Evolutionary algorithms PA2 Jordy van Miltenburg (s2336553)
j.p.van.miltenburg@umail.leidenuniv.nl

1 Introduction

Evolutionary strategies a subdomain of evolutionary algorithms which is focused on the continuous numbers. Evolutionary strategies a search paradigm inspired by human evolution mostly used for continuous black-box optimization. Solving problems by creating a population, recombining them into new offspring, mutated and evaluated to be formed into the new population. Which after multiple generations will come closer to solving the given problem. In this practical assignment, I try to explore the different possibilities of building an evolutionary algorithm. Such as recombination, mutation but also the different self-adaptive algorithms.

2 Assignment

For this practical assignment, we are sent on a journey to solve 24 Black-Box Optimization Benchmarking (BBOB) minimization problems. One algorithm that is able to perform well on as many problems as possible, therefore creating a generalizable algorithm. The evaluation of the algorithm is done by analysing the algorithm in the IOH analyzer, which spits out a lot of data about your algorithm. With things such as expected runtime, it shows the best fitness score over the number of evaluations. But also an overall performance score compared to other algorithms the aggregated empirical cumulative distribution. Each problem consists of a function with a bitstring input that is to be minimized.

$$f : [-5, 5]^n \Rightarrow \mathbb{R}$$

With n = {2,5,20} (i.e., 2-dimensional,5-dimensional and 20-dimensional search space).

2.1 Selection

In each generation, a set of parents is necessary to breed a new generation. The selections made during this process are based on the fitness value of the parents. Where the fitter parents have a higher chance to be selected for the breeding of the new generation. A different selection operator is chosen for a particular problem, two of those operators are proportional selection and tournament selection. Both of the operators have a random number generator to prevent the fittest parents to be chosen and the convergence to a solution will not be as fast. By doing selection in this way a broader search space is investigated.

2.2.2 Tournament selection

The tournament selection operator is working with a random number generator but every individual has an even chance in the first stage of the operator. Tournament selection works by selection k individuals from the whole population. To give an example the population size is 10 and k equals 4, this means that 4 individuals are randomly selected from the population. Of those 4 individuals, only the best is selected for breeding. Opposed to proportional selection tournament selection has a higher chance to select worst parents for breeding, which does explore the search space more.

2.2 Recombination

In order to create offspring for the new population, a recombination operator is used. Within this operator, the selected parents—done by using the earlier mentioned selection operator—are used to create new offspring for the next generation. The genes of the parents are combined to form offspring that has a bit of both. Different recombination operators can be used within a evolutionary strategies. Discrete, intermediate and the global variants of the two.

2.2.1 Discrete

The discrete recombination operator is the same as uniform recombination used in genetic algorithms. Where the two selected parents are used to build the new offspring. Cycling through every bit with a 50/50 chance to pick the bit of the first or the second parent.

2.2.2 Intermediate

With intermediate recombination, the two selected parents are combined in another way. The average of both parents is calculated to form the new offspring. So looking at the first bit of both parents combining them divided by two will form the new bit of the offspring. Also cycling through all the bits in this manner.

2.2.3 Global (Discrete, Intermediate)

The global variant of discrete and intermediate recombination means that not two parents are selected for recombination but the whole population μ is used. So for discrete recombination, this means that instead of there being a 50/50 chance, there is a chance of 1/mu that the bit of that particular parent is chosen. For intermediate recombination, this means that not the average of two parents is used to form the new offspring but the average of the whole population. So the average of all first bits of all parents divided by μ.

2.3 Step Size(sigma)

In the genetic algorithms, we use mutation to give a particular randomness to the new offspring in order to give a broader search space. For evolutionary strategies, something else is used. The step size represented by the symbol σ determines the distance that the offspring will be from the parent. Higher step size will give a faster but less precise search and a smaller step size will result in more precise but slower search.

2.3.1 1/5th

The 1/5th rule is used for the (1+1) algorithms where you can compare the parent with the offspring. The best of the two is picked for the next generation. During the selection for the next generation, a counter is used to determine the successful mutations. After a set amount of generations, the success rate is calculated to adjust the sigma in the process. The following condition is used to specify when the new success rate should be calculated. Where t is evaluation count or in the case of (1+1) the generation. n is the dimension of the problem, the amount of bits used in the problem.

$$t \,\%\, n \;==\; 0$$

To calculate the success rate ps we divide the successful mutations by the evaluation count. Afterwards, the sigma is adjusted according to these formulas. The c is a constant specified by Schwefel [1].

$$\sigma = \begin{cases} \sigma(t-n)/c & \text{if } p_s > 1/5 \\ \sigma(t-n)\cdot c & \text{if } p_s < 1/5 \\ \sigma(t-n) & \text{if } p_s = 1/5 \end{cases}$$

2.3.2 One sigma

The self-adaptive one sigma operator is used for more complex algorithms consisting of the (μ+λ) and (μ,λ) notation. In comparison to the 1/5th rule–which uses a sigma for all individuals–one sigma is used to have a sigma for each individual. Resulting in a more scattered offspring compared to the parents. This operator also comes with a new variable the learning rate τ which affects the speed of adaptation for sigma. Where a bigger τ is faster but more imprecise and a small t is slower but more precise. The formula to calculate the learning rate is the following according to Schwefel [2].

$$\tau_0 = \frac{1}{\sqrt{n}}$$

With the learning rate the next step is to update or in other words, mutate the step size for the current individual.

$$\sigma' \;=\; \sigma \cdot e(\tau_0 \cdot N(0,1))$$

2.3.3 Individual sigma

Individual sigma is the next self-adaptive operator. This operator is similar to the one sigma were each individual has its own step size but a layer deeper. In the individual sigma, each bit of an individual has its own sigma. In addition to the increased amount of sigma's we replace the learning rate by a local learning rate τ and a global learning rate τ'. To calculate the two different learning rates the following formulas are used suggested by Schwefel [2]

$$\tau = \frac{1}{\sqrt{2\sqrt{n}}} \qquad \tau' = \frac{1}{\sqrt{2n}}$$

To mutate the sigma for the individual sigma operator the following function is used. Note that the normal distribution N for the global learning rate is 1 per individual and the N_i for the local learning rate is per bit on an individual.

$$\sigma_i' = \sigma_i \cdot e(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$

2.4 Mutation

After the calculation of the new sigma, the offspring needs to be mutated to give more variation compared to the parents. This mutation helps the algorithm the search in different directions and also with a bit of a random factor. Moreover using a normal distributed number multiplied with the sigma.

2.4.1 Static and 1/5th mutation

In static and 1/5th rule mutation the sigma is used for all the individuals in the offspring, thus not providing a lot of variation.

2.4.2 One sigma and individual mutation

The one sigma operator is different in the sense that every individual has its own σ which is used to mutate the individual. This gives more complexity to the algorithm, thus increasing runtime. Next to runtime the search space is more scattered by the use of different sigmas for all the individuals in the offspring. The next formulas are used to mutate the individual.

$$x' = x + \sigma' \cdot N(0,1)$$

$$x_i' = x_i + \sigma_i' \cdot N(0,1)$$

3 Evolutionary strategies

For this assignment, I chose to go for the (μ,λ) implementation. The comma notation means that the parents and offspring are not both considered to be selected for the new generation. With the comma notation, the offspring is always used for the next generation, thus forming the new parents. The selection operator used for this algorithm is tournament selection, selecting two parents for recombination. To perform the recombination I've implemented an intermediate recombination function. At first glance, this algorithm would not seem very interesting but the IOH analyzer tell you otherwise. See the result in fig.1a with also the state of the art BIPOP ES which supposedly is currently the best evolutionary strategy out there. With also an overview of the expected runtime for all my algorithms fig.1b.

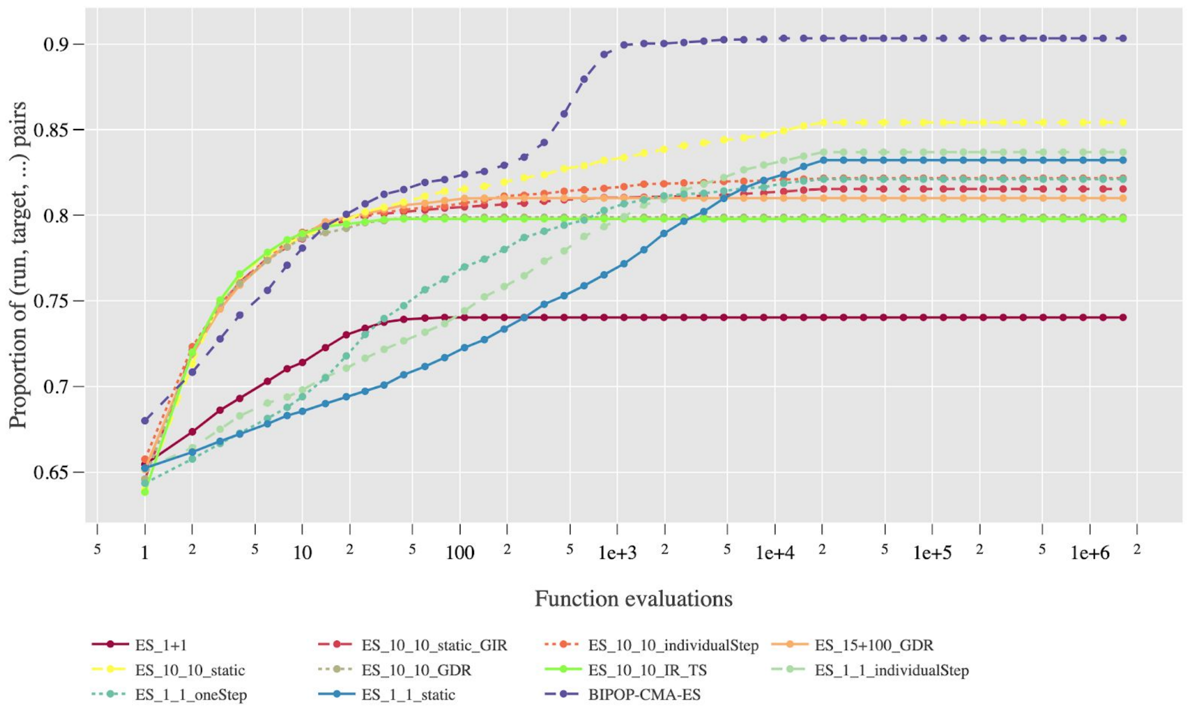

Fig.1a Different algorithms compared to the BIPOP a state of the art evolutionary strategies.

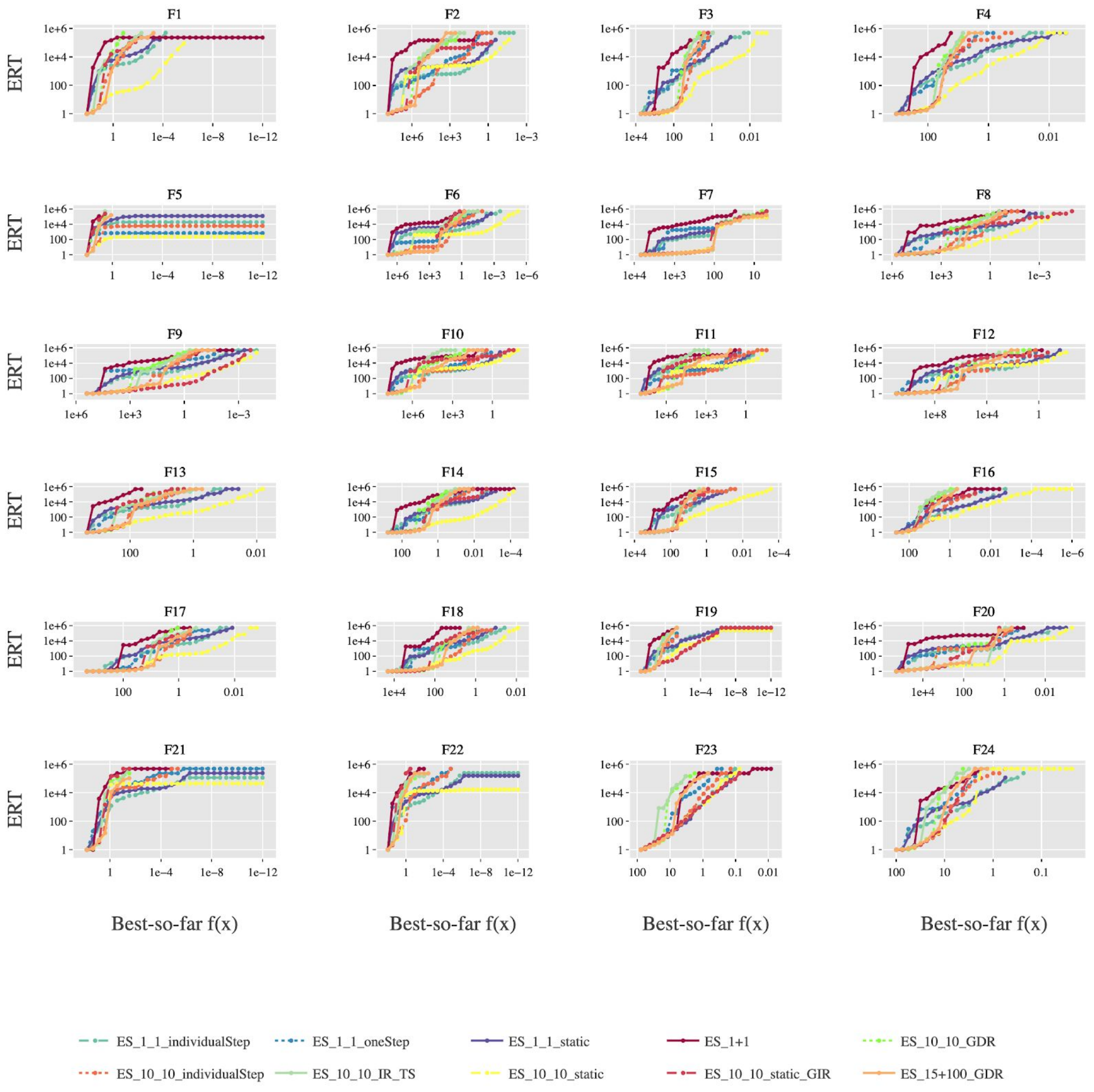

Fig.1b Expected runtime for all the algorithm I've used during this practical assignment.

3.1 Final parameters

The following parameters are used in the creation of my evolutionary strategy. As mentioned earlier I went for the (μ,λ) implementation with both of them having 10 individuals. The σ or step size used for this algorithms is 0.3 and because it is a static algorithm the step size stays the same. The k for the tournament selection operator is put at 6, followed by intermediate recombination which doesn't need any parameters for it to work. The parameters for the IOH Experimenter are: problem (1-24) of the BBOB suite, dimension (2,5,20), budget is set to 10000*n, instance 1-5 with 5 independent runs.

4 Experiments

4.1 The 1's

The smallest algorithm is the (1+1) and the (1,1). I've implemented different variations of them in order to test how step size affects an algorithm. (1+1) using the 1/5th rule to mutate the step size and three variants for the (1,1) static, one and individual sigma. In fig.2a we can see that the different step size mutations do not really affect the performance of the algorithms. Not even with a higher dimension count. However the (1+1) algorithm does outperform all the other algorithms with a dimension of 20. So depending on the dimension of the problem the (1+1) algorithm performs better.

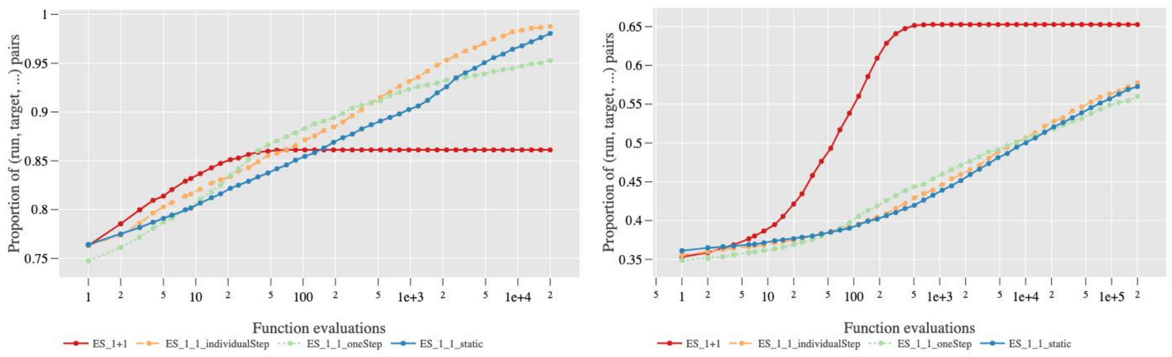

Fig.2 The aggregated empirical cumulative distribution for (1+1) 1/5th rule, (1,1) static, one and individual sigma. a. left dimension 2 b. right dimension 20

4.2 Ten comma ten

The following experiments are done using a (μ,λ) implementation both of them set to 10. Comparing the different sigma mutation again but with a larger group of individuals. The most surprising was how well the static sigma performed against the more complex algorithms that made use of different sigma mutations and recombination operators. Looking at fig.3 we can see that on almost every single function the algorithm with a static sigma performs the best. Zooming in on problem 23 we see that both of the one sigma algorithms are performing poorly.

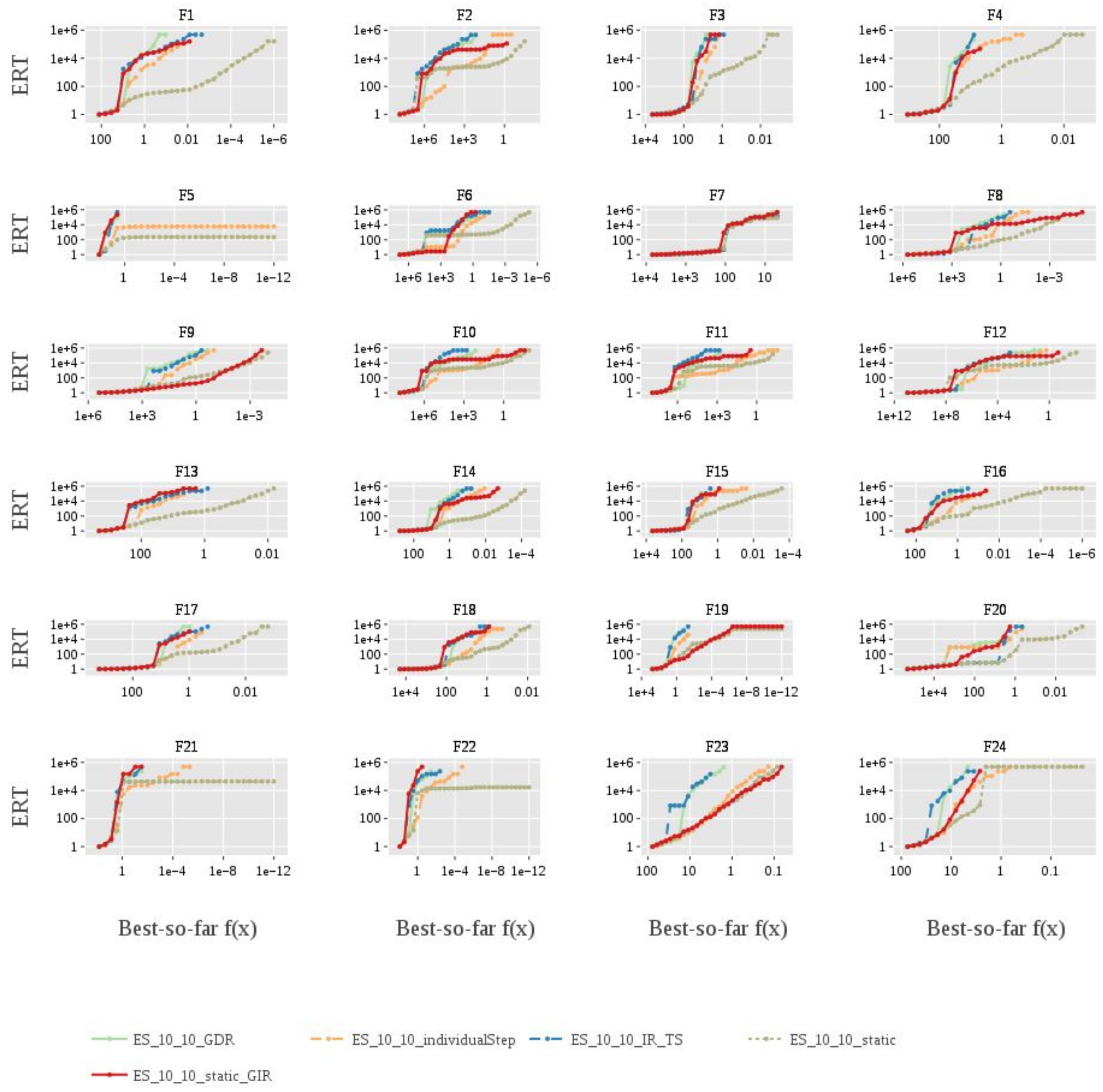

Fig.3 ERT of all function for the different (10,10) algorithms

After experimenting with different algorithms the (10,10) static sigma is still on top of the food chain. To check if any of the + notation algorithms would be able to beat it I’ve implemented a (15+100) one sigma algorithm with global discrete recombination. By using the + notation the algorithm is not discarding the parents that created the offspring, thus not killing some potentially good candidates. The (15+100) start of the same as the (10,10) but eventually flattens and is not able to overcome the static algorithm. fig.4

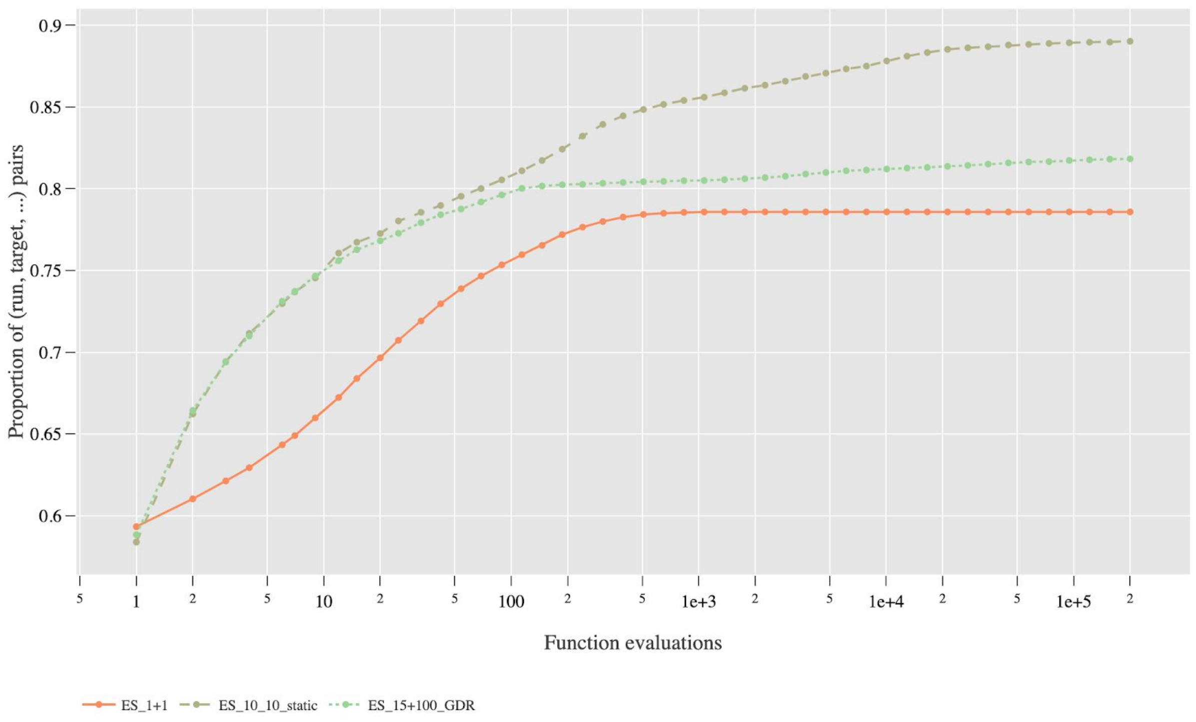

Fig.4 Comparison between (10,10) static sigma and two different + algorithms of the aggregated empirical cumulative distribution.

At last, my seemingly easy algorithm is even able to beat the state of the art BIPOP evolutionary strategy on problem 24 of the BBOB suite with a dimension of 20. It is going at a pretty stable rate stopping just slightly better.

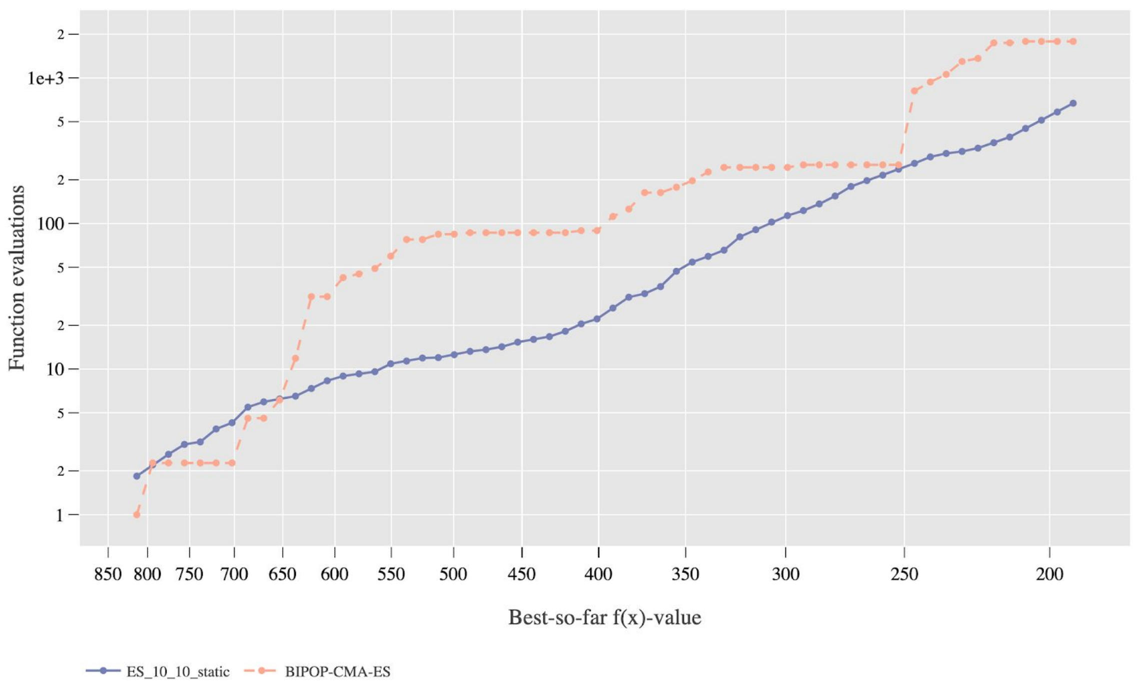

Fig.5 ERT on problem 24 for my current best algorithm (10,10) static sigma and the BIPOP algorithm

5 Discussion and conclusion

After all the experiments were done in the field of evolutionary strategies to tackle the 24 BBOB problems. The proposed algorithm is performing better than expected, because of its simplicity. The different sigma mutation did not improve the outcome of the algorithm, nor did the recombination operators. (10,10) static sigma with intermediate recombination did the trick for me, even beating the BIPOP ES in some of the given problems. For future work, I would like to go deeper into the correlated sigma mutations and try to make an algorithm that is able to outperform my current best algorithm.

Reference

[1] H.-P. Schwefel: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Birkhauser, 1977)

[2] H.-P. Schwefel: Evolution and Optimum Seeking (Wiley, 1995)