Report of TC2

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

ZHANG Xudong
PENG Qixiang
LI Zizhao
YANG Mo

October 18, 2017

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

ZHANG Xudong, PENG Qixiang, Li Zizhao, and YANG Mo

October 18, 2017

Abstract

This document firstly details the individual contributions of each group member to the outcome of the group project, benchmarking the algorithm $(\mu/\mu, \lambda)$-ES with Search Path with the COCO platform. Then the details of this project will be given, introduction and description of the algorithm, steps of algorithm realization on COCO, analysis of the result, etc. By the way, it also includes an anti-plagiarism phrase that needs to be signed by every group member.

1 Individual Contributions to the Group

1.1 Xudong Zhang

In this project, I was mainly responsible for the implementation of the algorithm. I have written the code for realizing $(\mu/\mu, \lambda)$-ES and make sure it can run correctly. Also I have tried to implement $(\mu/\mu_W, \lambda)$-ES, but not completely successful.

1.2 Mo Yang

I studied the related research of Evolution Strategies and the design of dedicated algorithms.

1.3 Qixiang Peng

I'm in charge of the COCO interface work and part of analysis work.

1.4 Zizhao Li

I have done the code review and the benchmarking, performance analysis of the implemented algorithms.

2 Anti-Plagiarism Statement

All group work, be it the source code, the documentation, the report, the presentation, or any other contribution have been solely written by the group members or corresponding reused parts have been cited accordingly. We, the group members, confirm herewith as well that we all have read and understood the "What is plagiarism" page of `http://www.plagiarism.org/plagiarism-101/what-is-plagiarism/`.

Here are our signs:

ZHANG Xudong, PENG Qixiang, LI Zizhao, YANG Mo

3 Introduction to Evolution Strategies

Evolution strategies, inspired by principles of biological evolution, represent an optimization technique that last for decades. The notion is first raised in early 1960s by students at the Technical University of Berlin (TUB). It is then developed in 1970s mainly in science of engineering. In the recent 15 years there have been increasing interests in using Evolution Strategies in domains like computation intelligence, natural computations. Despite the age of methodology, evolution strategy(ES) has shown a great performance in many cases. As an experimental result by OpenAI in a standard Reinforcement Learning benchmark(e.g. Atari/MuJoCo), ES even rivals the performance of standard reinforcement learning (RL) techniques while overcoming many of RL's inconveniences.Given this remarkable result, as students studying for learning algorithms it is totally necessary to pay attention and study the related research of it. We have studied the given paper with overview of Evolution Strategies. As the classic of the algorithm comparable to the biological notion it is not comprehensive to understand and we have thus implemented 2 algorithms 'The $(\mu/\mu, \lambda)$ - ES with Search Path' and '$(\mu/\mu_W, \lambda)$-ES' introduced by the paper.

3.1 Main principle

The idea of Evolution Strategies in optimization is that we define a population $\mathcal{P}$ which is a set of individuals. Each individual contains a solution or object parameter vector $\boldsymbol{x} \in \mathbb{R}^n$ (the visible traits), the control parameters s(the hidden traits), and an associated fitness value $f(x)$. Individuals are also denoted as parents and offspring, depending on context. A generational procedure thus is following:

(1) One or several parents are picked from the population (mating selection) and new offspring are generated by duplication and recombination of these parents.

(2) The new offspring undergo mutation and become new members of the population.

(3) Environmental selection reduces the population to its original size.

The main principles are following:

- **Environmental Selection:**select the u best individuals from population survive based on the individuals' fitness, f(x).
- **Mating selection and recombination:**mating selection picks individuals from the population to become new parents. Recombination generates a single offspring from these parents.

- **Mutation, Parameter Control:**Mutation introduces small, random and unbiased changes to an individuals. The average size of this change depend on endogenous parameters called control parameters. For example, in this case the step-length of mutation is controlled by step-size σ. And a Covariance matrix C determinate the shape of distribution by mutation.

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

In this section we will introduce the dedicated algorithms to be implemented later. For Evolution Strategies, there are several key points in algorithms design as mentioned before:

- **Environmental Selection**
- **Mating selection and recombination**
- **Mutation, Parameter Control**

In the following we will focus on these key points on dedicated algorithms

Algorithm 1 The $(\mu/\mu, \lambda)$ - ES with Search Path

given $n \in \mathbb{N}$, $\lambda \in \mathbb{N}$, $\mu \approx \lambda/4 \in \mathbb{N}$, $c_\sigma \approx \sqrt{\mu/(n+\mu)}$, $d \approx 1 + \sqrt{\mu/n}$, $d_i \approx 3n$
initialize $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\sigma} \in \mathbb{R}^n_+$, $\boldsymbol{s_\sigma} = \mathbf{0}$
while not happy **do**
    for $k \in \{1, ..., \lambda\}$ **do**
        $\boldsymbol{z_k} = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ iid for each k
        $\boldsymbol{x_k} = \boldsymbol{x} + \boldsymbol{\sigma} \circ \boldsymbol{z_k}$
    $\mathcal{P} \leftarrow sel\_\mu\_best(\{\boldsymbol{x_k}, \boldsymbol{z_k}, f(\boldsymbol{x_k}) | 1 \le k \le \lambda\})$ recombination and parent update
    $\boldsymbol{s_\sigma} \leftarrow (1 - c_\sigma)\boldsymbol{s_\sigma} + \sqrt{c_\sigma(2 - c_\sigma)}\frac{\sqrt{\mu}}{\mu} \sum_{\boldsymbol{z_k} \in \mathcal{P}} \boldsymbol{z_k}$
    $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \circ exp^{1/d_i}(\frac{|\boldsymbol{s_\sigma}|}{\mathbb{E}|\mathcal{N}(0,1)|} - \mathbf{1}) \times exp^{c_\sigma/d}(\frac{||\boldsymbol{s_\sigma}||}{\mathbb{E}||\mathcal{N}(\mathbf{0},\mathbf{1})||} - 1)$
    $\boldsymbol{x} = \sum_{x_k \in \mathcal{P}} x_k$

From $(\mu/\mu, \lambda)$we can tell that

- The parent population contains μ individuals
- All μ parents will be used in mating

- 'Comma' means that in Environmental selection, algorithm removes individuals out of age, only new offspring survive to the next generation.
- λ offspring will be generated in each iteration

In this algorithm, f is the function we want to minimize, $\boldsymbol{x}$ is its parameters. The exogenous strategy parameters are given in the begining: n the dimension of $\boldsymbol{x}$, μ the number of parents in the population $\mathcal{P}$, λ the number of offspring generated in each iteration. Also, s_σ is introduced as search path which will be used to update step-size in iteration.

Line 2 indicate the initialization step. Only a single parental centroid ($\boldsymbol{x}$, $\boldsymbol{s}$) is initialized.$\boldsymbol{x}$ here serves as beginning search point and $\boldsymbol{s}$ is the path control parameters, the step-size σ in this case. Search path s_σ is also initiated to zero. The fist mutation will take these parameters as input.

Line 3-10 indicate the procedure of each iteration. Line 4-6 shows the mutation procedure. λ offspring will be generated at the beginning of each loop, the only control parameter used here is the step-size σ. In line 7, u best of generated offspring will be selected and added to population based on f-values.(Environmental selection) All members of population will become new parents. Line 9-10 control parameter is updated. And the change of step-size σ is determined by the length of search path $||\boldsymbol{s_\sigma}||$. "Recombination" is postponed to the end of the loop, computing in line 10 with a new parental centroid to be used in the next iteration. The iteration will continue until stopping criteria or failure.

4 Implementation of $(\mu/\mu, \lambda)$-ES

In this section, we discuss the details for the implementation of algorithm $(\mu/\mu, \lambda)$-ES. Also in the paper Evolution Strategies, the author presented an algorithm $(\mu/\mu_W, \lambda)$-ES which can be seen as a general version of $(\mu/\mu, \lambda)$-ES. We also tried to realize this algorithm in the project. Although not completely successful, it is almost completed with lack of some numerical treatment to make up the loss of precision in calculation. We also present the details of implementation compactly for this algorithm, just for reference and comparison.

For realizing the algorithm, we use python as the programming language for its simplicity and convenience.

4.1 Initialization of Parameters

The first thing needed to be done is initialization of parameters. In order to facilitate the readers, here we presented the initialization of parameters with the same order in paper Evolution Strategies.

n is the dimension of the function, it can be initialized with the help of fun defined in coco. Just use $n = fun.dimension$ will finish the initialization of n.

The next parameter is λ number of offspring. Usually, we will not set λ a big value cause this will slow down the speed of calculation. In the algorithm, as $\mu \approx \lambda/4 \in \mathbb{N}$, we choose $\lambda = 8$. Although whether λ can be divided by 4 does not matter much, we still choose λ to be the multiple of 4 just because of a little OCD (Obsessive-compulsive disorder).

With n and λ well defined, other parameters μ, c_σ, d and d_i can be initialized according to n and λ.

For the parameter $\boldsymbol{x}$, we do not initialize it randomly. As the answer can be in any place in the searching area. Here, we put $\boldsymbol{x}$ in the center of the searching area, ie. $\boldsymbol{x} = (fun.lower_bounds + fun.upper_bounds)2$.

Finally, we put $\sigma = 0.02$ and we initialize $\boldsymbol{s_\sigma}$ with zero vector and we finish the initialization.

4.2 Construction the Loop

Although the loop part is the core of the algorithm, it is much easier to implement the loop than initialization of parameters. One can construct the loop simply by translating the pseudo code into python. However, there still exists some little difference and some tricks to facilitate coding. We present them below.

Firstly, as the function needs to return $\boldsymbol{x}$ which corresponds to the point at which we find the smallest value on the moment. We need to set a variable $\boldsymbol{x_{min}}$ and also the function value corresponded $\boldsymbol{F_{min}}$. We initialize $\boldsymbol{x_{min}} = 0$ and $\boldsymbol{F_{min}} = inf$. If we found a smaller function value, we change $\boldsymbol{x_{min}}$ and $\boldsymbol{F_{min}}$ with $\boldsymbol{x}$ and $fun(\boldsymbol{x})$.

One trick to facilitate coding is to change the inner for loop with much simpler and speed-faster python codes. With the help of module $numpy$, w can create all the $\boldsymbol{z_k}$ by $nump.random.multivariate_normal$. This is much quicker for that is takes much time to execute for loop in python.

For the rest pseudo codes, just translating them into python will solve the problem.

Last but not least, the end condition for the loop is the number of budget that we set in coco.

4.3 Semi-implementation of $(\mu/\mu_W, \lambda)$-ES

As is stated before, we have also tried to realize the algorithm $(\mu/\mu_W, \lambda)$-ES. It is almost finished. However, for the limit precision of computer, when we compute the square root of the co-variance matrix $\boldsymbol{C}$, sometimes it will not give correct answer for that the determinant of $\boldsymbol{C}$ is too small that the computer treat it as zero. We have not implemented some necessary numerical tricks to solve the problem because of the limited time. However, the logic of codes have been proved to be right.

The process to implement $(\mu/\mu_W, \lambda)$-ES is almost the same as $(\mu/\mu, \lambda)$-ES except now we need to update the co-variance matrix $\boldsymbol{C}$ in each loop. We strongly recommend to use $numpy$ to realize the product of matrix for that it is easy to use and also fast for executing. The codes (although can not be used now) is just below the implementation of $(\mu/\mu, \lambda)$-ES in the script, just for reference.

5 Information about the timing experiment

After the implementation of the algorithm, we test it on the Benchmark COCO [1].

COCO aims at providing several test suites for COmparing Continuous Optimizers in a black-box setting and generating plots and tables in HTML and LaTeX/PDF displaying algorithm performances and comparisons. For the single-objective noiseless 'bbob' test suite, clearly more than 100 algorithm data sets are already freely available, for the bi-objective 'bbob-biobj' suite, 16 algorithm data sets are available so far.

Since using python, what we did mainly is to re-write the **example_experiment.py**. And we keep those normal settings unchanged.

Firstly, as usual on COCO, we implement the algorithm as a function, called **solver** which will be called in the **coco_optimize**. It takes x_0 and as the input, which means the initial population and mutation step-size.

Then we change part of function of **coco_optimize**, here we give the initial value of x_0 and .

Finally, what we should do is just to redirect the variable SOLVER to our algorithm function.

Hence, we can run our algorithm with test suit bbob on COCO by the command:

```
python my_experiment.py bbob
```

Afterwards, we increase the budget, for exemple, 1000, 10000, etc, and test for several times by:

[1] The open source is available at: https://github.com/numbbo/coco/

```
python my_experiment.py bbob 1000
```

After testing our algorithm as a black-box on bbob, we have data-set which are saved in the folder **exdata**, and we are supposed to post-process it by

```
python -m cocopp -o OUTPUT_FOLDER exdata
```

Then, OUTPUT_FOLDER will be generated, which contains all output from the post-processing, including an index.html file, useful as main entry point to explore the result with a browser. And it's easy for us to analyse our algorithm's performance.

6 Discussion of the results

In this section, we will compare our algorithm $(\mu/\mu, \lambda)$-ES with Search Path with two baseline algorithms: BIPOP-CMA-ES (BI-population CMA-ES) and BFGS.

Newton method is a kind of descent optimization method of which the descent direction is Newton direction $-[\nabla^2 f(x_k)]^{-1}\nabla f(x_k)$. In order to calculate Newton direction, we should compute the inversion of Hessian matrix which needs the objective function is second differentiated and we should notice that the inversion of Hessian matrix is hard to compute. Thus, there appears lots of quasi-Newton method of which the main point is using an approximation of the inverse Hessian matrix. BFGS is one the most popular implementation of quasi-Newton method.

The algorithm we implemented and BIPOP-CMA-ES are stochastic optimization algorithms in the category of evolutionary algorithm which always accompanies with recombination and mutation. The difference between CMA-ES and $(\mu/\mu, \lambda)$-ES with Search Path is that CMA-ES adapts three parameters: mean, step size and covariance matrix, the algorithm $(\mu/\mu, \lambda)$-ES with Search Path only introduces first two parameters, the covariance matrix respects always the distribution $\boldsymbol{N}(0, \boldsymbol{I})$. Compared with CMA-ES which could be in some functions with many local optima, the BIPOP-CMA-ES 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.

In our test case, we want to optimize $f : \Omega \subset \boldsymbol{R^n} \mapsto \boldsymbol{R^k}$ where $n = 2, 3, 5, 10, 20, 40$, $k = 1$, f is the 24 test functions implemented in coco of which the goal is to evaluate the characteristics of optimization algorithms.

We will compare the algorithms with a fixed budget of $1000 * dimension$ and for each group of functions.

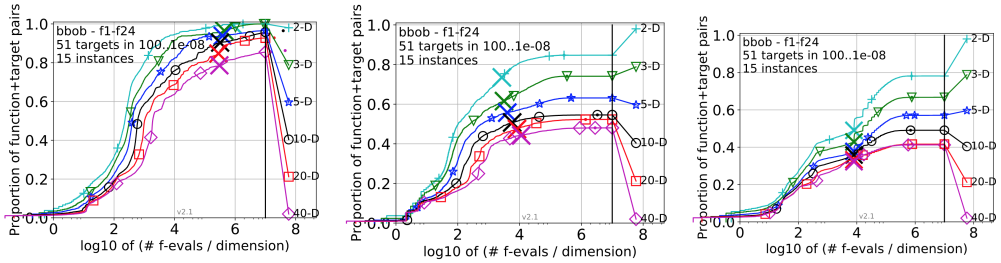

Figure 1: Runtime distributions (ECDFs) over all targets (from left to right: BIPOP-CMA-ES, BFGS, $(\mu/\mu, \lambda)$-ES with Search Path)

As we said above, we only focus on the 1000 numbers of f-evaluation. We observed these three algorithms have similar performance in this range from the empirical cumulative distribution functions (ECDFs) curve (Figure 1). However, if we took a look at the run-time distribution summary per group, we notice that globally, these three algorithms all performed well in the first function group which contains five separable functions. With the augmentation of variable dimension and complexity of evaluated function, the performance of BIPOP-CMA-ES is much better than the others. Simply, we could implement a restart strategy to improve the performance of our algorithm and for BFGS, we could smooth the evaluated function before applying the algorithm or choose a better start point.

7 Conclusion

After this project, we clearly understand the ES series algorithms. The most important step is how to generate good and useful offspring, and, in our opinion, it depends on the good strategy of mutation and parameter control. In fact, many improvements are focused on this point. Firstly, we take the 1/5th rule to update the mutation step-size σ. Afterwards, we use self-adaptation on σ, just like our implemented algorithm. But it's not enough, so we use self-adaptation on covariance matrix: CMA(we implemented part of it). Further more, we could introduce a new restart strategy to obtain better performance. At the level of benchmark we could evaluate the algorithm with a much bigger budget to test the stability of algorithm.

In conclusion, we got the expected result by implementing the $(\mu/\mu, \lambda)$-ES with Search Path algorithm, and this is a meaningful project.