EVOLUTIONARY STRATEGIES

Ivan Horokhovskyi
s3069176
i.horokhovskyi@umail.leidenuniv.nl

December 8, 2021

1 Introduction

In this assignment evolutionary strategies algorithms for solving 24 BBOB problems were implemented and compared. Moreover, an optimal set of hyperparameters was suggested.

Evolutionary strategies - black-box optimization method which unlike Genetic Algorithms operates on a real number space $f(x) : R^n \Rightarrow R$ which is much more convenient to operate in real-world tasks.

2 Implementation

In this section we are going to discuss core attributes of implemented evolutionary strategies software. The pros and cons of each approach will be discussed in conclusion section.

Since each training requires considerable amount of time it was decided to make code maximally computational-efficient, this is why functional style instead of OOP was used. In addition, it was used vectorized computations which allow to speed up the process. Moreover, the program can be executed in two modes: single core and multi-core (using multiprocessing to avoid Python GIL restrictions). Even though multi-core version works correct we would like to bug-report: there is an error message in the end of execution raised by IOH library and run-folder is sometimes corrupted.

terminate called after throwing an instance of 'std::experimental::filesystem::v1::filesystem_error'
what(): filesystem error: cannot remove: Directory not empty [experiment/run_folder]

The reproduction cycle was used as it was proposed in lectures, it is shown on Figure 1. For initialization it was decided to use uniform distribution with ranges [-1, 1]. This choice is motivated by the fact that we want to have diversity in the initial population and not to stuck in local optimum. At the same time we did not want to go far from 0 value since we do not have any prior knowledge about optimal solution it might be a good idea to remain neutral and equally far from boundaries.

Algorithm 1 Generational ES Model

$t \rightarrow 0$
`Initialize`$(P(t))$
`Evaluate`$(P(t))$
while Termination criterion not met **do**
    $P'(t) \leftarrow$ `Recombine`$(P(t))$
    $P''(t) \leftarrow$ `Mutate`$(P'(t))$
    $P'''(t) \leftarrow$ `Select`$(P''(t) \cup Q)$ ▷ $Q \in \{\emptyset, P(t)\}$
    `Evaluate`$(P'''(t))$
    $P(t+1) \leftarrow P'''(t)$
    $t \leftarrow t+1$
end while

Figure 1: Reproduction cycle

2.1 Mutation

In this task we were operating with 3 types of mutations namely one-sigma, individual-sigma and correlated mutations.

One-sigma is the basic and most simple adaptation mechanism. In this approach we have a single sigma for all parameters. Next method has an individual sigma for each training parameter. The more sophisticated method is correlated mutation the idea behind that is to have an 'angle' within each unique pair of features and operates with a correlation matrix to update population.

2.2 Recombination

For recombination intermediate and discrete approaches were used. First we randomly select 2 parents (unlike GA when parents selection was based on the fitness function they have), after that we apply one of the two recombination methods to receive an offspring. The idea behind intermediate is to average the parents values, while discrete recombination is more similar to GA and is just makes random combination of parents parameters.

2.3 Selection

It was decided to try mu plus lambda and mu coma lambda. The key difference between this methods is that plus variant leaves best parents and offsprings in the population while coma variant leaves offsprings only in population (or subset of offsprings if population size is smaller than offsprings quantity).

3 Experimental Results

To pick optimal parameters both manual choosing and random search hyperparameter optimization techniques were used. The search space can be observed on Table 1.

Parameter	Ranges
Population size	logspace from 10 to 1000
Offspring size	logspace from 10 to 1000
Recombination type	intermediate, discrete
Selection type	plus, coma

Table 1: Hyperparameter search space

Obviously the main parameter we were interested in is mutation type, we have not tried a lot of experiments with single-sigma type because it is too naive and can't be used in real world cases. The AUC value we got was around 0.6. More interesting were experiments with individual-sigma, depending on configuration the AUC score was usually fluctuating in ranges 0.91-0.95 an example ECDF curve and score (0.968) example can be observed on Image 2 and 3.

The correlated-mutation variant is performing significantly slower but much better with 99 average AUC score, but our realization has some bug which we could not fix. Due to this bug time to time optimization fails.

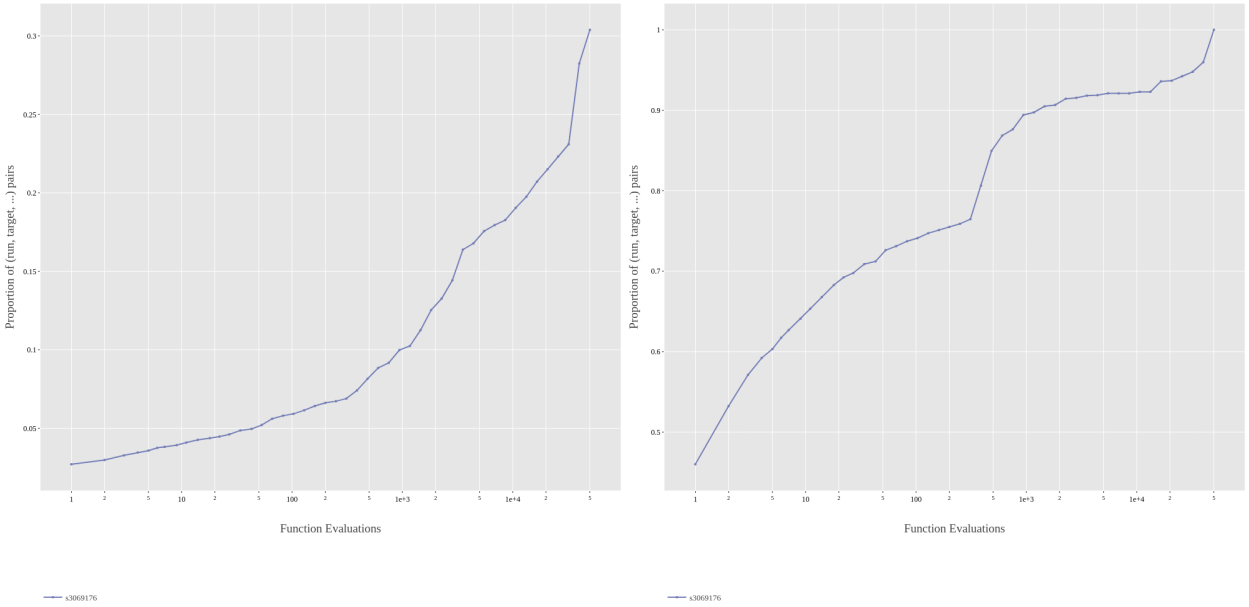

Figure 2: ECDF curve BBOB on the left and linear on the right

Figure 3: Result example

4 Discussion and Conclusion

Evolutionary strategies are state-of-the-art optimization method which allows to optimize black-box tasks (unlike for instance gradient descent). ES were explored to find a global optimum in a multi-peak optimization problems.

4.1 Population size

Unlike GA problems from the first assignment where it was fine to have population size around 20, ES requires bigger population sizes of at least more than a hundred. Such behaviour can be explained because real-numbers have enormously wider value range than binary vectors this is why they need more different experiments/mutations to converge. This hyperparameter is significant to achieve good ES performance.

4.2 Mutation

Single and individual sigma approaches are easier to implement and what is more important they are more interpretable. However in single-sigma $\sigma(t)$ and $\sigma(t+1)$ are highly correlated, this is why it is hard for algorithm to adjust the exploration space when needed, especially when features have different scale. Individual sigmas help to solve scale problem and give much more flexibility, however it does not benefit in diagonally elongated search space, in this case individual sigmas perform just as in rectangular space. Correlated mutations solves the issue by taking care of pairwise dependencies between samples in distribution with a covariance matrix.

Speed wise correlated mutations are the slowest, but in real world examples each offspring fitness evaluation is often computationally expensive and correlated mutations allow to converge in fewer amount of iterations this is why this method speed and accuracy wise outperforms single and individual sigma approaches.

4.3 Recombination

Discrete recombination allows to explore search space more rapidly due to increment of feature combination variety. While intermediate recombination is efficient in exploitation, but is not very helpful in search space exploration (unless there is variety of species in population). Overall, it is recommended to use discrete recombination for small population sizes and intermediate for large ones.

4.4 Selection

Both selection methods are deterministic which is good for experiment reproducibility and algorithm interpretability. Mu plus lambda selection allows to avoid deteriorations, however it also tend to stuck in local optimums.

Overall it is recommended it use following parameters:

1. Population size = 350
2. Offspring size = 350
3. Recombination type = discrete
4. Selection type = plus

Future work: experiment with initial parameters like τ, τ_0 and β. Find an optimum balance between exploration and exploitation for mutation. Fix correlated mutation bug. Implement global intermediary recombination.

5 References

1. Evolution Strategies, Weng Lilian, 2019, https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html