Introduction to Optimization

Group project

Final report

$(\mu/\mu, \lambda)$-ES with Search Path

Authors:
Théo Cornille
Mirwaisse Djanbaz
Cédric Guerard des Lauriers
Luc Gibaud
Aurélien Mascaro

Professor:
Mr. Dimo Brockhoff

October 24, 2018

Abstract

The aim of this project was to contribute to the numerical benchmarking of a black-box optimization algorithm, namely the $(\mu/\mu, \lambda)$-ES with Search Path[2], via the COCO platform. This algorithm is an evolution strategy used to solve single objective optimization problems in continuous search spaces. After implementing it in Python, we used COCO to run it against a noiseless suite of well-understood test functions and thus obtained a thorough benchmarking. The algorithm performances on the different functions are automatically plotted on graphs, as well as comparisons with other algorithms. We can then extrapolate the performances of our strategy on functions outside the test suite.

1 Introduction

In this report, we will first introduce the motivation for our work, then we will present the $(\mu/\mu, \lambda)$-ES with Search Path algorithm. Its implementation in Python will be discussed, including the parameters and the stopping criteria. The results of the benchmarking provided by the COCO platform will be then analyzed, and compared with those of other algorithms, BIPOP-CMA-ES and BFGS.

1.1 Black-box optimization

In practice, many optimization problems cannot be solved analytically. Black-box methods are then a collection of methods that optimize a problem without requiring the gradient (that's why they are also called "derivative-free methods").
Thus, they can be applied to problems that aren't differentiable, or problems for which the derivative is in practice hard to compute (e.g. expensive numerical simulations). The search for the optimum is done only through function evaluations, so the problem class is fairly general.

Figure 1: Black box optimization[1]

In consequence, many algorithms can be applied to black-box problem and the difficulty lies in choosing the best algorithm for a particular problem.
Indeed, most of the time, the search space is too large to be visited completely, and many algorithms are ill-fitted for a specific problem (slow convergence, get stuck in local optima etc).
So we need to benchmark black-box optimization algorithms, in order to see which algorithm performs best on each type of problem. With the COCO platform, we can test our implementation against a set of well-understood test functions, and then use the results to predict what will be the performances of the algorithm on other problems.

1.2 Evolutionary strategies

Evolutionary strategies are evolutionary algorithms, a subset of optimization algorithms, mainly used in the case of continuous search spaces. Created in the 1960s, evolutionary strategies find their inspiration in the biological evolution and its mechanisms such as mutation, recombination and selection.

Evolutions Strategies are driven by progress-rate with which we can obtain an estimation of the progress made after each step of evolution and deduce from the results how to adjust parameters such like recombination weights.
As in the biological evolution, an evolution strategy must include adaptation mechanisms in order to adapt to the given parameters as otherwise, this would merely be a common search algorithm based on randomness.
We can distinguish multiple approaches of evolution strategies such like the (+) ones where solutions from the previous step are added to the current step. On the other side with have the comma (,) strategies where the exploitable solutions only depend of the current step.
Our work about evolution strategies will concern the $(\mu/\mu, \lambda)$-ES with Search Path, which designate a weighted-combination based strategy, meaning that the μ parents will give a mean solution after we proceed to a weighted-recombination.

This induces the following main principles:

- We need to select all the μ best results from population P depending on fitness. Each result not satisfying the targeted fitness will be definitely removed before a new step stats.
- We have to generate a set of recombinations between each result to generate a set of newer results with an improved fitness.
- At each step, we have to consider, given the control parameters, that each result may slightly mutate randomly to provide the new individuals.

2 Description of the algorithm

2.1 Pseudo-code

Below, we will introduce the parameters as they were interpreted in our implementation :

- f(x) is the function we want to minimize in order to get the best offspring possible - the fitness value.
- x are the parameters of the function f while n is the dimension associated.
- λ is the number of offsprings and μ the number of parents, knowing the fact of that f will only keep the best offspring population that will compose our μ.
- Indeed, some individuals might be member of parents and offspring in the same time.

Algorithm The $(\mu/\mu, \lambda)$-ES with Search Path

0 **given** $n \in \mathbb{N}_+$, $\lambda \in \mathbb{N}$, $\mu \approx \lambda/4 \in \mathbb{N}$, $c_\sigma \approx \sqrt{\mu/(n+\mu)}$, $d \approx 1 + \sqrt{\mu/n}$, $d_\mathrm{i} \approx 3n$
1 **initialize** $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\sigma} \in \mathbb{R}^n_+$, $\boldsymbol{s}_\sigma = \mathbf{0}$
2 **while** not happy
3 **for** $k \in \{1, \ldots, \lambda\}$
4 $\boldsymbol{z}_k = \mathcal{N}(\mathbf{0}, \mathbf{I})$ // i.i.d. for each k
5 $\boldsymbol{x}_k = \boldsymbol{x} + \boldsymbol{\sigma} \circ \boldsymbol{z}_k$
6 $\mathcal{P} \leftarrow \mathsf{sel\_}\mu\mathsf{\_best}(\{(\boldsymbol{x}_k, \boldsymbol{z}_k, f(\boldsymbol{x}_k)) \,|\, 1 \le k \le \lambda\})$
// recombination and parent update
7 $\boldsymbol{s}_\sigma \leftarrow (1 - c_\sigma)\, \boldsymbol{s}_\sigma +$
7b $\sqrt{c_\sigma\,(2 - c_\sigma)}\,\frac{\sqrt{\mu}}{\mu} \sum_{\boldsymbol{z}_k \in \mathcal{P}} \boldsymbol{z}_k$
8 $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \circ \exp^{1/d_\mathrm{i}} \left(\frac{|\boldsymbol{s}_\sigma|}{\mathbb{E}|\mathcal{N}(0,1)|} - \mathbf{1} \right)$
8b $\times \exp^{c_\sigma/d} \left(\frac{\|\boldsymbol{s}_\sigma\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1 \right)$
9 $\boldsymbol{x} = \frac{1}{\mu} \sum_{\boldsymbol{x}_k \in \mathcal{P}} \boldsymbol{x}_k$

- S_σ is our search path.
- x_k represents the offsprings while z_k are the mutation steps and σ the mutation vectors.

Operating mode:
As we have seen in the Evolutionary strategies section, the algorithm is governed by three main principles which are the representations, from first to last, of:

- Environmental Selection.
- Recombination and mating selection.
- Parameters control and mutations.

Thus, knowing this algorithm also is defined by a search path, we need the criterias to be initialized.
In order to understand how the algorithm has been implemented and why, we will at first focus on the global objectives of the main lines from the pseudocode.
This first step is the one who will define the input parameters, among those who will lead the upcoming mutations.
At first, we need an initialization of the parental centroid (x, s), the beginning search point and to initialize S_σ to 0.
This step is done at line 1.
At line 2 we enter the while loop, allowing each step to take place and to make evolve our population P to the next generation.
The for loop at line 3 is dedicated to the mutation procedure. Looking at this loop, we see that this procedure only involves the step-size σ.
The next lines are the dedicated to update the control parameters.

As the model evolves, the control parameters also have to change in order to consider the new state of the population P at the current step.
The algorithm continues the while loop till it reaches the stopping criteria or it ends up failing.

Now we have an overview of the main ideas of the lines coming from the pseudocode, we can look in detail how a step works in a more detailed way.
Thus, we summarize the steps executed by the algorithm as follows:

- At first, we apply the recombination and mating selection principle by choosing parents from μ in order to generate a new offspring coming from duplication and recombination of the selected μ parent(s).
- Then, the new offspring generated will be subject to the effects of the mutation defined by the given parameters. After this mutation, the new offspring will finally become a member of the population.
- At last, the population undergoes environmental selection that reduces the total population to the original size.

3 Description of the implementation

The algorithm has been implemented in Python, with the use of some libraries such as NumPy, math or functools.

3.1 Initialization

The parameters of our implementation are all initialized from x and λ. Indeed, the size of our research space n is given by the length of the initialization vector x. Moreover, only n, x and λ are needed to compute μ, c_σ, d, d_i, σ and s_σ.

```
def init(x, lambda_):

    n=len(x)
    mu = int(lambda_/4)
    c_sigma = math.sqrt(float(mu)/(float(n)+float(mu)))
    d = 1 + math.sqrt(mu/n)
    di = 3*n

    sigma = np.ones(n)
    s_sigma = np.zeros(n)

    return n, mu, c_sigma, d, di, x, sigma, s_sigma
```

In addition to all the parameters of our algorithm, the COCO maximum iteration to process, or budget, needs to be defined.

3.2 Algorithm

As follow, the creation of offsprings is easily implemented using numpy.random.normal(0, 1, n).

```
# Creation of offsprings
Zk = []
Xk = []
L = []
for k in range(lambda_):
    Zk.append(np.random.normal(0, 1, n))
    Xk.append(x + sigma*Zk[k])
    L.append((Zk[k],Xk[k]))
```

In order to select the mu best (x_k, z_k), we implemented the following function which takes as argument L, the list of (x_k, z_k) and the function to minimize f, and return the list (x_k, z_k) sorted from the best x_k to minimize f to the worst.

```
def sorting(L,f):
    H = list(map(lambda args : (args[0],args[1],f(args[1])),L))
    H = sorted(H, key=lambda tuple_: tuple_[2])
    H = list(map(lambda args : (args[0],args[1]),H))
    return H
```

The previous function allows us to associate with P the list of the mu best (x_k, z_k). The sums of the z_k of P and of the x_k of P are the easily defined.

```
# Choice of offsprings
P = sorting(L,f)[:mu]

Zk_best = list(map(lambda args : args[0],P))
Zk_best_sum = functools.reduce(lambda a,b : a+b, np.array(Zk_best))

Xk_best = list(map(lambda args : args[1],P))
Xk_best_sum = functools.reduce(lambda a,b : a+b, np.array(Xk_best))
```

In order to update σ, we need to compute:

- $\mathbb{E}|\mathcal{N}(0,1)|$, which is equal to $\sqrt{2/\pi}$
- $\mathbb{E}||\mathcal{N}(0,\mathbf{I})||$ which is equal to $\sqrt{n}$

s_σ, σ and x can then be updated as follow.

4 Results

4.1 Our algorithm

4.2 Comparison with BIPOP-CMA-ES

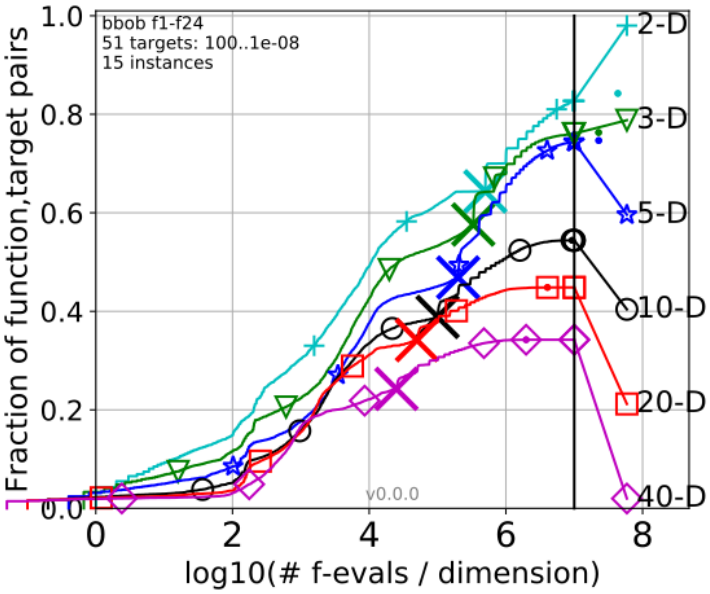
CMA-ES stands for *covariance matrix adaptation evolution strategy*. As we said previously, **Evolution strategies** (ES) are **stochastic, derivative-free methods** for **numerical optimization** of **non-linear** or **non-convex continuous optimization** problems.

CMA-ES and $(\mu/\mu, \lambda)$-ES with Search Path belongs to this class of algorithms.

```
# Updating s_sigma
s_sigma = (1-c_sigma)*s_sigma+math.sqrt(c_sigma*(2-c_sigma))*(math.sqrt(mu)/mu)*Zk_best_sum

# Updating sigma
esp_normale_1D = math.sqrt(2/math.pi)
esp_normale_I = math.sqrt(n)
sigma = sigma\
        *(np.exp((np.abs(s_sigma)/esp_normale_1D)-1)**(1/di))\
        *np.exp((np.sqrt(np.array(list(map(lambda a : a**2,s_sigma))).sum())/esp_normale_I)-1)

# Updating of x
x = Xk_best_sum/mu
```



However, CMA-ES differs from $(\mu/\mu, \lambda)$-ES with Search Path as it adapts three parameters: mean, step size and covariance matrix allowing to solve highly non-separable problems. Indeed, it represents the pairwise dependencies between the variables in the multivariate normal distributions during the sampling of the new population.
The covariance matrix adaptation method updates thus the **covariance matrix** of this distribution and conducts an iterated **principal components analysis** of successful search steps to find a new geometrical representation that increases the likelihood of previously successful search steps.
CMA-ES is thus particularly useful if the function f is ill-conditioned.

BIPOP CMA-ES stands for Bi Population CMA-ES. It corresponds to $(\mu/\mu_w, \lambda)$-CMA-ES algorithm applied in a multistart strategy: 2 start regimes are used depending on the budget during function evaluations and multiple restarts are conducted adapting the population size λ : diminishing it or increasing it.

BIPOP-CMA-ES algorithm is the winner of the 2009 black-box optimization benchmarking competition (BBOB). We can see on the picture below taken from the course that the area of the ECDF curve of BIPOP CMA-ES algorithm is the smallest: thus it has the lowest average log run time.
We clearly see that over all functions in any dimension, BIPOP CMA-ES outperforms $(\mu/\mu, \lambda)$-ES with Search Path algorithm. More than 80% of the functions in any

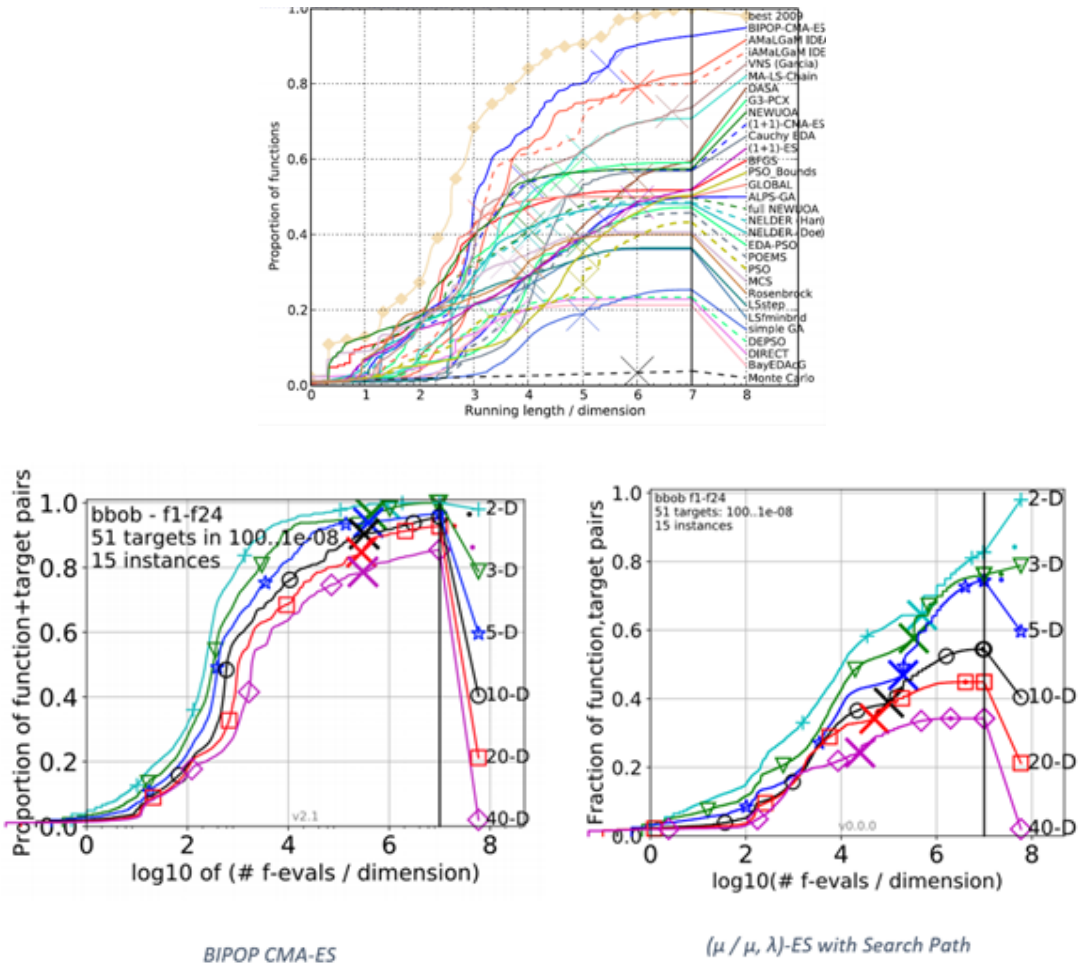


Figure 2: Runtime distributions (ECDFs) over all targets

dimension reach the target 1e-08 for BIPOP CMA-ES for a budget between 3000 and 6000 function evaluations while CMA-ES have not above 30% under 6000 functions eval in any dimension.
Moreover when comparing algorithms on the type of the noiseless BBOB functions, $(\mu/\mu, \lambda)$-ES with Search Path algorithm works well for multi modal functions as it has no difficulty to find an local optima due to the shape of the function while it is harder for ill-conditioned functions because this algorithm has no space adaptation (covariance matrix adaptation).

Conclusion

The BIPOP-CMAES algorithm improves the performance of CMA-ES by giving a better restart strategy that will repeatedly restart the algorithm with varying population size and parameters preventing itself to be blocked in local optima for many functions as CMA-ES can be. And the covariance matrix is really helpful to optimize complex functions as ill-conditionned functions and multi-modal functions as it doesn't get stuck in a local optima.
However, this algorithm can be considerably outperformed (by others algorithms) at least on functions that are smooth, "regular" and only moderately ill-conditioned (f1, f5, f8, f9), on separable functions (in particular f3 and f4) and on the multimodal

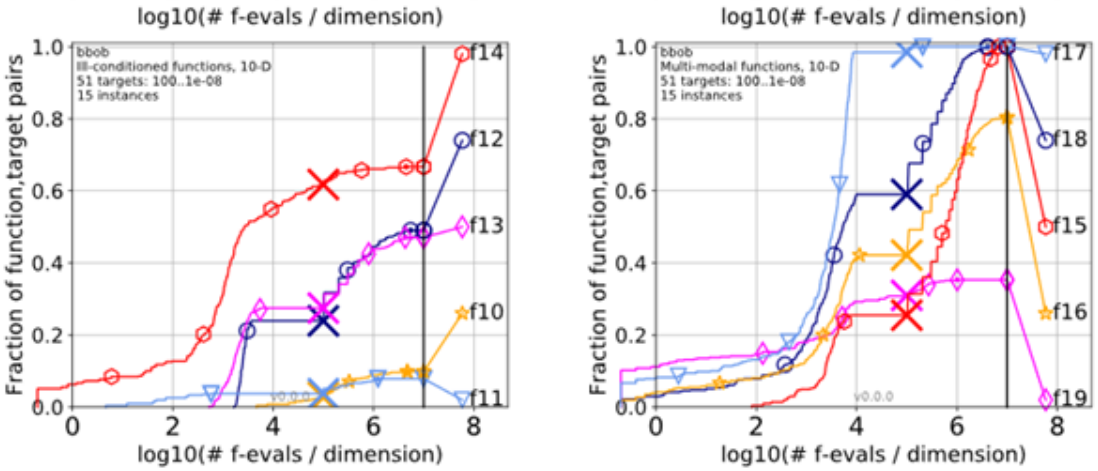


Figure 3: Runtime distributions (ECDFs) summary per function groups, (dimension = 10)

functions f21 and f22 (see 3 below).

4.3 Comparison with BFGS

5 Conclusion

References

[1] Dimo Brockhoff. Introduction to optimization, september 2018. http://www.cmapx.polytechnique.fr/ dimo.brockhoff/optimizationSaclay/2018/slides/01-intro.pdf.

[2] Dirk V. Arnold Nikolaus Hansen and Anne Auger. Evolution strategies. *Handbook of Computational Intelligence*, pages 871–898, 2015.

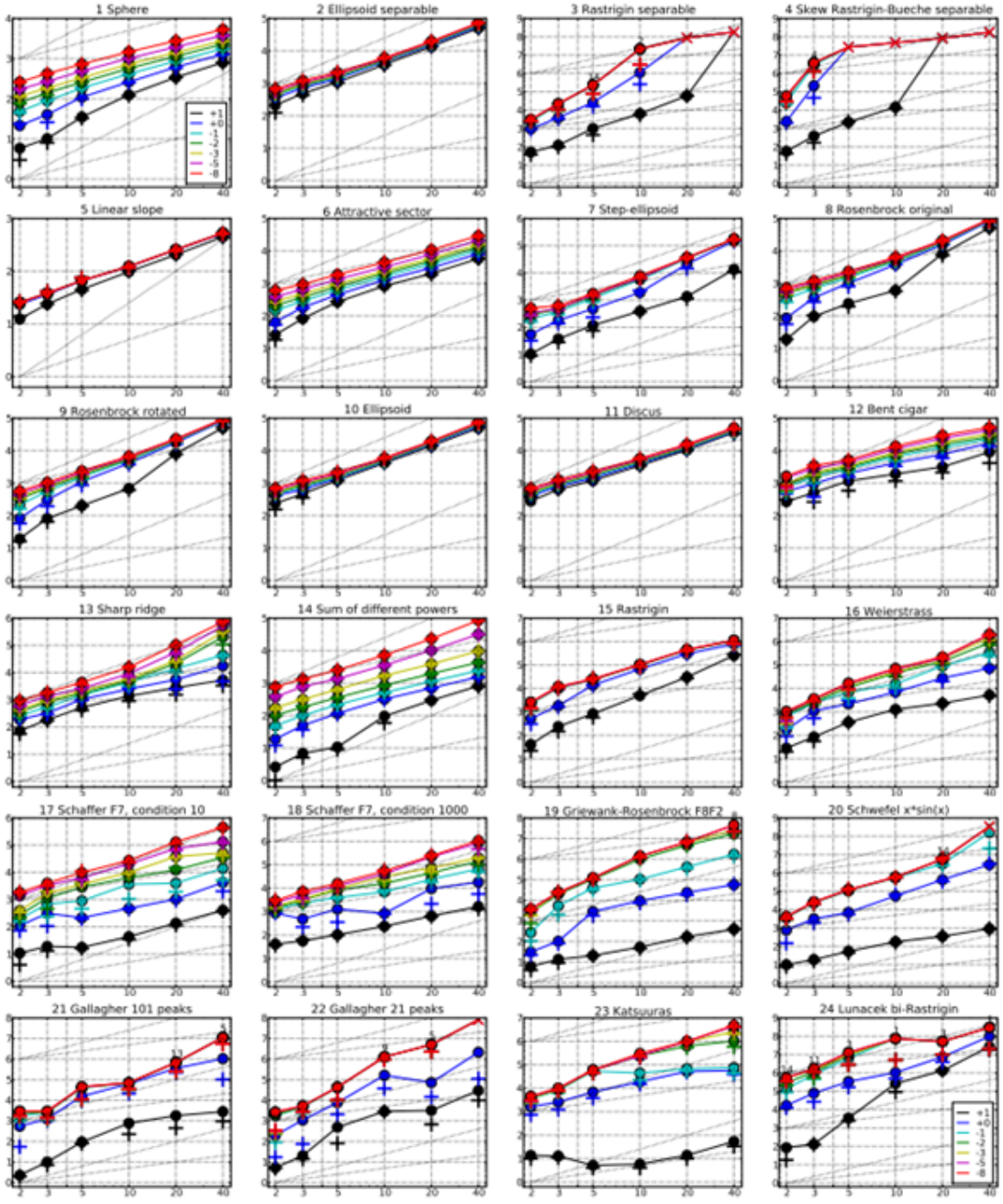


Figure 4: Average number of f-evaluations to reach target (X Axis : Dimension, Y Axis : Log10(F evals/Dim))