USING EVOLUTION STRATEGY TO SOLVE THE BBOB PROBLEMS

Dimitrios Ieronymakis
Student number 3372804
`d.ieronymakis@umail.leidenuniv.nl`

Maria Ieronymaki
Student number 3374831
`m.ieronymaki@umail.leidenuniv.nl`

December 8, 2021

1 Introduction

This document reports the steps followed for the implementation of an evolution strategy algorithm capable of finding the best fitness for 24 noise-free real-parameter single-objective benchmark problems provided by the Black-box Optimization Benchmarking (BBOB). Moreover, it gives a representation of the carried out experiments and finally suggests the best parameter settings to achieve the minimum possible fitness. Our algorithm's performance was evaluated across multiple functions using the aggregated ECDF curve and the AUC value provided by the IOHanalyzer platform.

2 Implementation

Our algorithm follows the Evolution strategies' framework which involves several basic steps: initialization, evaluation process, recombination for the offsprings' generation, mutation and finally selection of next generation's parent. The following subsections are devoted to provide an explanation of their implementation in our algorithm as discussed in the lectures of the Evolutionary Algorithms course given by Dr. Hao Wang and Prof. Thomas Bäck, at Leiden University. The seed used to replicate our results was set with numpy.random.seed(0).

Initialization

In this step the parameters used as an input are initialized. Among them we find the Parent size μ, Offspring size λ, Recombination type r_t, Mutation type m_t, Selection type s_t and the Fallback patience f_p. Their functionality is going to be explained thoroughly in the following subsections. This step plays a crucial role in the performance of our algorithm and therefore it is important to implement a hyperparameter tuning to find the optimal parameters that are able to achieve the minimization goal, since the results depend on the combination used.

Evaluation

The evaluation step consists in calculating the fitness score (f_i) of each individual i. These scores indicate how adequate an individual is in solving the specific problem. This process has to be implemented at every iteration, in order to evaluate the fitness of the next generation's parents.

A $budget$ has also been set to 50000, which refers to the maximum number of evaluations for each run of the algorithm. For each time a solution is evaluated, the budget decreases by one.

Recombination

The recombination step is used for the generation of offsprings. The creation of children is done iteratively and the number of offsprings created depends on the λ parameter provided in the initialization. Creation takes place by selecting the parents who are chosen through a deterministic process. Specifically, the parents' choice is fitness-independent and totally random. There are several types of recombination that can be implemented in evolution strategies and the most important ones are reported subsequently. The type of recombination used in the algorithm is defined by the parameter r_t [1].

Discrete

The $Discrete$ recombination is a method that acts per position by randomly selecting the parent from which the variable at position i will be copied to the offspring. The variables copied from the parents must be uniformly distributed over the offspring.

Intermediate

In the $Intermediate$ recombination the offspring's variable at position i is the arithmetic mean of the parents' variables in the same position. In other words, if we consider two randomly chosen parents, where $x_{1,i}$ is the variable of parent 1 at position i and $x_{2,i}$ the variable of parent 2, the offspring value in position i will be:

$$\frac{x_{1,i}+x_{2,i}}{2}$$

Global intermediate

The $Global\ Intermediate$ recombination works similar to the Intermediate recombination. However, instead of choosing only two parents to calculate the offspring, for each variable i of our individual, we consider two different random parents and then calculate the average of their variables in position i. This method produces offspring with a lot more variance and therefore is able to explore

Global intermediary

The $Global\ intermediary$ recombination generates the offspring's variable i by calculating the average of the variables in position i of all parents. The same process continues for the entire length of the parents. The formula used is the following, where j represents the index of the parent considered:

$$\frac{1}{\mu}\sum_{i=1}^{\mu} x_{j,i}$$

Mutation

The mutation step is the last process of the evolution strategy algorithm before moving on to the new population selection and repeating the process described so far [1]. It is used to introduce a small diversity by making use of normally distributed variations and it is controlled by two parameters:
- σ which is considered to be the $mutation\ step$
- τ which represents the $learning\ rate$

Self-adaptation

Instead of using the '1/5 success rule', which claims that the step size should be increased if the ratio of successful mutations to all mutations is greater than 1/5 and decreased if otherwise, the parameter control is implemented via the $Self\text{-}Adaptation$ method [2]. Specifically, parameter settings are updated in each iteration following dynamically the optimum and by adjusting the mutation step size σ to the right direction.

Individual sigma

The $individual\ sigma$ is a mutation method in which different step sizes are considered for each variable of each individual **a**. $a = ((x_1, ..., x_n), (\sigma_1, ..., \sigma_n))$

The individual sigma mutation is summarized in the steps below:

1. Individual before mutation: $a = ((x_1, ..., x_n), (\sigma_1, ..., \sigma_n))$
2. Update individual step sizes: ${\sigma_i}' = \sigma_i e^{(N(0,\tau')+N_i(0,\tau))}$
3. Mutation of the individual's variables: ${x_i}' = x_i + N_i(0, \sigma_i'))$
4. New individual after mutation: $a = ((x_1', ..., x_n'), (\sigma_1', ..., \sigma_n'))$
   where $\tau' = \frac{1}{\sqrt{2n}}$ is the global learning rate, $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$ and n is the length of each individual.

Correlated

Correlated Mutation is a mutation procedure the operates with respect to the coordinate system [3]. This method is controlled by an additional learning rate called β. The correlated mutation process is summarized in the steps below:

1. Individual before mutation: $a = ((x_1, ..., x_n), (\sigma_1, ..., \sigma_n), (\alpha_1, ..., \alpha_{n(n-1)/2}))$
2. Update individual step sizes: ${\sigma_i}' = \sigma_i e^{(N(0,\tau')+N_i(0,\tau))}$
3. Update angles: ${\alpha_j}' = \alpha_j + N_j(0, \beta)$
4. Threshold: if $\left|a'_j\right| > \pi$ then $a'_j = a'_j - 2\pi \cdot sign(a'_j)$
5. Mutation of the individual's variables: ${x_i}' = x_i + N(0, \mathbf{C}'))$
6. New individual after mutation: $a = ((x'_1, ..., x'_n), (\sigma'_1, ..., \sigma'_n), (\alpha'_1, ..., \alpha'_{n(n-1)/2}))$

   where $\beta = \frac{\pi}{36}$,τ' and τ are calculated in the same way as in the individual sigma mutation and C' is the updated covariance matrix. For simplicity we will not describe how to calculate the covariance matrix, however the implementation was based on the lectures slides.

Custom individual sigma

"He who never made a mistake, never made a discovery."

- Samuel Smiles

A mistake made while implementing the *individual sigma* mutation mechanism, led us to discover a mutation mechanism that yielded some, if not the best, results for our algorithm.
The implementation is exactly the same as the individual sigma, however, in step 2 mentioned above, we treat $N(0, \tau')$ as $N_i(0, \tau')$. This means that we don't consider this normally distributed vector as a constant for mutating each σ_i, but we generate it for each σ_i of the individual, just as we do with $N_i(0, \tau)$.
Consequently, step 2 of the individual sigma mutation mentioned above can be rewritten as:

$${\sigma_i}' = \sigma_i e^{(N_i(0,\tau')+N_i(0,\tau))}$$

Although we don't exactly understand how and why this works so well, our hypothesis is that it is able to produce more variance in our mutation and therefore push our offspring in different points in space that are not limited by the constant variable t'. Further information about results and performance of this type of mutation will be discussed in section 3.

Selection

The selection step occurs last and is used to select the new population after mutation. To avoid applying a selective pressure process, we only experimented with settings where the number of offspring λ was at least 7 times greater than the population size μ. There are two types of selection: (μ+λ) and (μ,λ) that are explained in detail below.

The $(\mu + \lambda)$ selection method considers the parents plus the new λ offspring generated. This means that the parents compete with the children in order to be part of the new generation and only the best μ out of $\mu + \lambda$ individuals are selected.
On the other hand, in the (μ, λ) selection method, parents are not considered and only the best μ out of λ offspring survive.

Fallback condition

In order to try and improve further our results, a *fallback* condition was implemented. This condition controls what kind of Selection we want to use in the current iteration of our algorithm. In general, this method was created to overcome the possibility to fall into a local optima of the $(\mu + \lambda)$ selection mechanism. To activate our fallback, we use a *curr_patience* counter, which we increase every time we do an evaluation and we reset to 0 when we find a better fitness value. Once our *curr_patience* reaches specific threshold, the fallback is activated and therefore we switch our selection mechanism from $(\mu + \lambda)$ to (μ, λ) for one iteration. Since (μ, λ) allows for backtracking and deteriorations are possible, we can therefore overcome local optima even if we mainly utilize the $(\mu + \lambda)$ selection. This idea proved to be very useful and was able to perform better in almost every scenario where the $(\mu + \lambda)$ selection was used.

Algorithm 1: Framework of the Evolutionary Strategy used

```
Input       : Parent size $\mu$
              Offspring size $\lambda$
              Recombination type $r_t$
              Mutation type $m_t$
              Selection type $s_t$
              Fallback patience $f_p$
Termination : The algorithm terminates when:
              - the budget is totally spent
$used\_budget$ = 0;
$curr\_patience$ = 0;
$fallback$ = False;
$p$ = Parents($\mu$);
$evals$ = Evaluation($func$, $p$);
$used\_budget$ += number of evals;
$curr\_patience$ += number of evals;
$i$ = 0;
while $used\_budget$ < $Budget$ do
    $p'$ = Recombination($p$,$r_t$);
    $p''$ = Mutation($p'$,$m_t$);
    $evals'$ = Evaluation($func$, $p''$);
    $used\_budget$ += number of evals;
    $curr\_patience$ += number of evals;
    $p'''$ = concatenate($p$,$p''$);
    $evals''$ = concatenate($evals$, $evals'$);
    if fallback do
        $p$, $evals$ = Selection($p'''$, $evals''$, $(\mu, \lambda)$);
    else do
        $p$, $evals$ = Selection($p'''$, $evals''$, $s_t$);
    if $curr\_patience$ < $f_p$ or fallback do
        $fallback$ = not
    $fallback$;
        $f_p$ = 0;
end
```

3 Experimental Results

This section reports the results obtained from our experimentation, together with observations we consider interesting, specifically by analysing the Empirical Cumulative Distribution Function curve (ECDF) and the Area under the Curve (AUC) in the IOHAnalyzer platform.

Mean Absolute Error

Since the number of experiments carried out was too great to simply analyze on IOH, we used the Mean Absolute Error as a metric to estimate how close our algorithm gets to the targets. For each problem (1-24), we calculate the Mean Absolute Error between our best fitted value and our targets for all the instances of the problem. However, since there are 24 problems in total, it would still be too hard to analyze and compare visually all experiments. For this reason we calculate the mean value of all the Mean Absolute Errors to map one value to each run of our algorithm. We can later use this value to compare each run of our algorithm and estimate which ones will produce the best outputs.

Empirical Cumulative Distribution Function curve (ECDF)

Several experiments were conducted to evaluate which combination of parameters was the best. In particular, all methods involving recombination and mutation step were tested, however, for simplicity, we selected only some of them, which are showed in Figure 1. The description of the Empirical Cumulative Distribution Function curve provided in the IOHanalyzer is: "The fraction of (run,target value, ...) pairs satisfying that the best solution that the algorithm has found in the i-th (run of function f in dimension d) within the given time budget t has quality at least v is plotted against the available budget t".

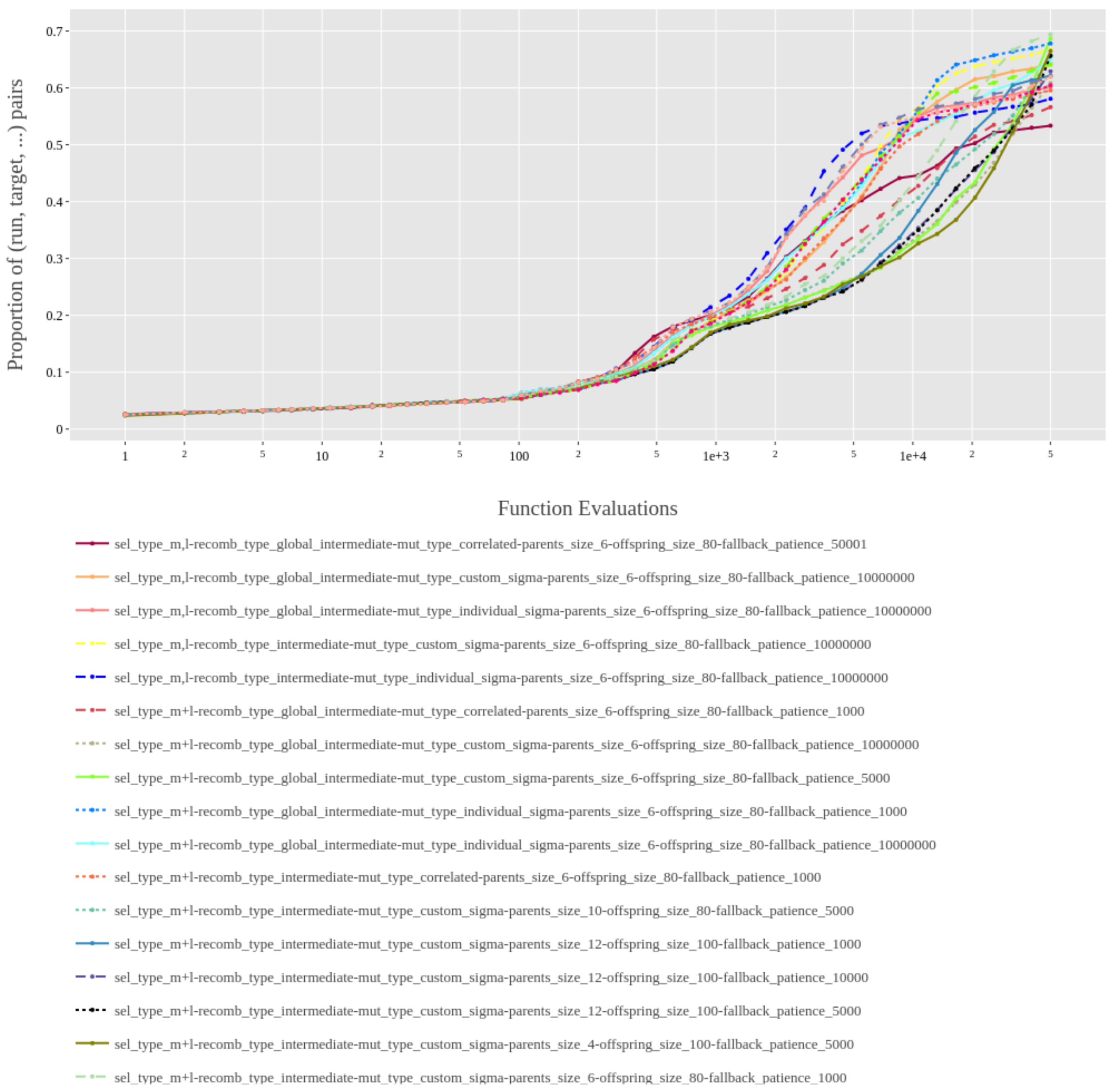

Figure 1: The ECDF curve of some experiments. The figure is downloaded from IOHanalyzer. Labels indicate the parameters tested in the algorithm.

From the figure, we can see that the (μ, λ) selection finds a larger fraction pairs with less budget, but at the same time the curve rises more slowly. On the other hand, the $(\mu + \lambda)$ has a more exponential trend and consequently needs a larger budget. The best ECDF value is slightly less than 0.7 and was obtained with the custom sigma method, a population 6 and 80 offspring.

Area Under the Curve (AUC)

The table below reports the AUC and the ECDF values of the five best experiments carried out.

μ	λ	r_t	m_t	s_t	f_p	**AUC**	**ECDF**
6	80	intermediate	custom sigma	$\mu + \lambda$	1000	0.9477	0.694
6	80	global intermediate	custom sigma	$\mu + \lambda$	5000	0.9212	0.688
6	80	global intermediate	individual sigma	$\mu + \lambda$	1000	0.9212	0.679
6	80	intermediate	custom sigma	μ, λ	10000000	0.9357	0.668
4	100	intermediate	custom sigma	$\mu + \lambda$	5000	0.9388	0.675

Table 1: Parameters of the best experiments carried out and the corresponding Area under the curve and Empirical Cumulative Distribution Function.

4 Discussion and Conclusion

In conclusion, from all the experiments carried out we can deduct that the fallback condition works great for every type of Recombination. As far as Mutation is concerned, it is very effective for *custom sigma* and *individual sigma*, while it becomes less effective when dealing with *correlated mutations* combined with *global intermediate* Recombination. This happens because the *global intermediate* implementation is not optimal and angles for the offspring are calculated randomly. This causes the algorithm to waste some budget in order to find again optimal angles to keep improving our population fitness in the next iterations. For these reasons we believe that the fallback was quite successful. In the future, it would be very interesting to try and incorporate more fallback features, like changing the Recombination or the Mutation strategy for various thresholds, in order to see if further improvement is possible.
Another important consideration has to be made for the *custom sigma* Mutation. Running it on various parameter combinations and on different seeds, it proved to be the most consistent in providing satisfying results. In general, when utilizing it, the Mean Absolute Error did not fluctuate much in comparison to using *correlated* and *individual sigma* mutations.

Finally, the best parameters have already been set on our submitted 3372804_3374831.py file and it can be run by running *python* 3372804_3374831*.py* on the terminal. We are also going to list them here for completeness:

- parents_size: 6
- offspring_size: 80
- recomb_type: intermediate
- mut_type: custom_sigma
- sel_type: m+l
- fallback_patience: 1000

These parameters result in an AUC of 0.9477 and an ECDF of 0.694. In Figure 2 below we can see its ECDF curve.

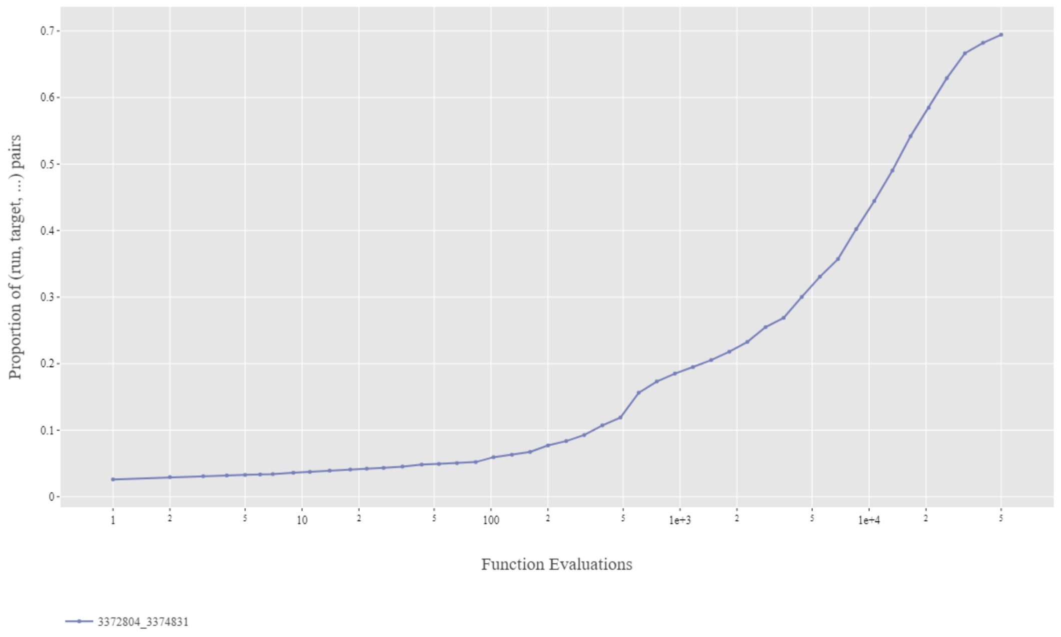

Figure 2: The ECDF curve of the best parameter settings. The figure is downloaded from IOHanalyzer.

References

[1] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. Evolution strategies. *Springer Handbook of Computational Intelligence*, page 871–898, Feb 2015.

[2] A. E. Eiben, Zbigniew Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. *Parameter Setting in Evolutionary Algorithms Studies in Computational Intelligence*, page 19–46.

[3] Nikolaus Hansen. Invariance, self-adaptation and correlated mutations in evolution strategies. *Parallel Problem Solving from Nature PPSN VI Lecture Notes in Computer Science*, page 355–364, 2000.