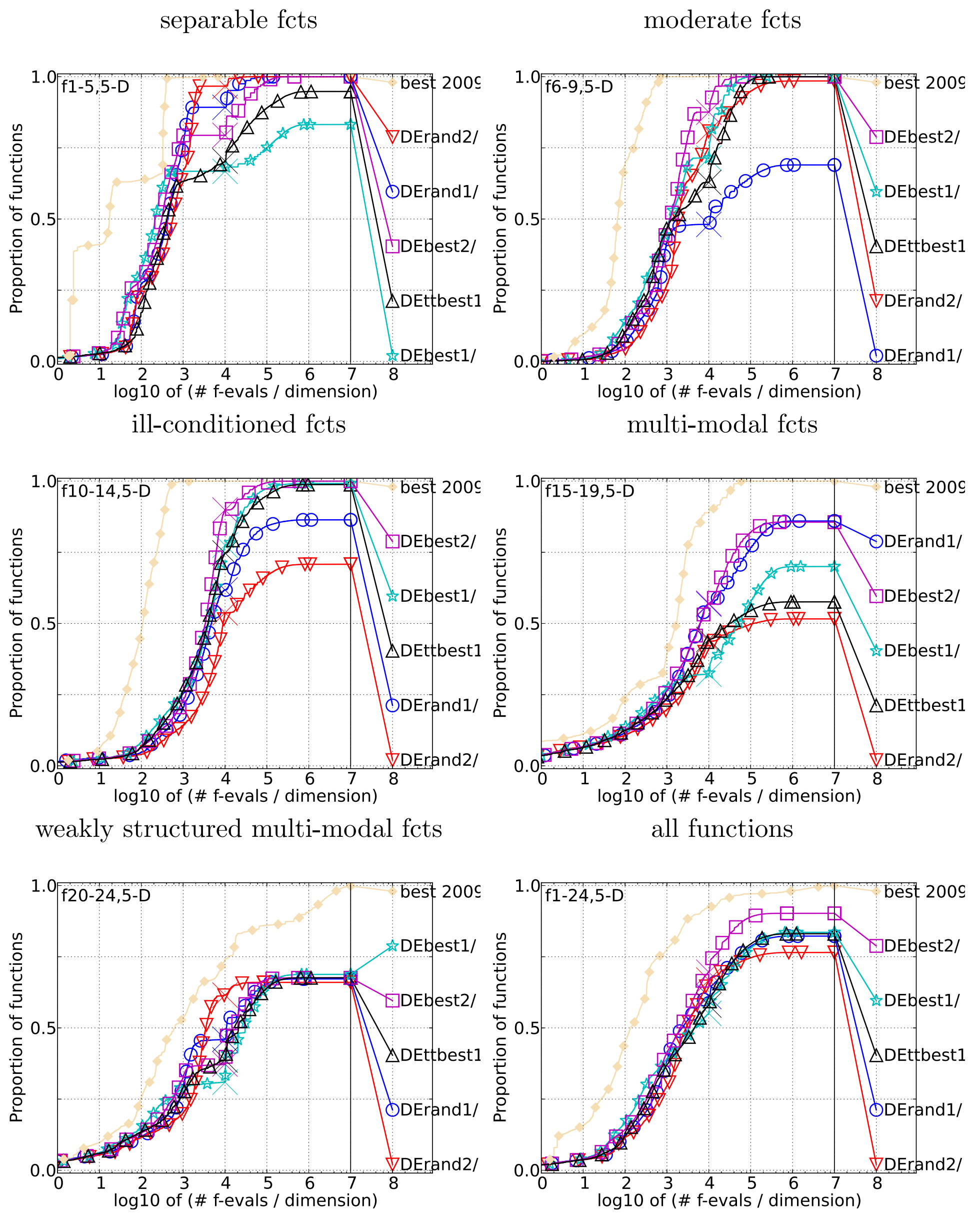

Figure B.3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

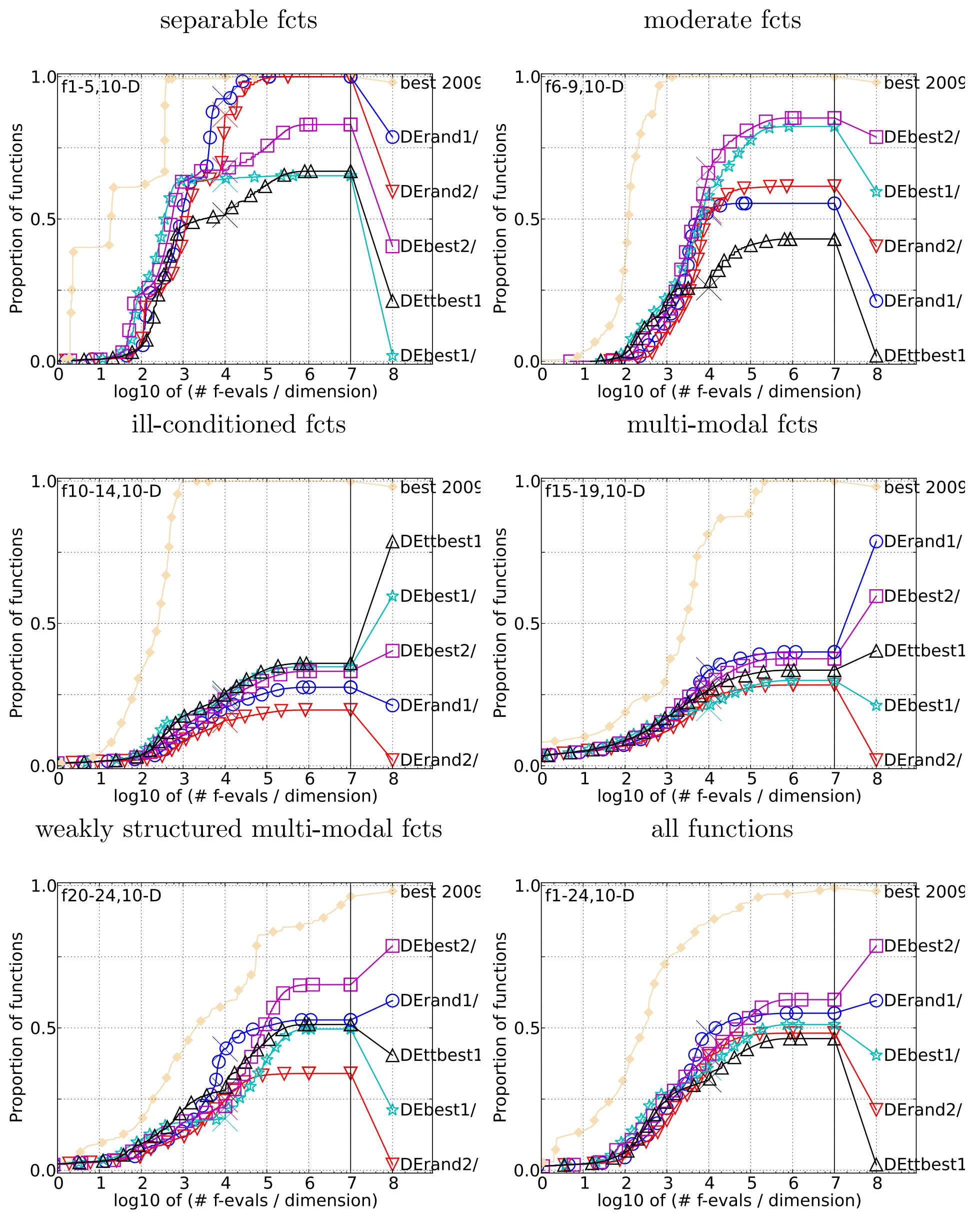

Figure B.4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 10-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

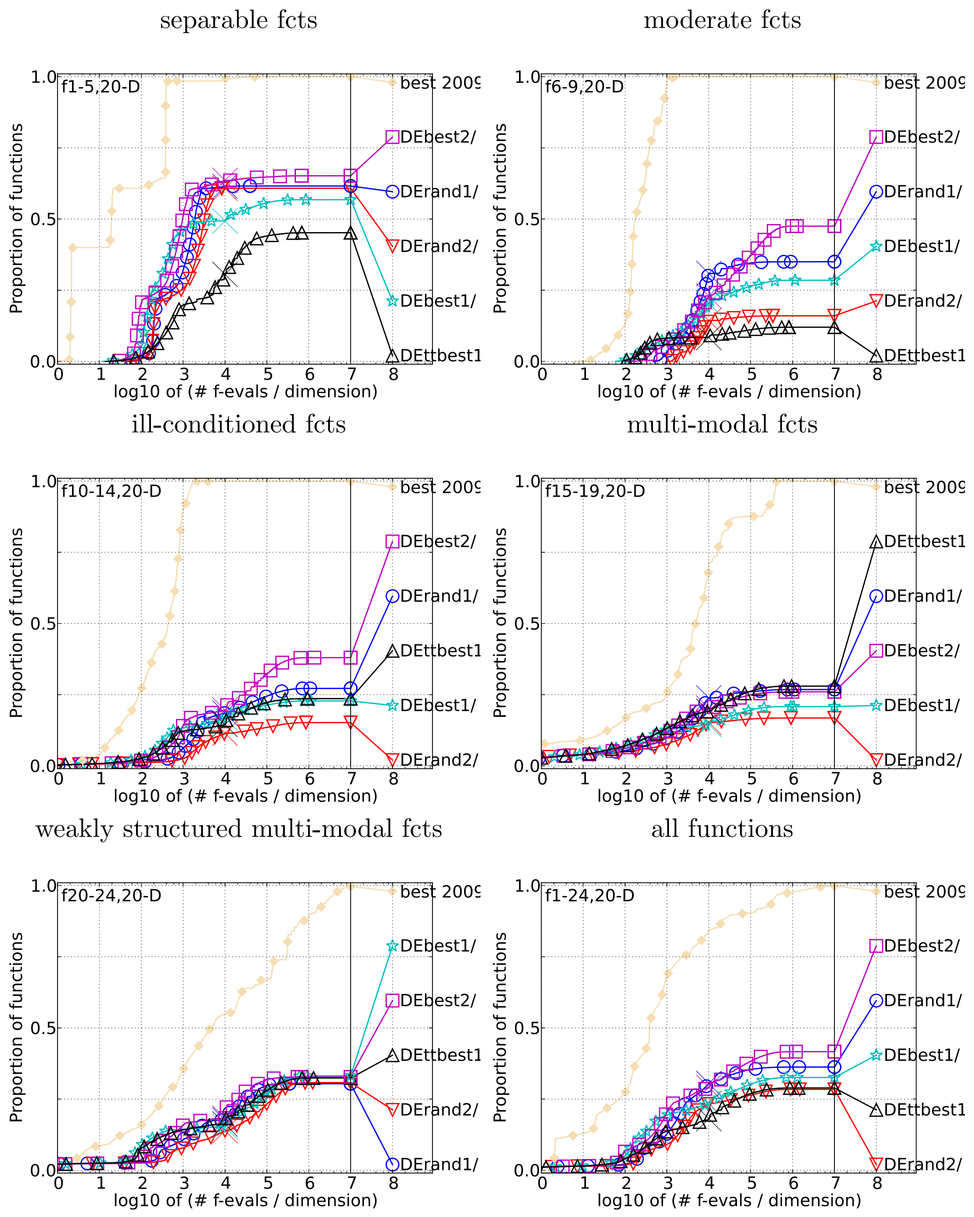

Figure B.5: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 20-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

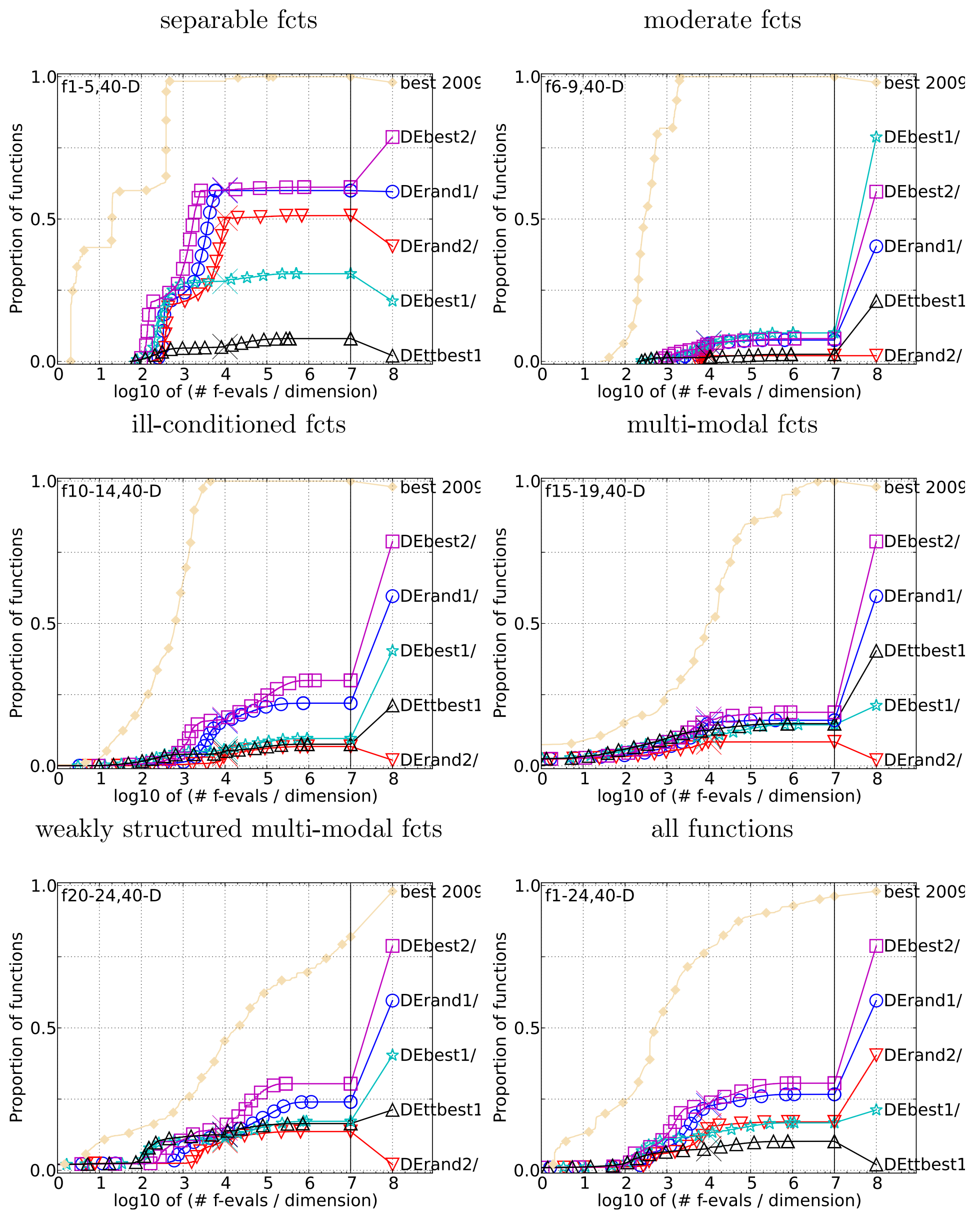

Figure B.6: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 40-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

Appendix C

Comparative Performance Graphs of JADE and the best DE Variants

Here, the performance graphs of jaDE and the best performing DE variants, with the same parameters as in Appendix B, are shown.
The performance graphs show the success rate of a specific strategy, measured over a group of functions, the type being labeled on top of each graph. The horizontal axis shows the number of function evaluations divided by the number of dimensions, in log scale. The verical axis displays the proportion of functions that has converged. The bottom right graph shows overall performance, over all 24 BBoB function. Finally, beneath each group of graphs, the meaning of the labels behind each performance line on each graph is explained.
For most comparisons and conclusions, the performance of all functions for $D = 20$ and $D = 40$ were used, to assess overall performance, where the performance for $D = 40$ weighed more strongly than the performance in $D = 20$.

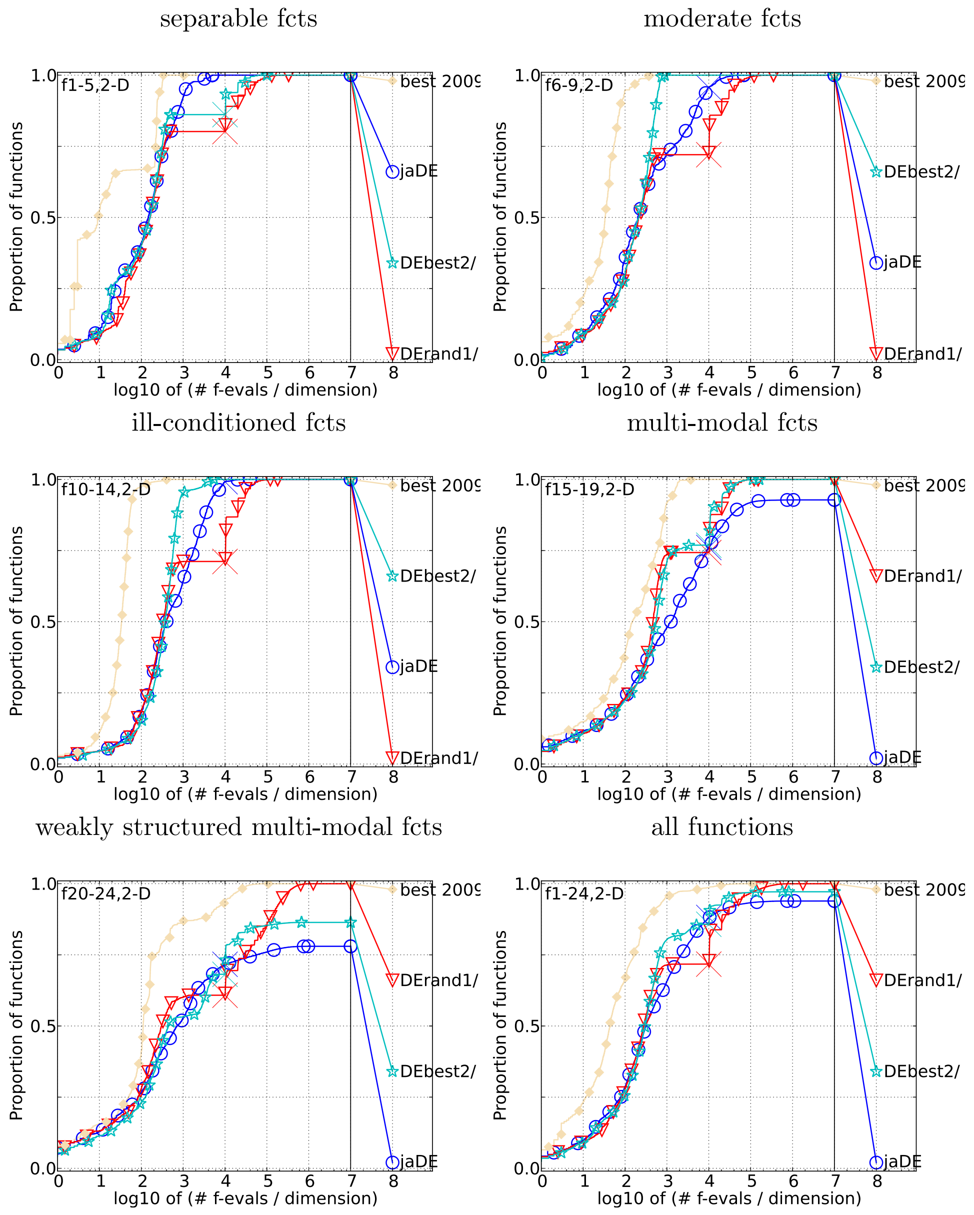

Figure C.1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

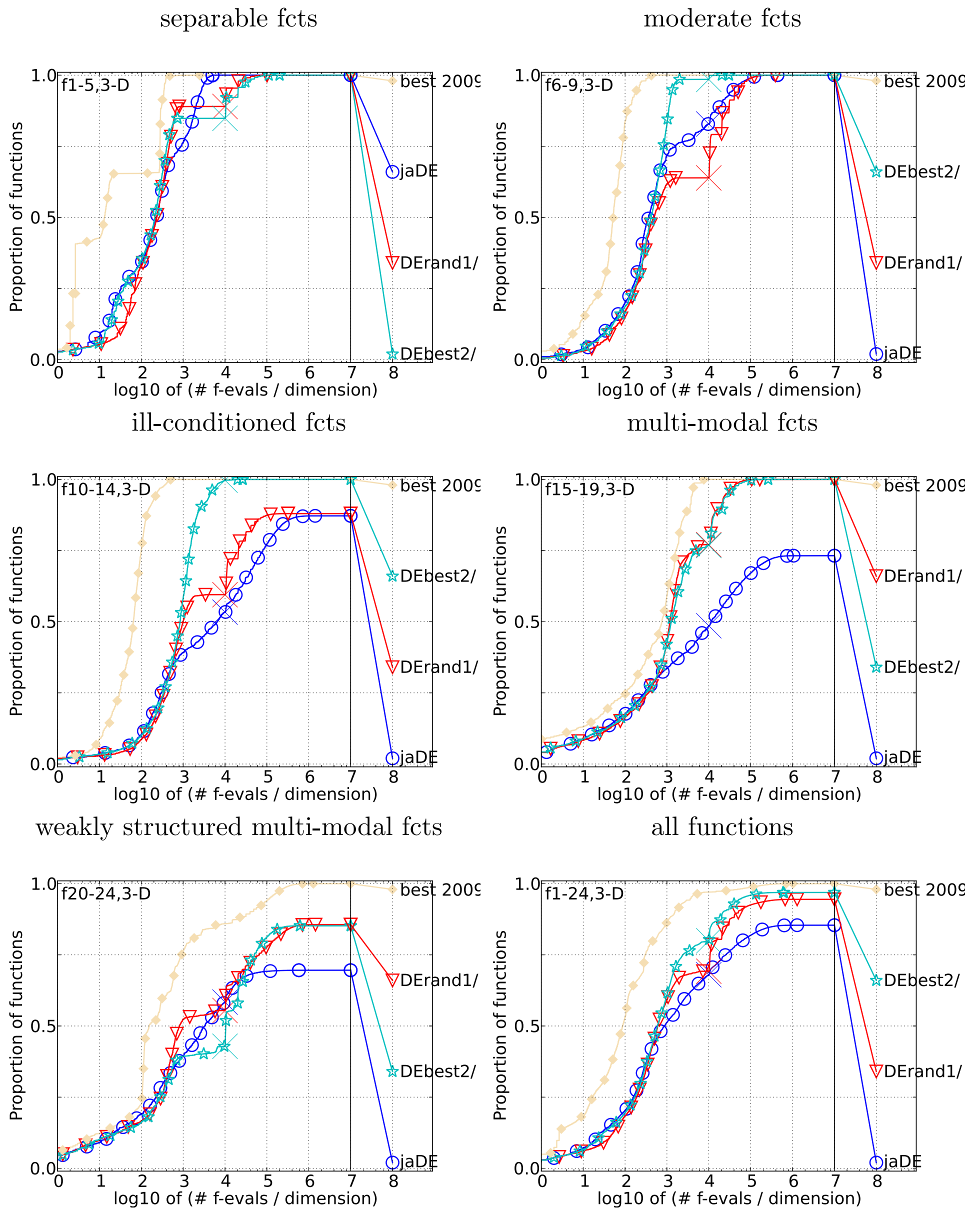

Figure C.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 3-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

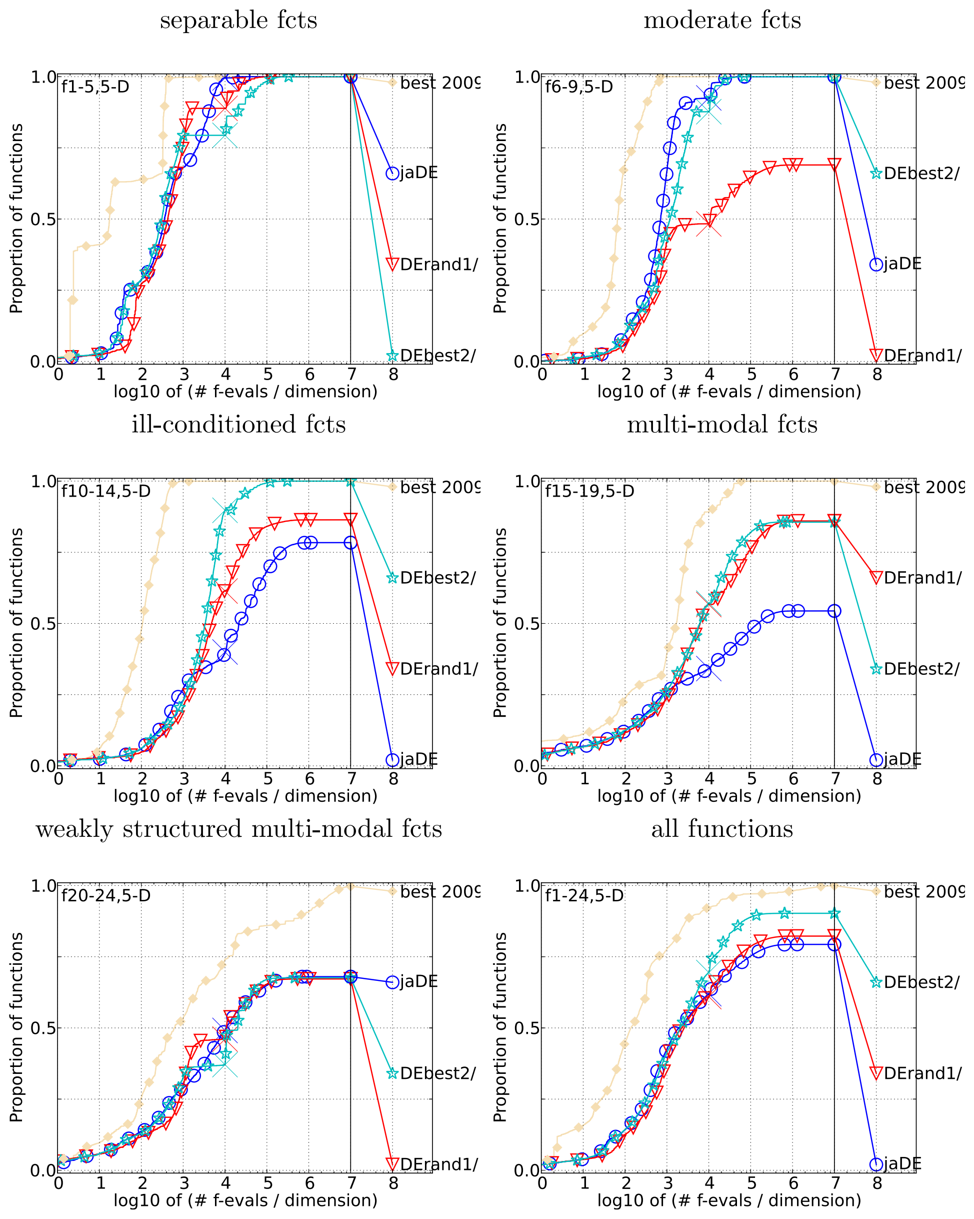

Figure C.3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

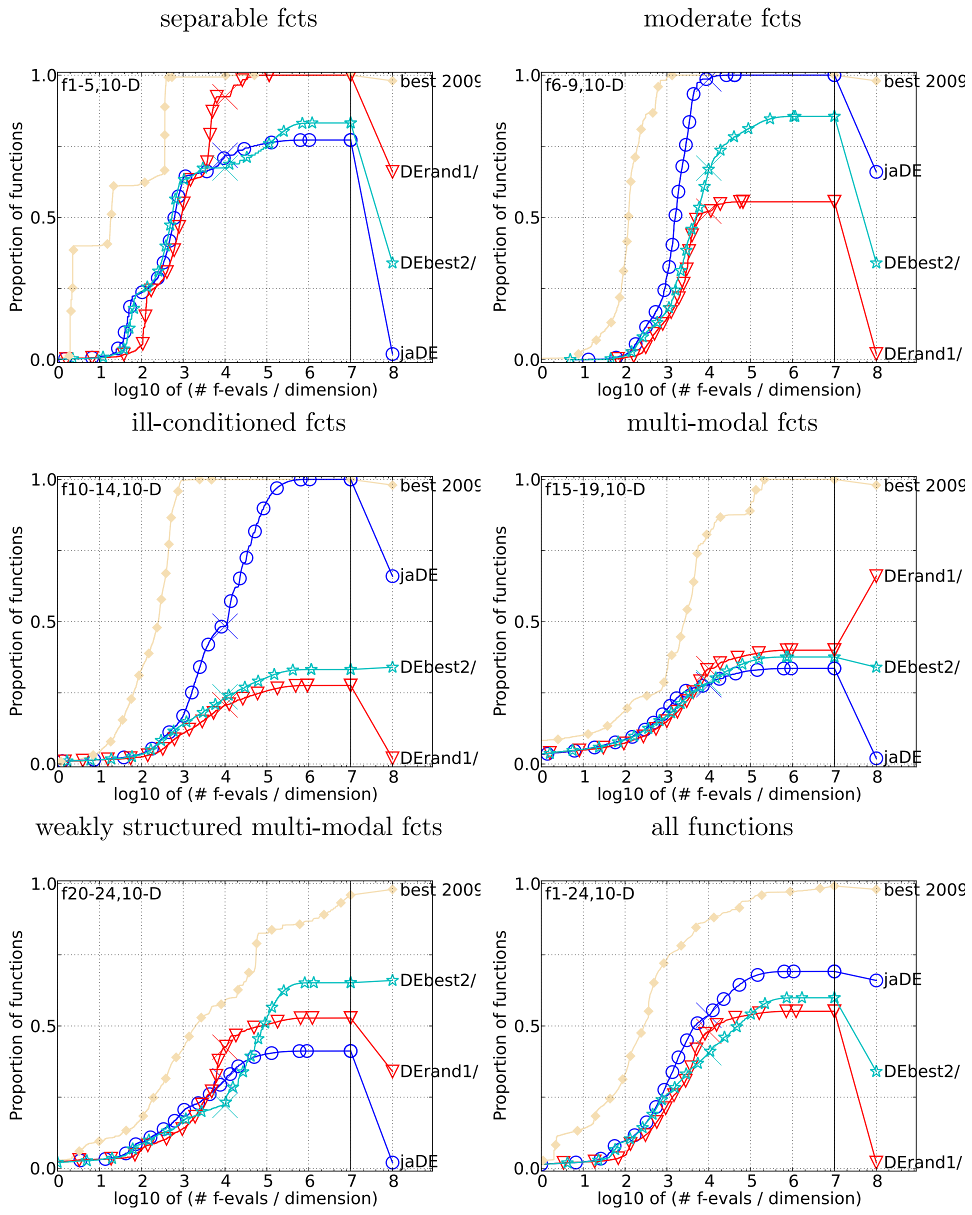

Figure C.4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 10-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

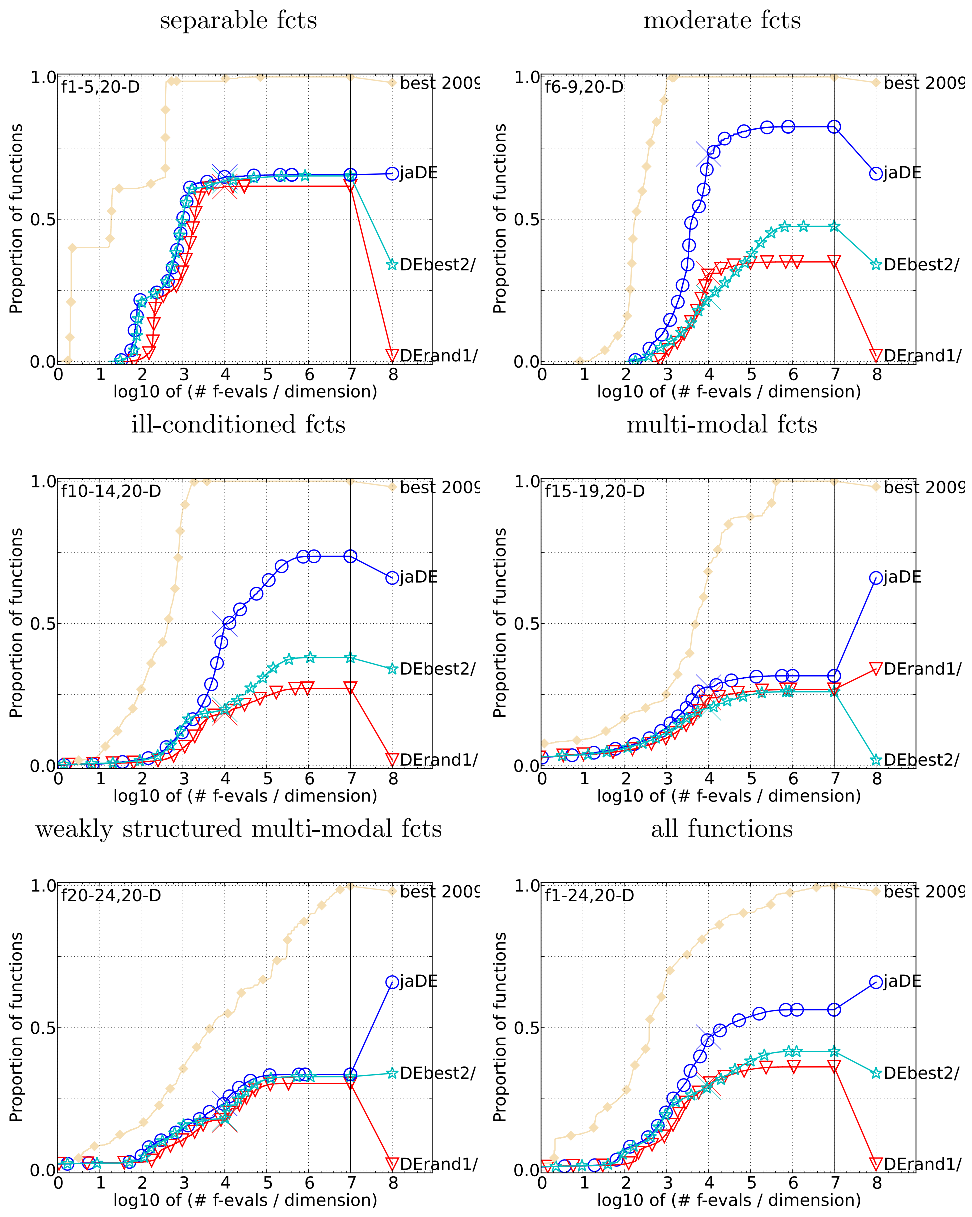

Figure C.5: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 20-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

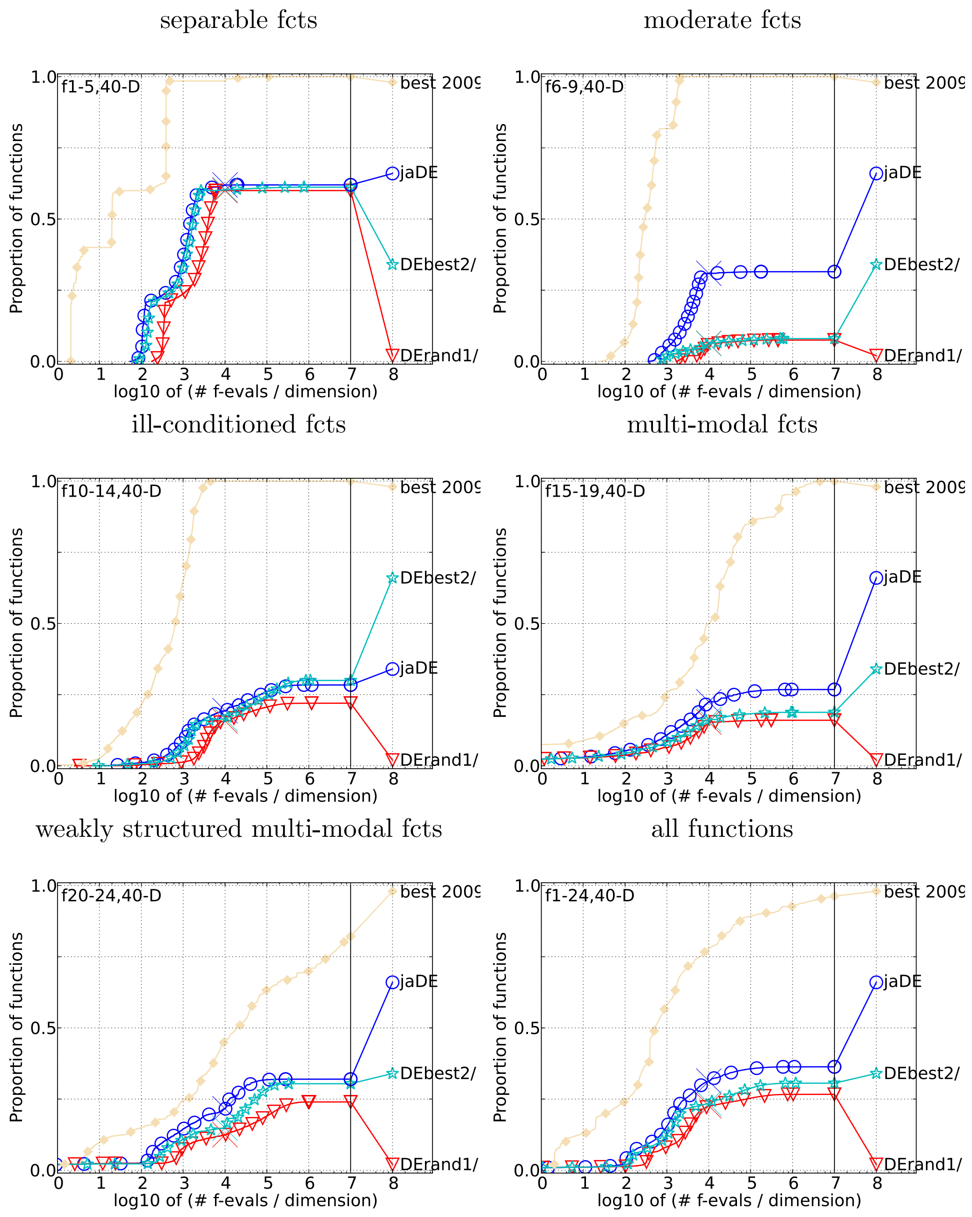

Figure C.6: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 40-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

Appendix D

Comparison Graphs of MODE, MOJADE and JADE

Here, the performance graphs of moDE, moDE with target to best strategy and mojaDE are shown. These are the novel alogrithms, inspired by Differential Evolution, jaDE, as well as particle swarm optimization. The performance of these novel algorithms is shown in this Appendix and compared with jaDE, that acts as a benchmark.
The performance graphs show the success rate of a specific strategy, measured over a group of functions, the type being labeled on top of each graph. The horizontal axis shows the number of function evaluations divided by the number of dimensions, in log scale. The verical axis displays the proportion of functions that has converged. The bottom right graph shows overall performance, over all 24 BBoB function. Finally, beneath each group of graphs, the meaning of the labels behind each performance line on each graph is explained.
For most comparisons and conclusions, the performance of all functions for $D = 20$ and $D = 40$ were used, to assess overall performance, where the performance for $D = 40$ weighed more strongly than the performance in $D = 20$.

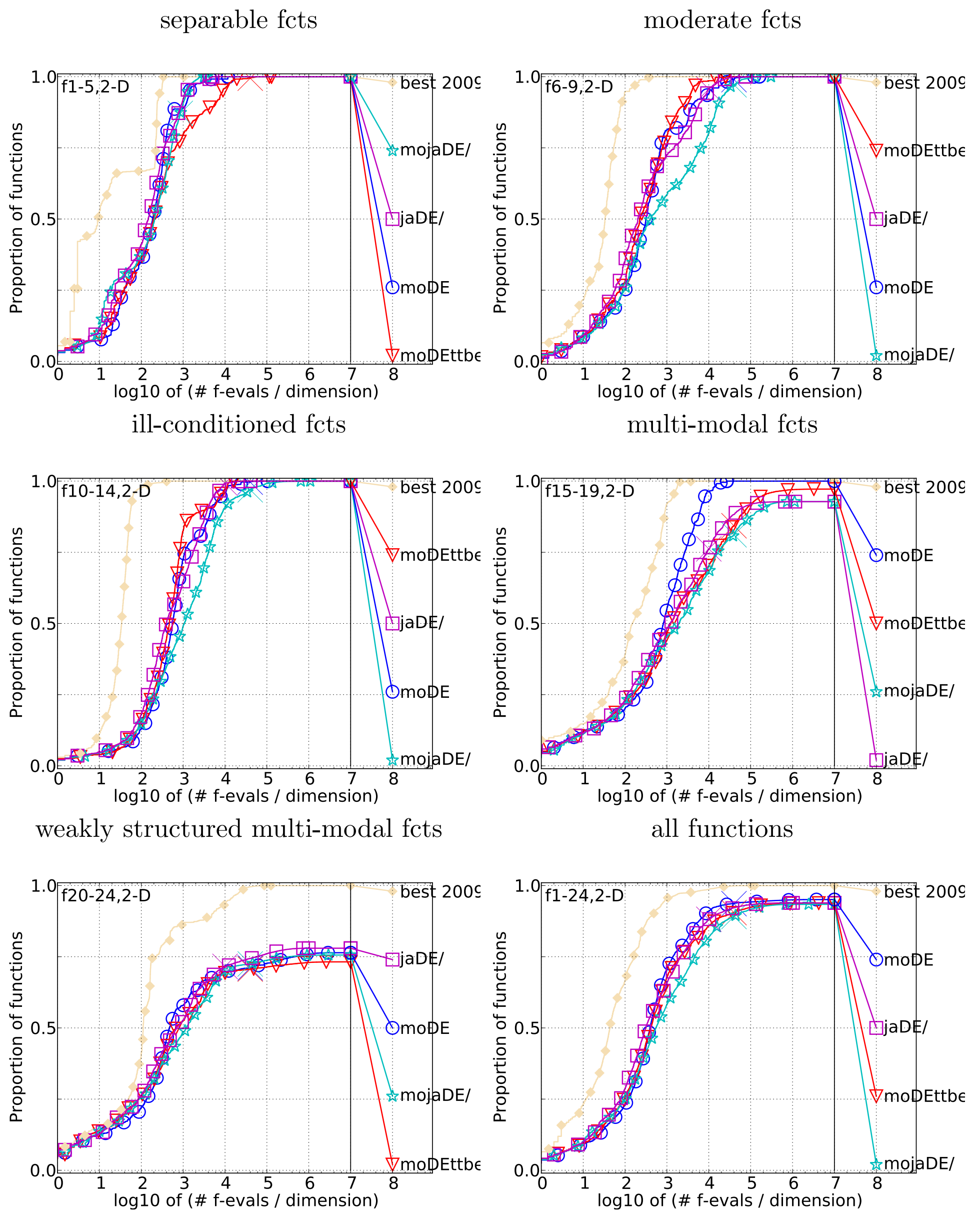

Figure D.1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

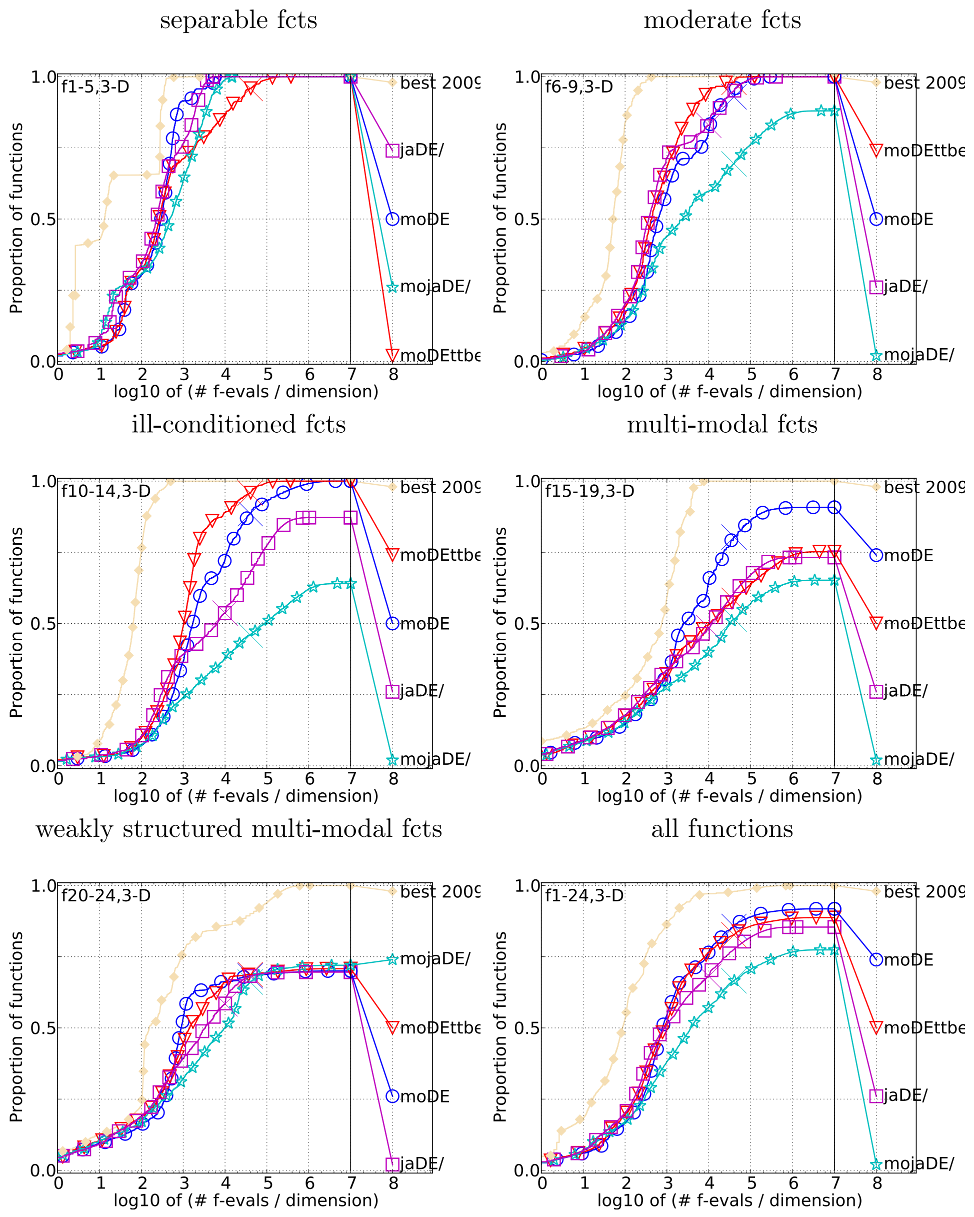

Figure D.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

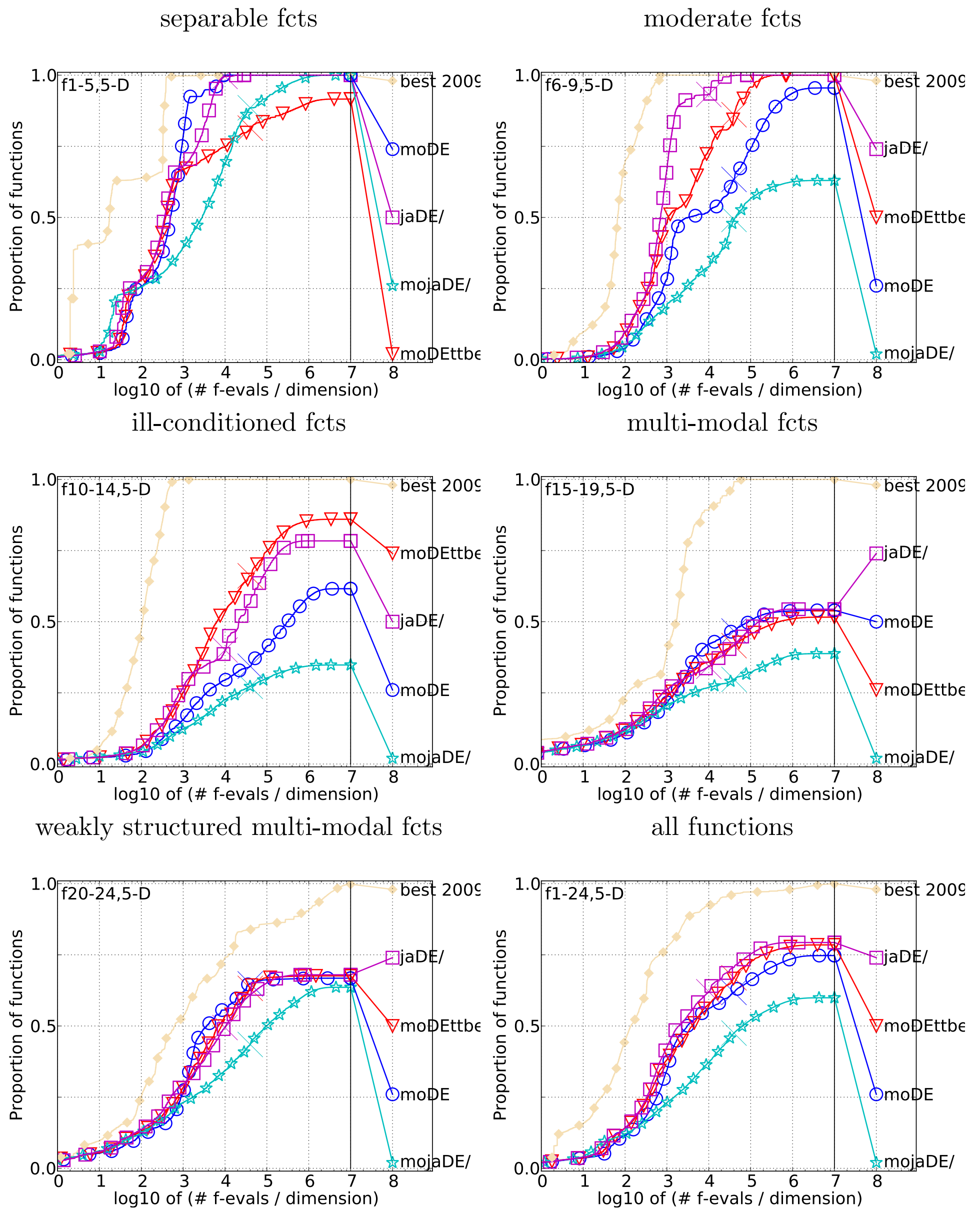

Figure D.3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

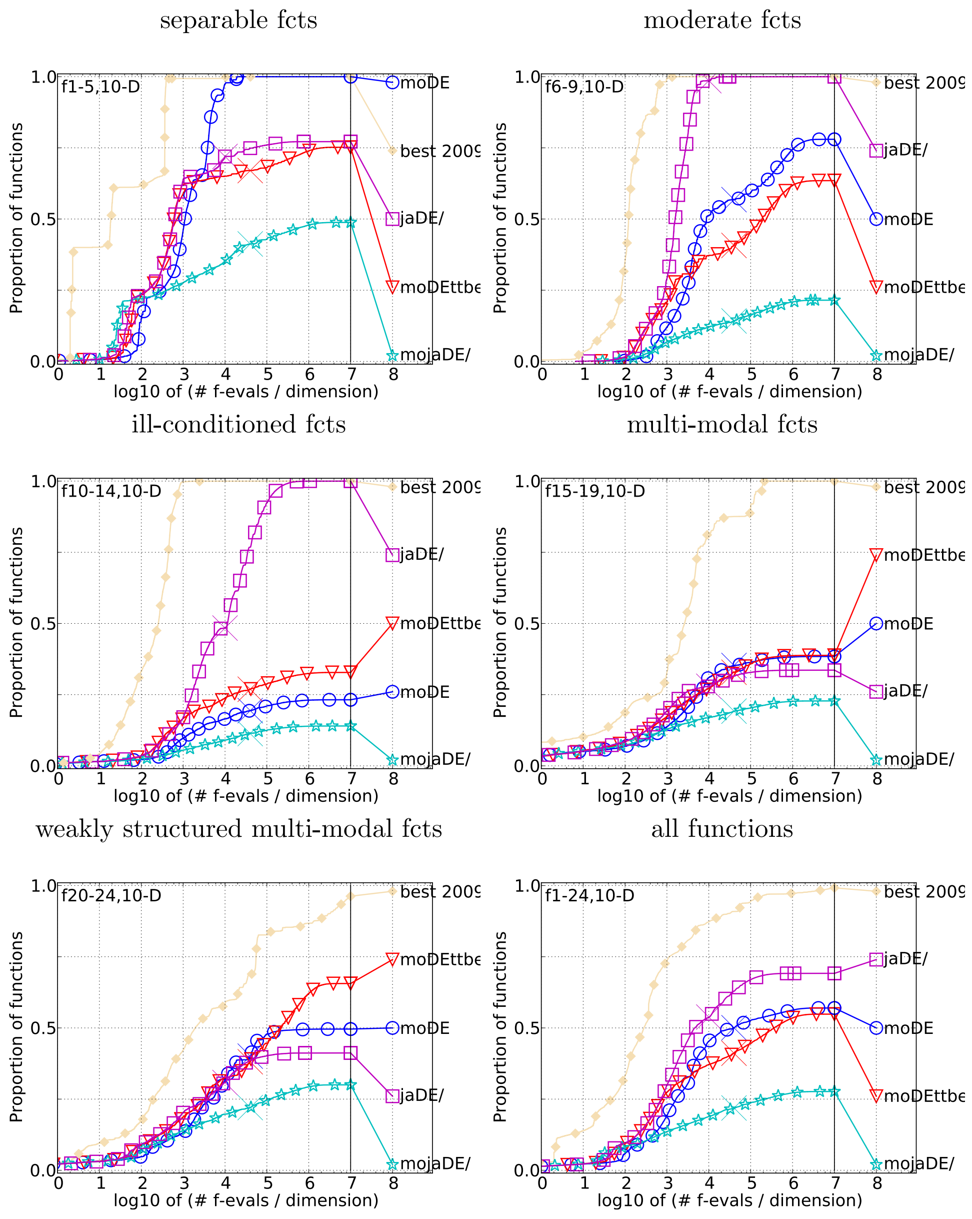

Figure D.4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

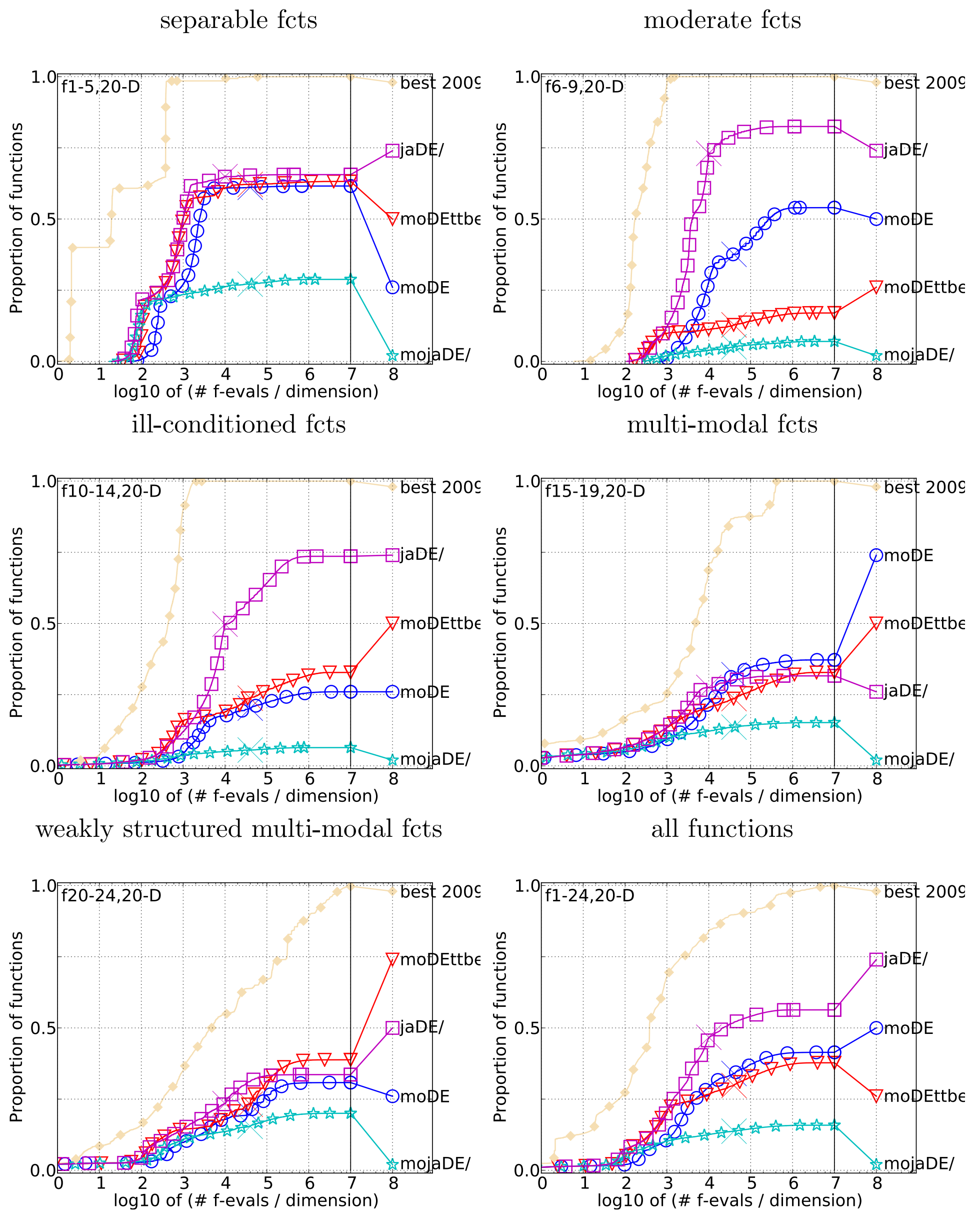

Figure D.5: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

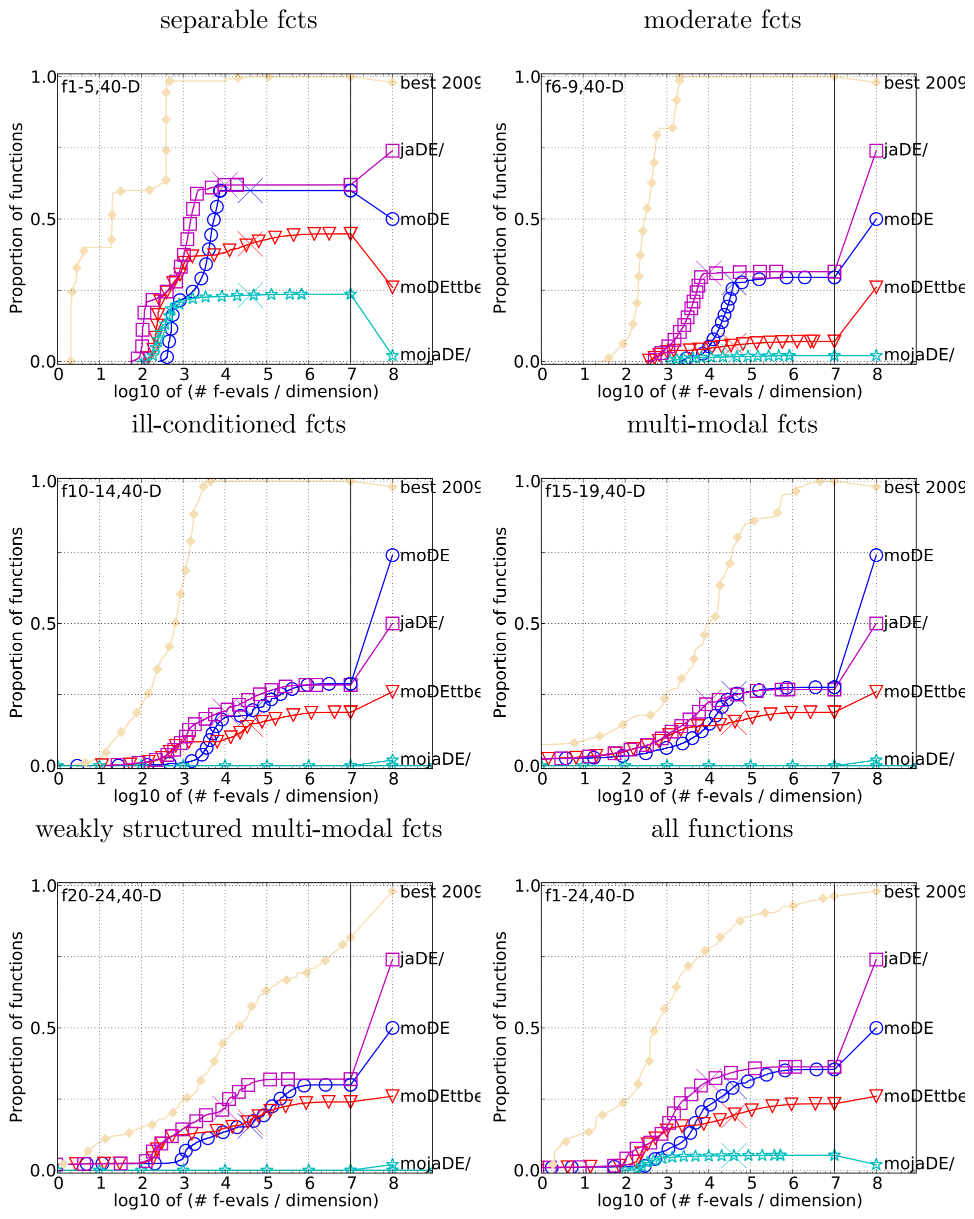

Figure D.6: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

Appendix E

Comparison Graphs of Differential Evolution and Evolutionary Strategies

Here, the performance graphs of the Canonical DE, jaDE, as well as DR2, (1+1) Cholesky CMA-ES and (1+1) Active CMA-ES are shown. This Appendix compares the algorithms that have performed best so far, to see whether DE ,or a variation thereof, performs better than a state of the art Evolution Strategy. The performance graphs show the success rate of a specific strategy, measured over a group of functions, the type being labeled on top of each graph. The horizontal axis shows the number of function evaluations divided by the number of dimensions, in log scale. The verical axis displays the proportion of functions that has converged. The bottom right graph shows overall performance, over all 24 BBoB function. Finally, beneath each group of graphs, the meaning of the labels behind each performance line on each graph is explained.
For most comparisons and conclusions, the performance of all functions for $D = 20$ and $D = 40$ were used, to assess overall performance, where the performance for $D = 40$ weighed more strongly than the performance in $D = 20$.

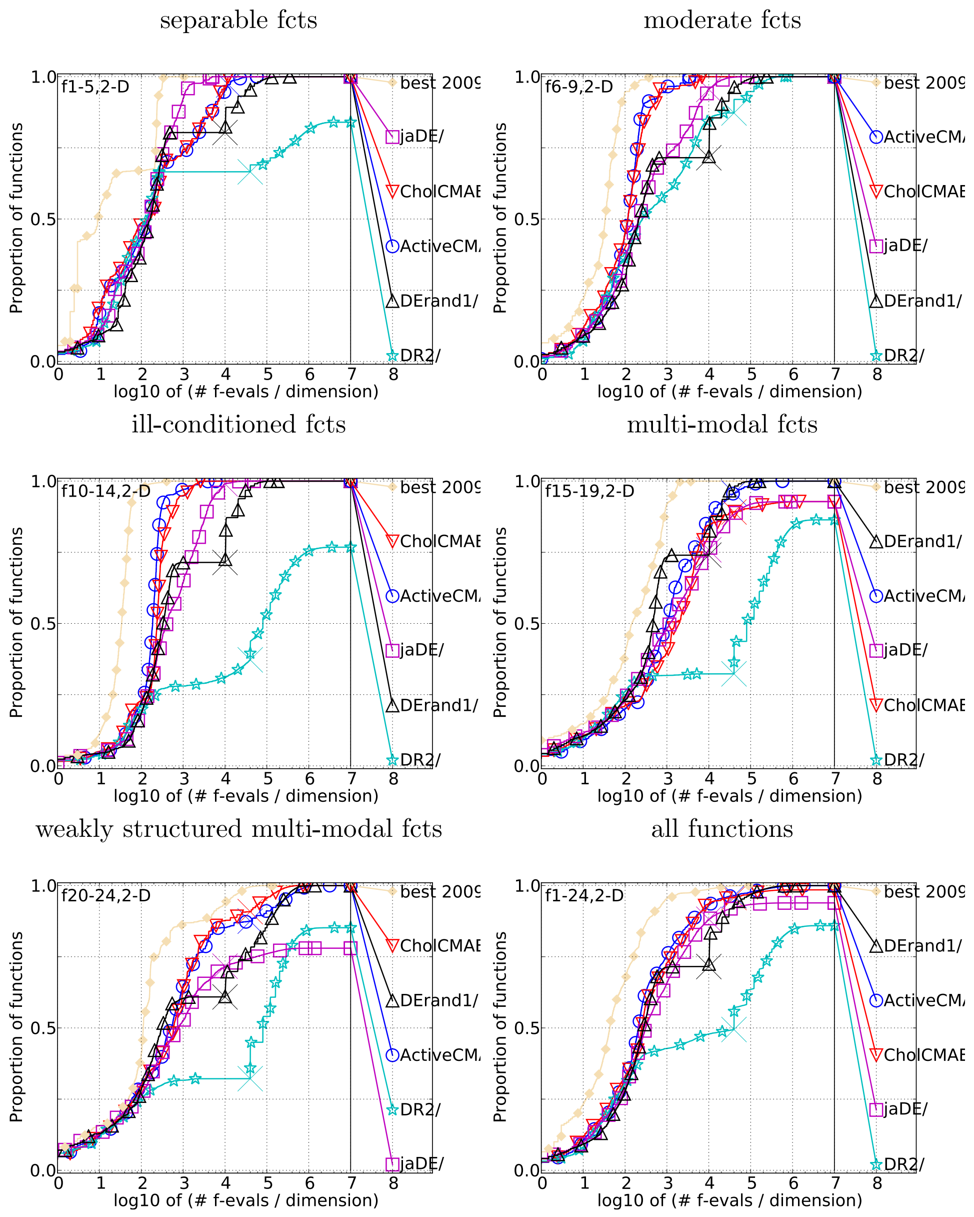

Figure E.1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

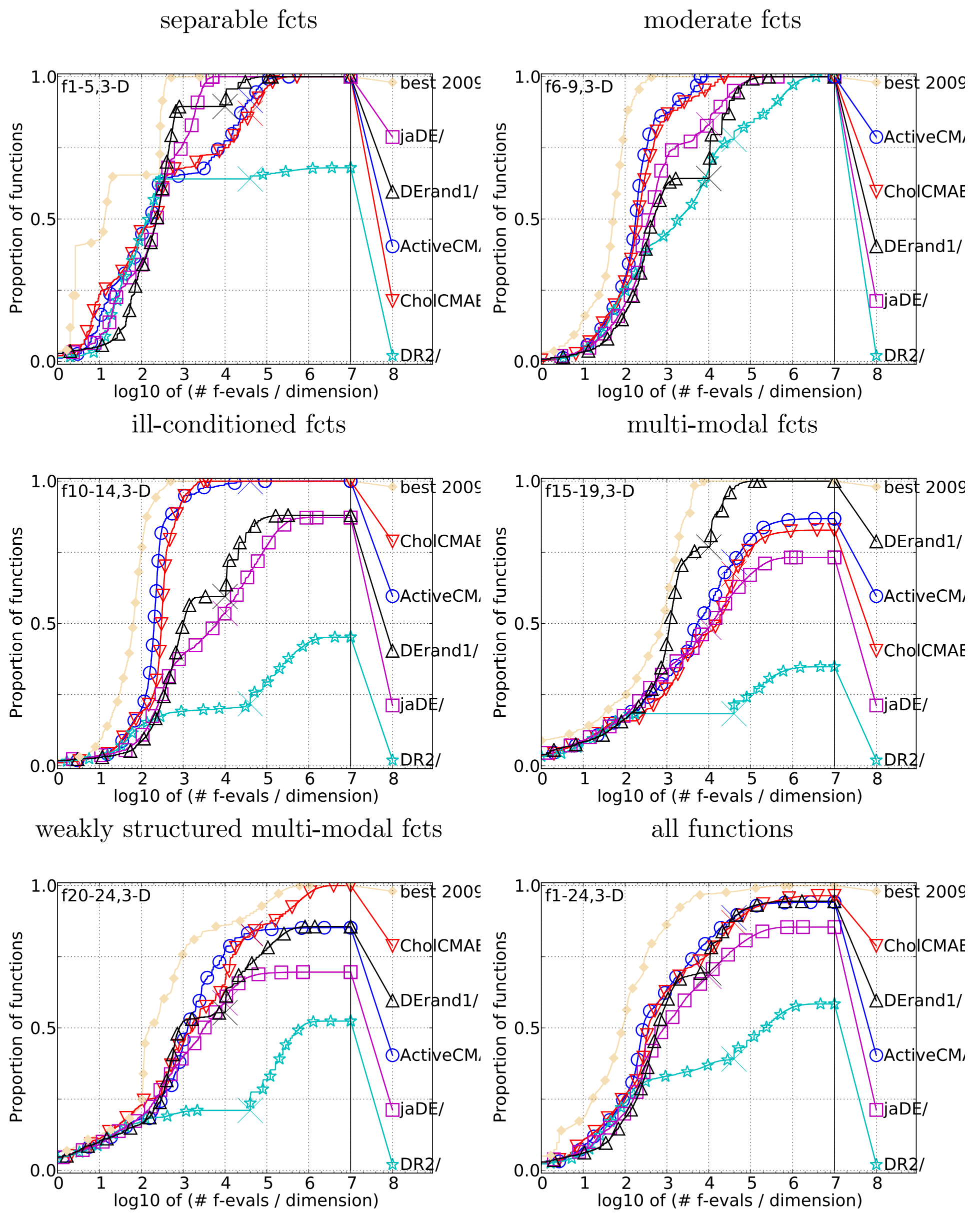

Figure E.2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

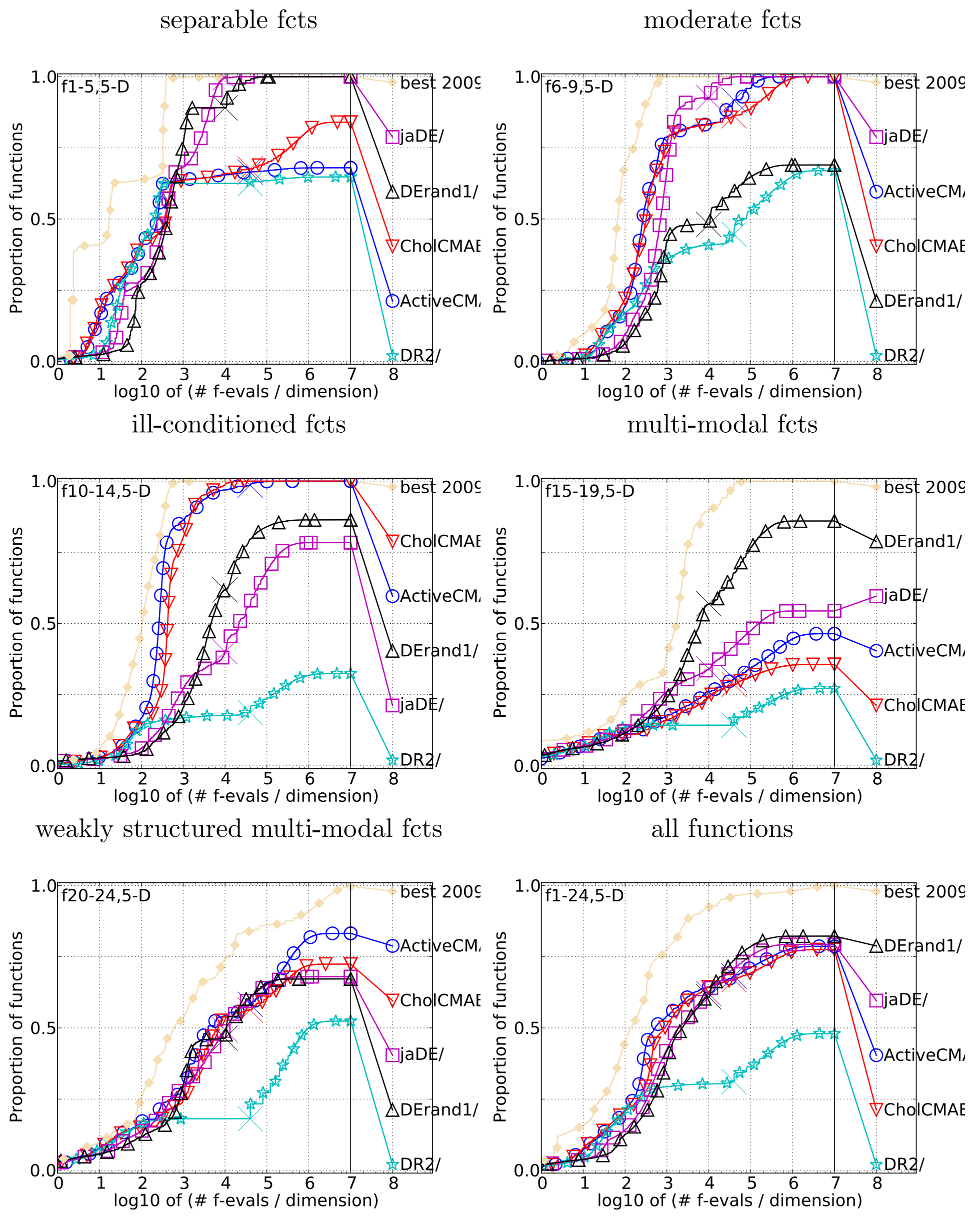

Figure E.3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

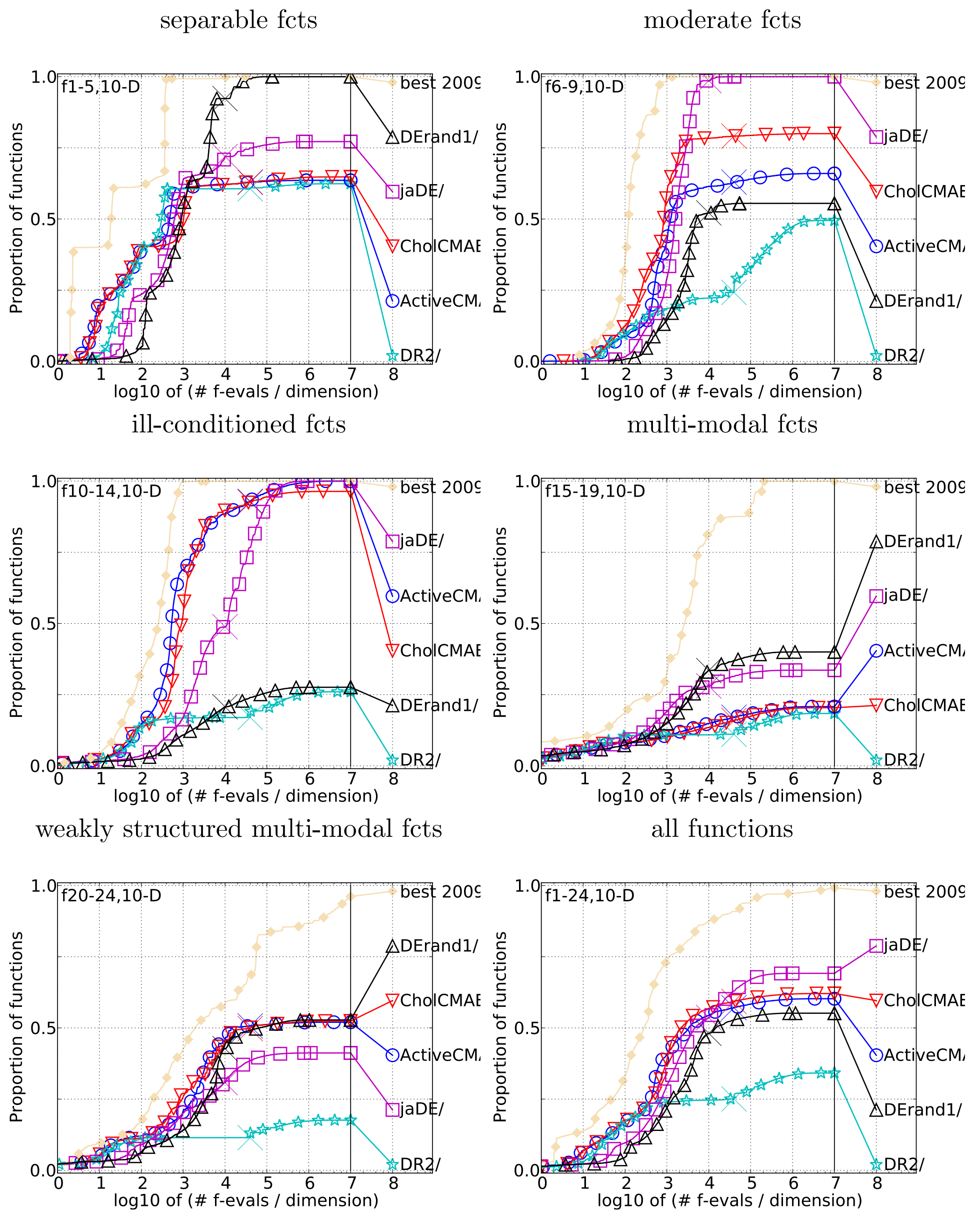

Figure E.4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

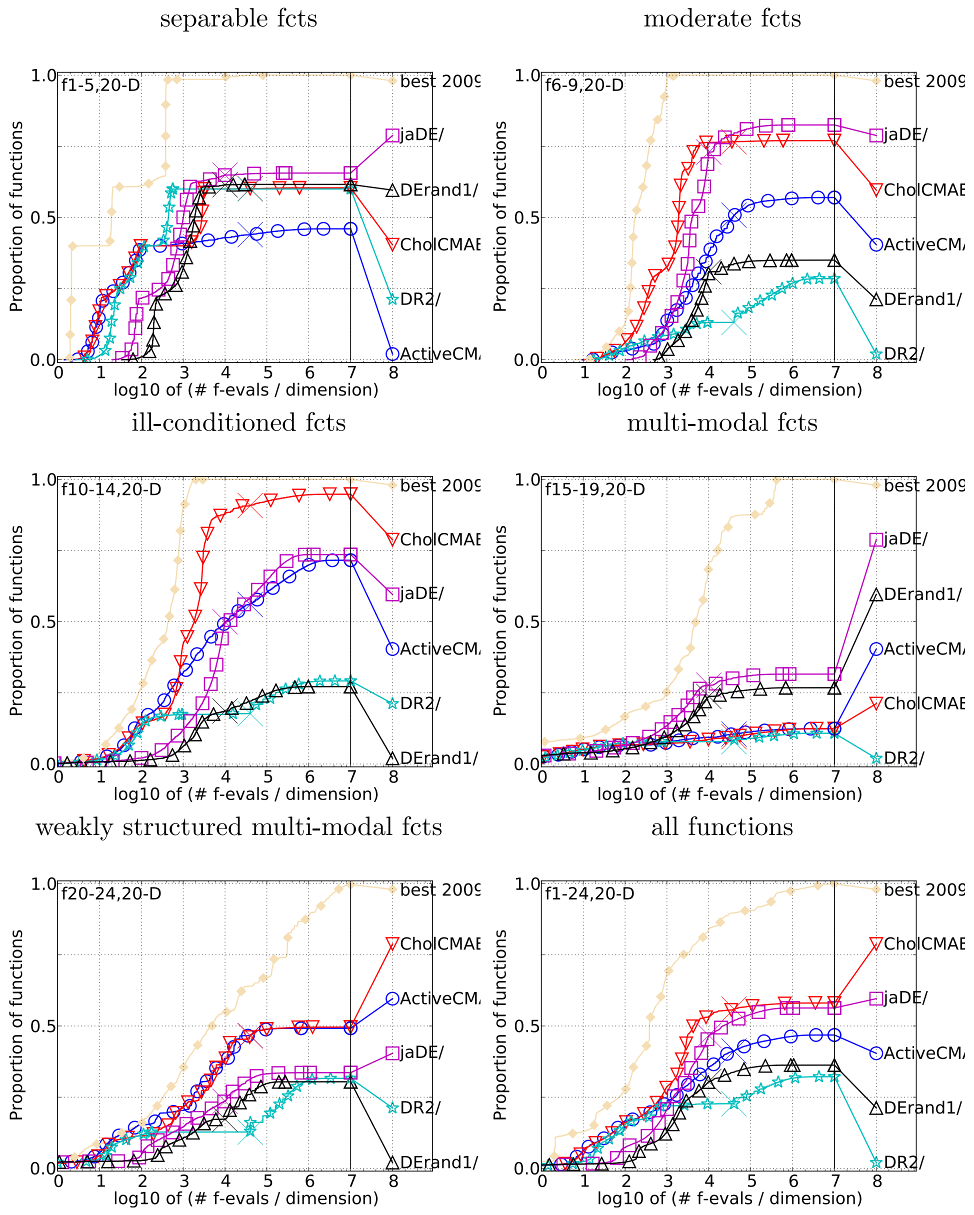

Figure E.5: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The "best 2009" line corresponds to the best observed during BBOB 2009 for each single target.

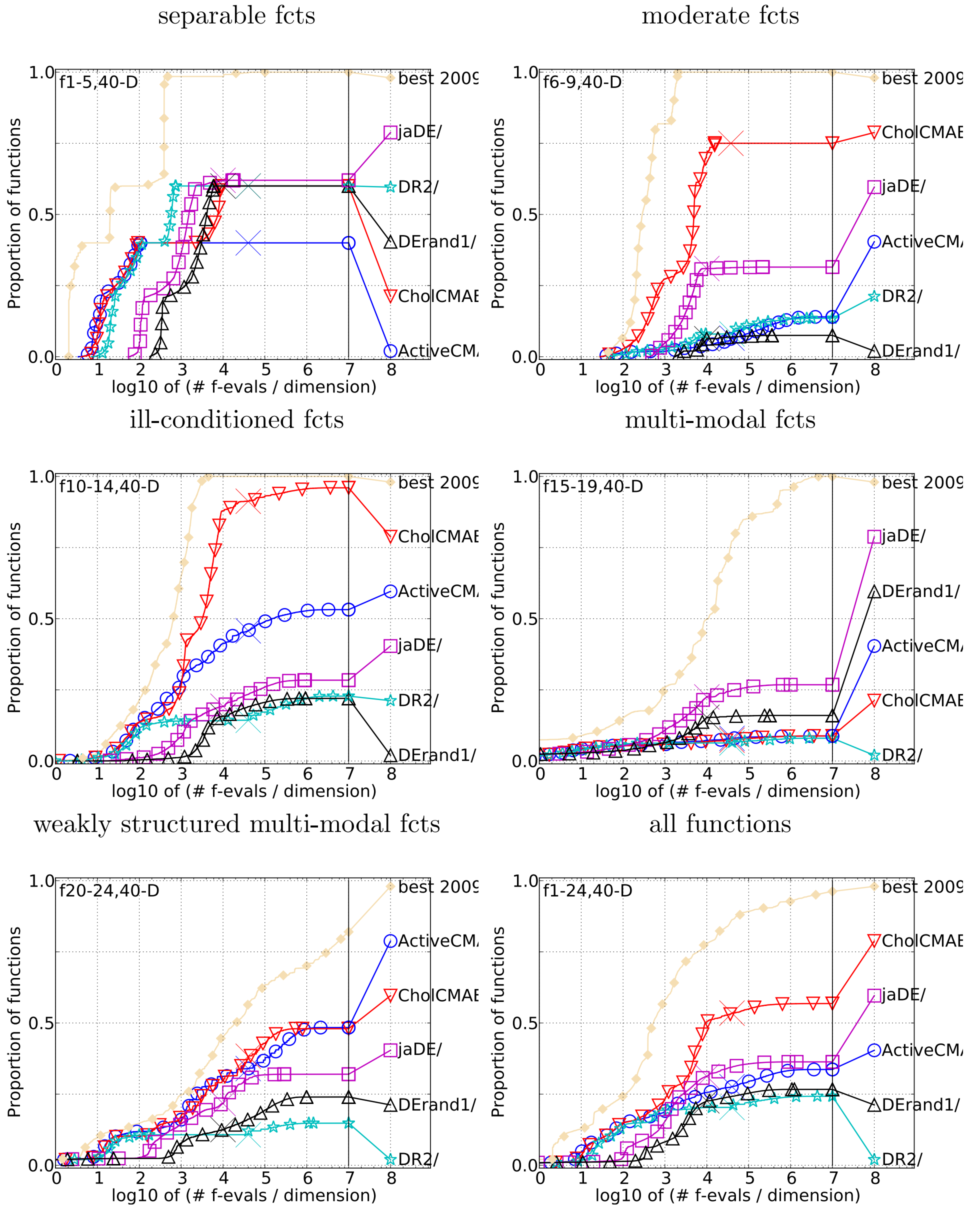

Figure E.6: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 2-D. The “best 2009” line corresponds to the best observed during BBOB 2009 for each single target.

Appendix F

LSBO performance of moDE and jaDE

Comparison graphs of moDE and jaDE in LSGO, the Large Scale Global Optimization benchmark. This benchmark is different from BBOB, as all functions have a dimensionality around 1000. Note the absence of Evolution Strategy algorithms; these algorithms use a matrix, and the space complexity rises to $O(D^2)$, which causes an unacceptable amount of overhead. MoDE, the best performing novel algorithm, is compared to jaDE, the best performing Differential Evolution variant.
Here, the average fitness value, displayed on the vertical axis, can be seen for each point at runtime, as displayed on the horizontal axis. The first 15 graphs display the average performance of moDE, the next 15 graphs display the average performance of jaDE. From the results, it is clear that jaDE performs better; moDE does not always converge, and jaDE does. Also, jaDE finds better values for each function than moDE.

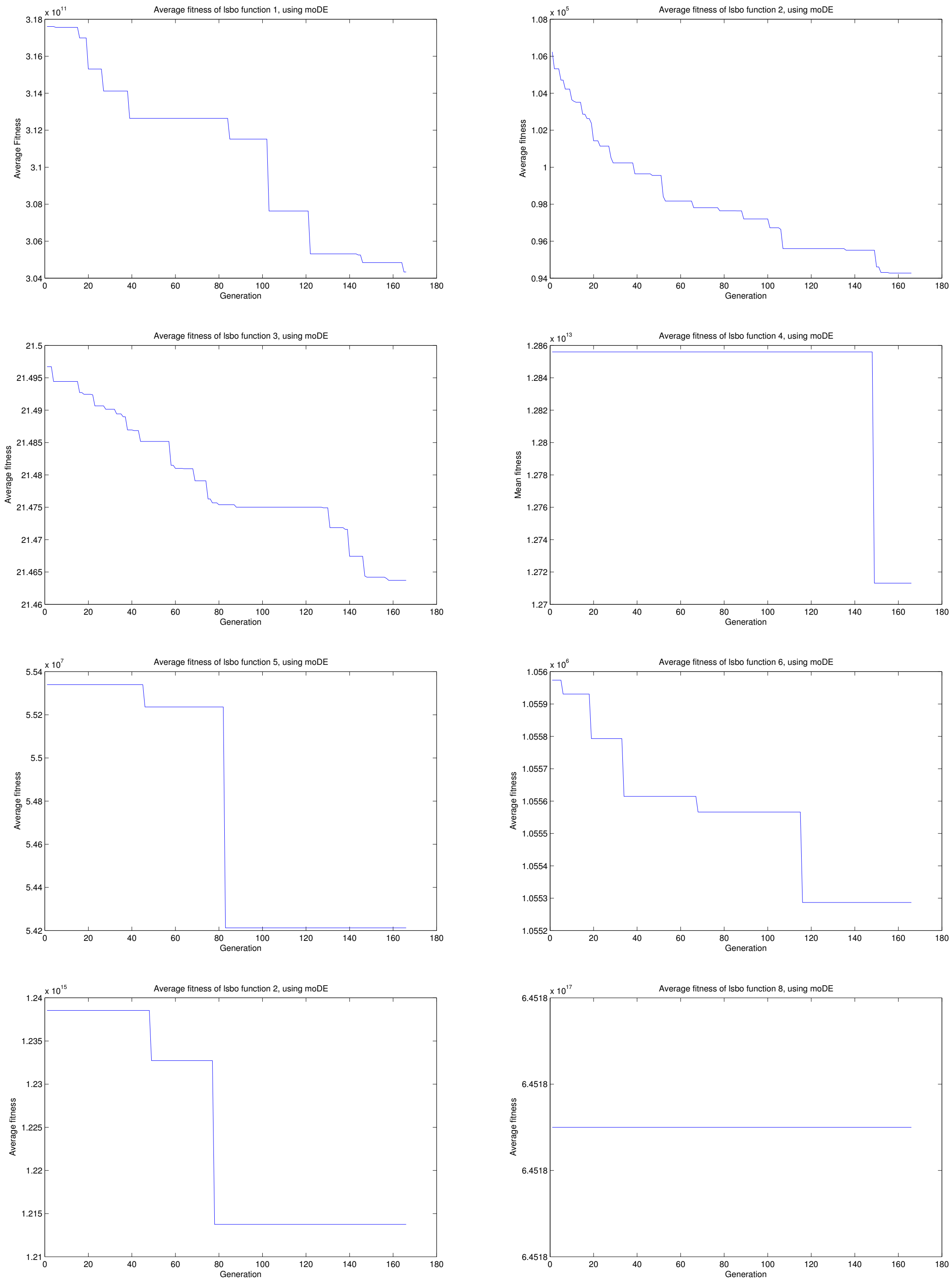

Figure F.1: The average fitness of moDE at each generation, for the LSGO functions 1-8.

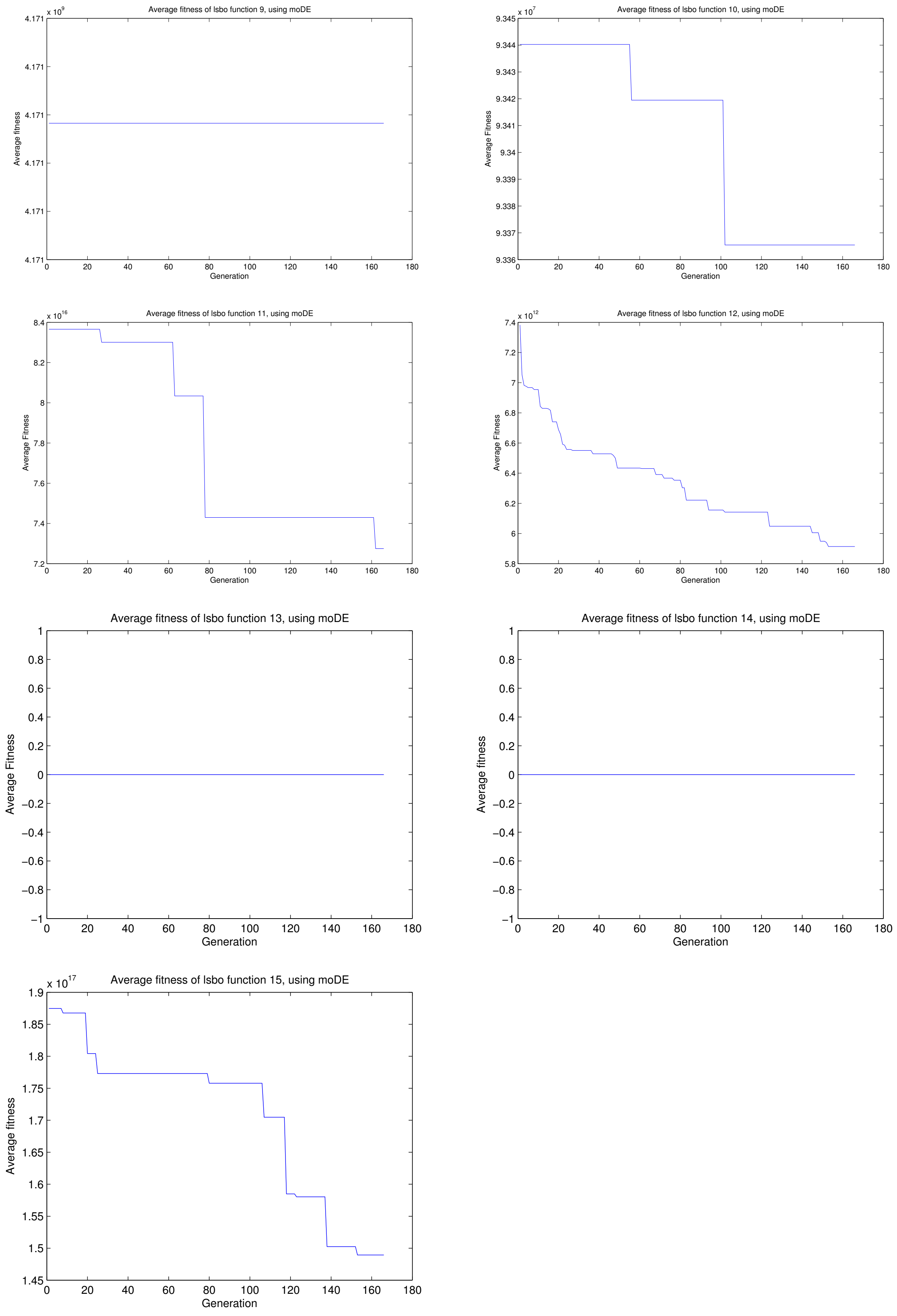

Figure F.2: The average fitness of moDE at each generation, for the LSGO functions 9-15.

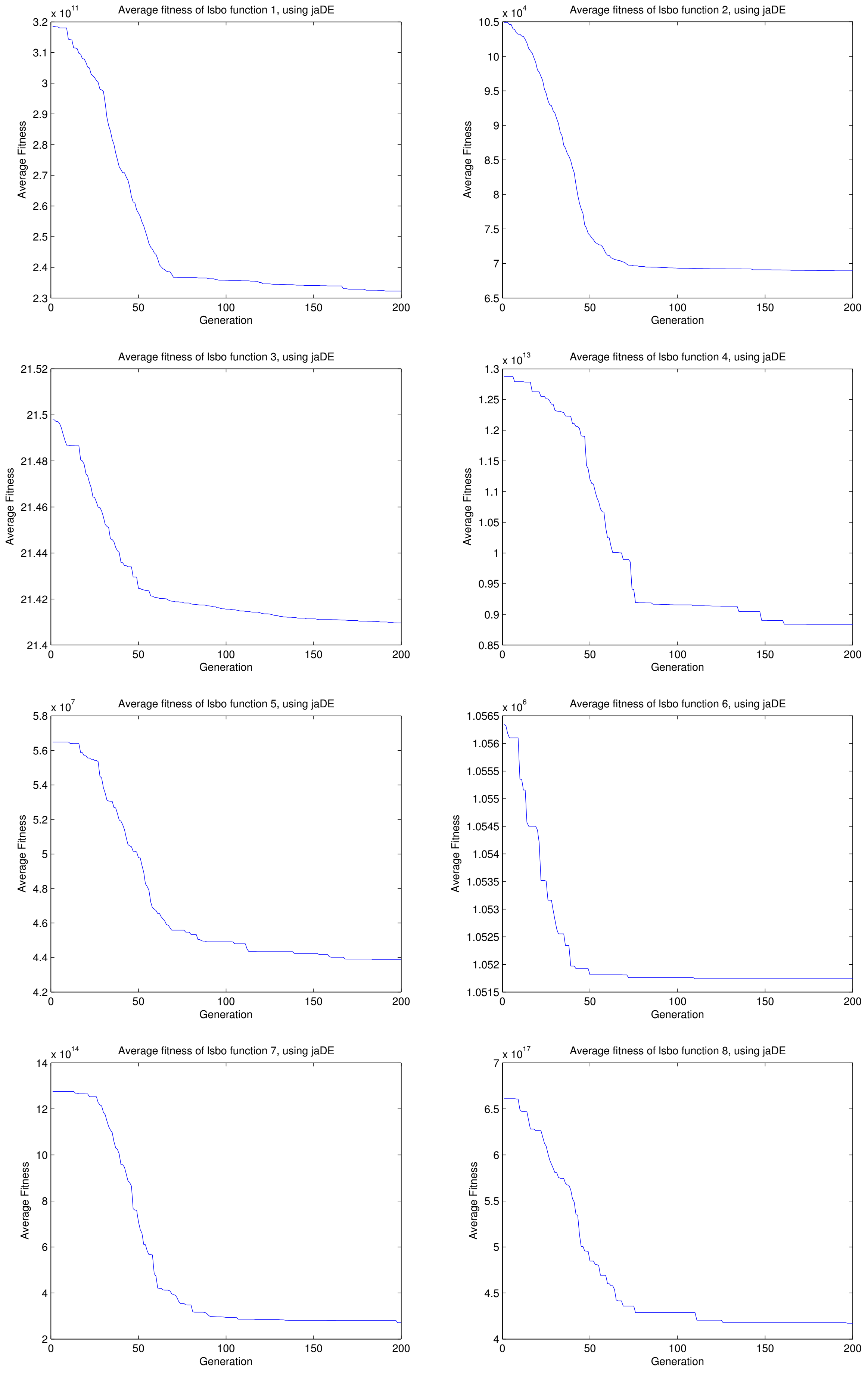

Figure F.3: The average fitness of jaDE at each generation, for the LSGO functions 1-8.

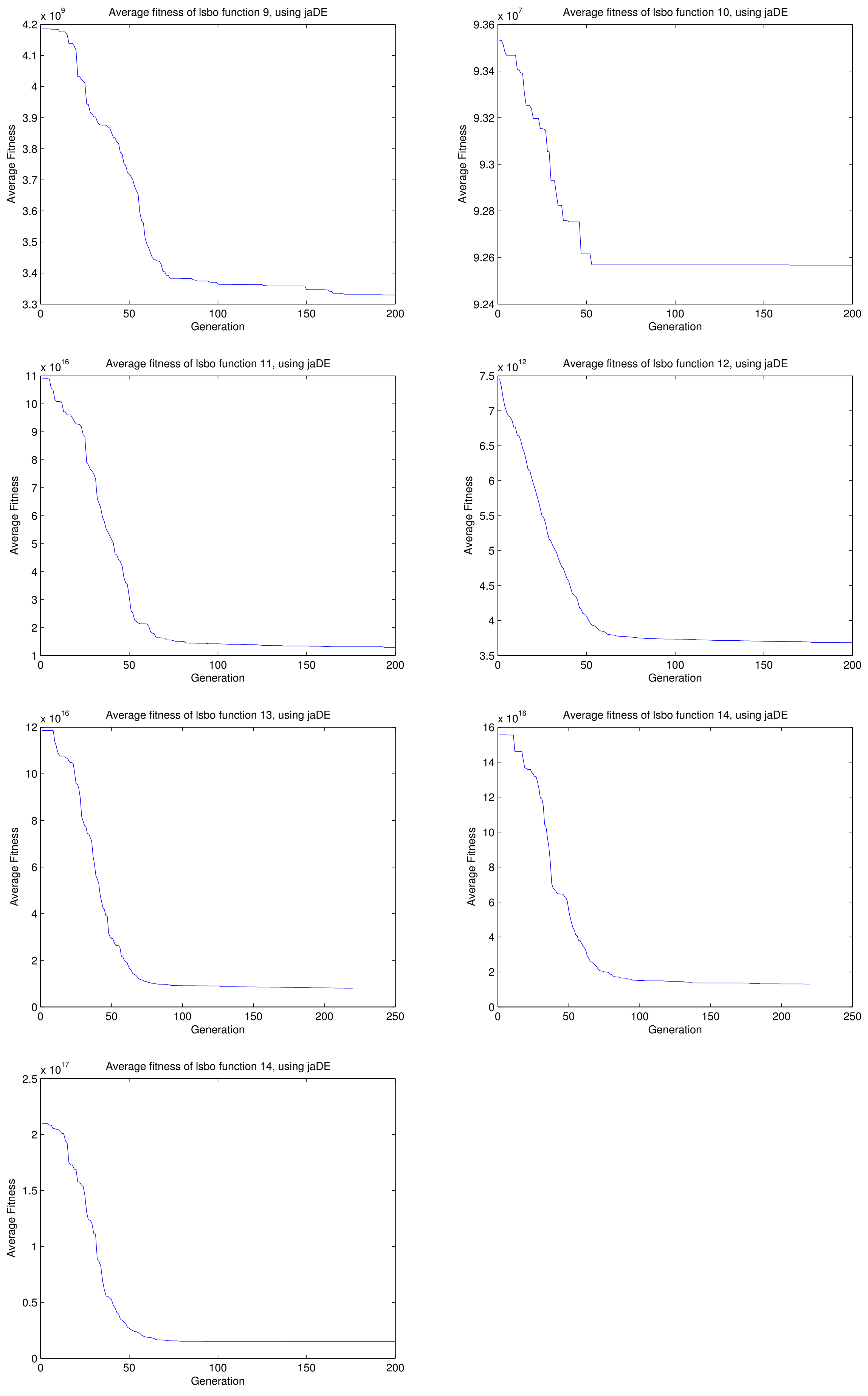

Figure F.4: The average fitness of jaDE at each generation, for the LSGO functions 9-15.

Bibliography

[1] Black box optimization benchmark. http://coco.gforge.inria.fr/doku.php?id=start. Accessed: 2013-09-11.

[2] Dirk V Arnold and Nikolaus Hansen. Active covariance matrix adaptation for the (1+ 1)-cma-es. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 385–392. ACM, 2010.

[3] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

[4] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[5] Oxford Math Center. Dot products, norms, and angles between vectors. http://www.oxfordmathcenter.com/drupal7/node/168. Accessed: 2013-06-27.

[6] Carlos Artemio Coello Coello and Gary B Lamont. *Applications of multi-objective evolutionary algorithms*, volume 1. World Scientific Publishing Company, 2004.

[7] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, 2011.

[8] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.

[9] Christian Igel, Thorsten Suttorp, and Nikolaus Hansen. A computational efficient covariance matrix update and a (1+ 1)-cma for evolution strategies. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 453–460. ACM, 2006.

[10] James Kennedy, Russell Eberhart, et al. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

[11] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin, and Hefei China. Benchmark functions for the cec2013 special session and competition on large-scale global optimization. *gene*, 7:33, 2013.

[12] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. Step-size adaptation based on non-local use of selection information. In *Parallel Problem Solving from NaturePPSN III*, pages 189–198. Springer, 1994.

[13] Petr Pošík and Václav Klema. Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 197–204. ACM, 2012.

[14] Ingo Rechenberg. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

[15] Terence Soule, editor. *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, New York, NY, USA, 2012. ACM. 910122.

[16] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[17] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, 2009.