Abstract

This paper explores Evolutionary Strategies (ES), a subset of natural computing algorithms characterized by self-adaptation, wherein both solution and algorithmic parameters coevolve. We implement a specific ES algorithm, employing it to optimize well-known functions such as Sphere, De Jong N5, and Schwefel. Beyond merely finding optimal solutions, the study aims to analyze the impact of varying algorithmic parameters. Experimental results for each function are discussed, leading to key conclusions about the efficacy and adaptability of the ES approach.

Mail	florez.fernandez.david@gmail.com

Contents

1 Introduction

Evolutionary strategies (ES) are another member of the natural computing algorithms. This kind of algorithms have a self-adaptive feature. This means that not only possible solutions of the problem are in the chromosome but also the strategy parameters. So the parameters that reigns the algorithm coevolve with the solutions and because of this this property is referred as self-adaptation. Table 1 shows a brief summary of what kind of ES we will use.

Representation	**Real-valued vectors**
Recombination	No Recombination, Discrete or Intermediate
Mutation	Gaussian Perturbation
Parent Selection	Uniform Selection
Survivor Selection	(μ, λ), $(\mu + \lambda)$

Table 1: Summary of the ES implemented.

The purpose of this activity is the implementation of a strategy algorithm (detailed in Section 2) and thereafter use it to resolve some optimization problems for well-known functions: Sphere, De Jong N5 and Schwefel (see Section 3 for details). The resolution of those problems is not only focused in finding the optimal solution. Otherwise it is meant to analyze the influence of different parameters configuration and parameters trying to find answers reasonably in each case. Section 4, Section 5 and Section 6 will cover some experiments for those functions. Finally in Section 7 we end with the most relevant conclusions of this study.

2 Implementation

The software chosen for this activity was Matlab (Version: 7.10 R2010a). It was decided to make this program using a functional paradigm. There are 2 main functions depending in the class of the function used. For Sphere and Schwefel it is used the *ES_fhandle.m* and for De Jong N5 it is used *ES_DeJong5.m*. This split is explained because Sphere and Schwefel have an closed analytical function which can be resolved using matrix algebra. However De Jong N5 needs loops in the calculation. By this way we have one function with Sphere and Schwefel in *funct.m* and De Jong N5 in *dejong5.m*. The configuration of parameters it is made using external csv files. Then the algorithm (ES_fhandle and ES_DeJong5) follows the description given in Reference [1], that is:

1. Initialize objects variables and their standard deviations from random if not one is provided by the user. Function *inition.m* makes this step.
2. Recombinate objects variables and standard deviations. Function *recombination.m* is in charge of this step for x and σ.
3. The mutation step. Function *mutation.m* makes the mutation for solutions and strategy parameters, for both the mutation is the same, i.e. it is not possible to make an intermediate mutation on parameters and a discrete in variables. Here it was introduced a cap (1%) in the mutated volatility to avoid standard deviations near zero. Also when the object variables fall outside the limits defined in the function they are caped to the limit exceeded. Also it is important to notice that we have a factor named as *mutation power* that multiply the learning rate τ.
4. Survivor selection for the next step. Function *survivor.m* is used here.

In Table 1 it is summarized the methods implemented for each strategy procedure in the ES algorithm. And Figure 1 depicts the algorithm graphically. Methods of each procedure was made following Reference [2].

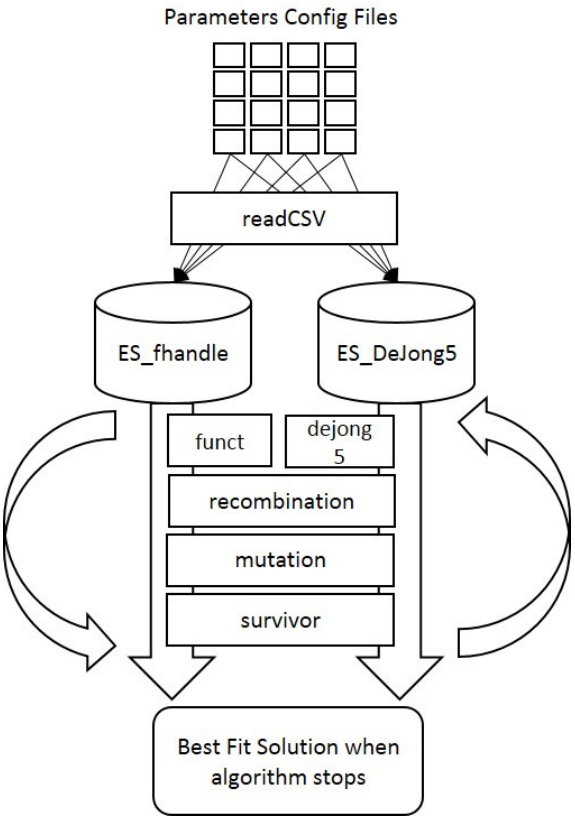

Figure 1: Sketch of the algorithm implemented.

We will use some hard coded parameters, and they are:

- **Error tolerance:** It is first coded as $10e^{-8}$ but after a first analysis with the Sphere function (Section 4) it was decided to relax the error to $10e^{-4}$. This has a strong effect in the velocity of the algorithm.
- ϵ_{Vol}**:** It is coded as 1%. That means that there are not σ below 1%.
- **Stop criteria:** A maximum 30 generations without changes are admissible, after 30 generations without changes program stops.

3 Functions

This section will explain what problem are meant to be resolved. Each function will be described in one subsection and there we will give the mathematical description and graphics for the case with 2 variables. And also it will be given the global minimuns for each function and other aspects related with the searh of that minimums (local minimums, space of definition).

3.1 Sphere

Sphere function is bowl-shaped and it is defined as in Equation (1). It is continuous, convex and unimodal. Figure 2 shows its two-dimensional form. It is the most easy function considered for searching the minimum, because its shape and also the small domain we will considered.

Definition Domain: It will be evaluated on the hypercube $x_i \in [-10, 10]$, $\forall\ i = 1, \ldots, n$.

Global Minimum: It has one global minimum at $f(x^*) = 0$ with $x^* = (0, \ldots, 0)$.

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{1}$$

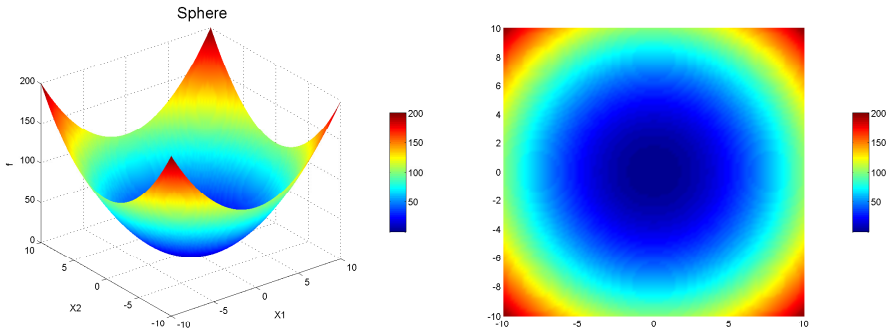

Figure 2: Sphere 3D representation (left) and 2D representation (right) with color legend.

3.2 De Jong N5

The fifth function of De Jong is continous, multimodal, with very sharp drops on a mainly flat surface. Equation (2) shows the mathematical formulation. Figure 3 shows its two-dimensional form. This function has a high difficult in the searching of the minimum because we have a huge domain in a flat surface with the minimums at the center of that plane. Here will be very important the initial x_0 and the population size.

Definition Domain: It will be evaluated on the plane $x_i \in [-65536, 65536]$, $\forall\ i = 1, 2$.

Global Minimum: It has one global minimum at $f(x^*) = 1$ with $x^* = (-32, -32)$ and 25 local minimums at points (a_{1j}, a_{2j}) with $j = 1, \ldots, 25$

$$f(x) = \frac{1}{\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}} \tag{2}$$

$$A = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{pmatrix}$$

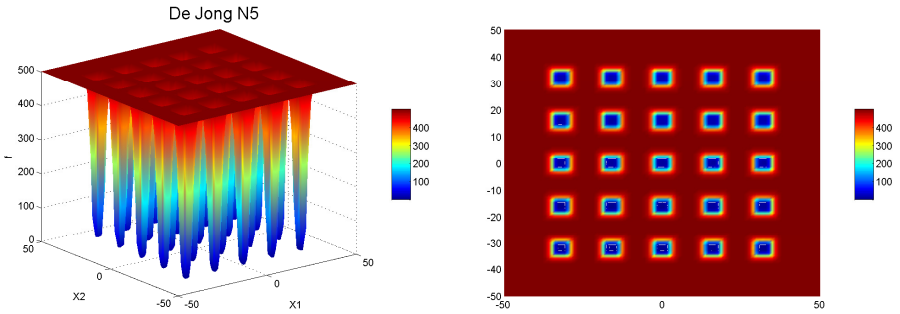

Figure 3: De Jong N5 3D representation (left) and 2D representation (right) with color legend.

3.3 Schwefel

The Schwefel function is complex, with many local minima. Equation (3) shows the mathematical formulation. Figure 4 shows its two-dimensional form. This function has a high difficult in the searching of the minimum because we have the big domain and because it has many local minimums. Here will be very important the initial x_0 and the population size.

Definition Domain: It will be evaluated on the hyperplane $x_i \in [-500, 500], \forall\, i = 1, \ldots, n$.

Global Minimum: It has one global minimum at $f(x^*) = 0$ with $x_i^* = 420.9687$.

$$f(x) = 418.9829 \cdot n + \sum_{i=1}^{n} (-x_i \cdot sin(\sqrt{|x_i|})) \tag{3}$$

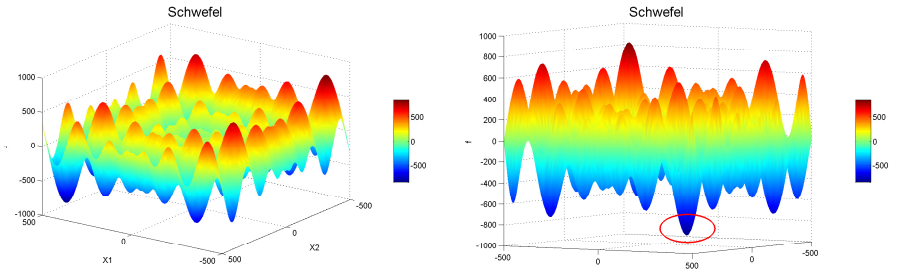

Figure 4: Schwefel 3D representation (left) and lateral representation (right) with color legend. The global minimum is highlighted in red.

4 Sphere

We are going to use the ES implemented to solve the minimum searching of the function Sphere. Table 2 summarizes the list of configurations and the name used to denote them. Here we initiate the variables and sigmas randomly. Fitness function is the error in absolute terms.

Analysis on the error tolerance:

ConfigName	mu	lambda	nGen	selection	recomb_var	recomb_str	mut_type
SphereConfig1	30	200	1200	mu.lambda	Intermediate	NoRecomb	1step
SphereConfig2	30	200	1200	mu.lambda	Intermediate	Intermediate	nstep
SphereConfig3	30	200	1200	mu+lambda	Intermediate	NoRecomb	1step
SphereConfig4	30	200	1200	mu+lambda	Intermediate	Intermediate	nstep

Table 2: Configuration definition used in the following analysis.

First of all we are going to analyze what error tolerance is adequate to use in this ES. We had run 30 executions with *SphereConfi1* using a tolerance of $10e^{-8}$ and in the 30 executions the number of executions was 1200. We considered to decreased this tolerance to $10e^{-5}$ obtained an average of 164 generations to achieve an error less than that tolerance.

Analysis on Uncorrelated mutation with (μ, λ)-Selection for different mutation methods (SphereConfig1 vs SphereConfig2):

We are going to compare for (μ, λ)-Selection the impact of using 1 step size versus n step size mutation. The mutation case 1 step size we input *No recombination* for the strategy parameters because it is necessary in ouR code. This is not relevant because with the same σ in every variable the recombination will always return the same σ. So with the 1 step mutation we are losing the recombination variation power. We ran 30 random experiments with the ES applied to Sphere function using SphereConfig1 and SphereConfig2. Figure 5 shows the best fit solution in each execution using 1 step mutation and n step mutation. In those cases when the dotted line is not continous it is because in that execution the ES returns 100. There are two important conclusions:

- nstep uncorrelated mutation is much more slow than 1step. For nstep the maximum number of generations was achieved always. For 1step was necessary 160 simulation in average.
- nstep mutation does not reach the tolerance error and even there are three executions with no convergence at all, with fitness equals 100 (these was eliminated from the chart).

Analysis on Uncorrelated mutation with $(\mu + \lambda)$-Selection for different mutation methods (SphereConfig3 vs SphereConfig4):

Now we will carry out the same analysis but with $(\mu + \lambda)$-Selection. Figure 6 shows the big difference between the two different kind of mutation. 1step is again much better than nstep with smaller fitness.

Analysis on Selection:

If we enter to consider the different kind of selection, we can compare our previous results fixing the kind of mutation. So if we compare 1step mutation we find that (μ, λ) always reach the error tolerance and $(\mu + \lambda)$ sometimes doesn't reach the tolerance as depicted in Figure 7.

For the nstep case, the things occurs in the inverse way. It is better in terms of fitness $(\mu + \lambda)$ than (μ, λ) as clearly seen in Figure 8.

Analysis on progress speed:

If we compare the progress curves for two random runs of $(\mu + \lambda)$ selection with 1step and nstep we find that 1step is clearly quicker than nstep. Table 3 summarizes the average and standard

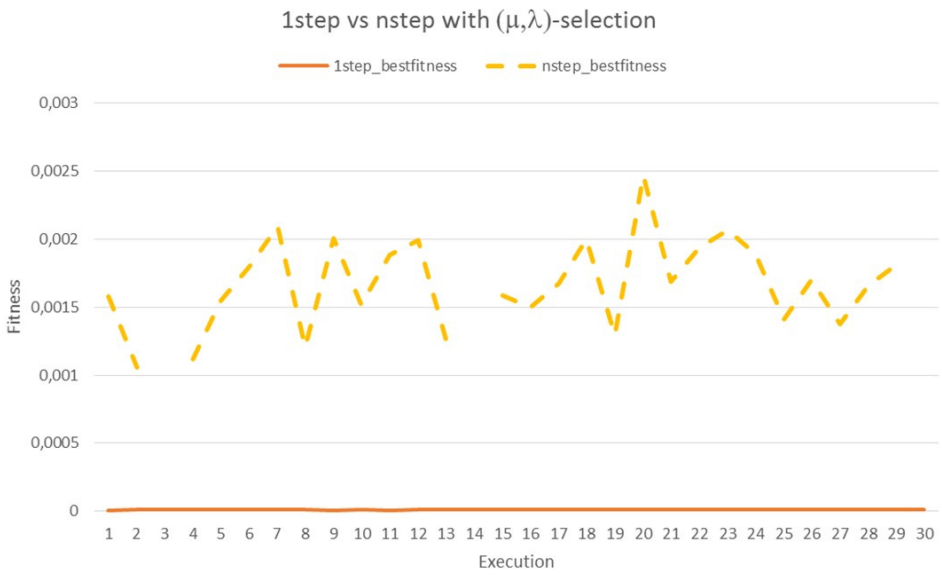

Figure 5: Best fitness solution in each execution with (μ, λ) selection and mutation 1step or nstep for Sphere.

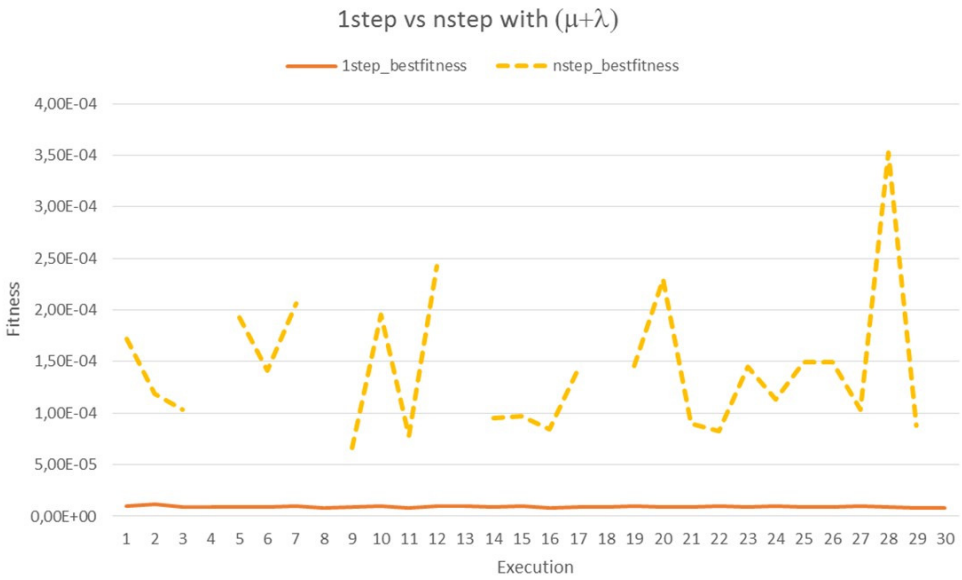

Figure 6: Best fitness solution in each execution with $(\mu + \lambda)$ selection and mutation 1step or nstep for Sphere.

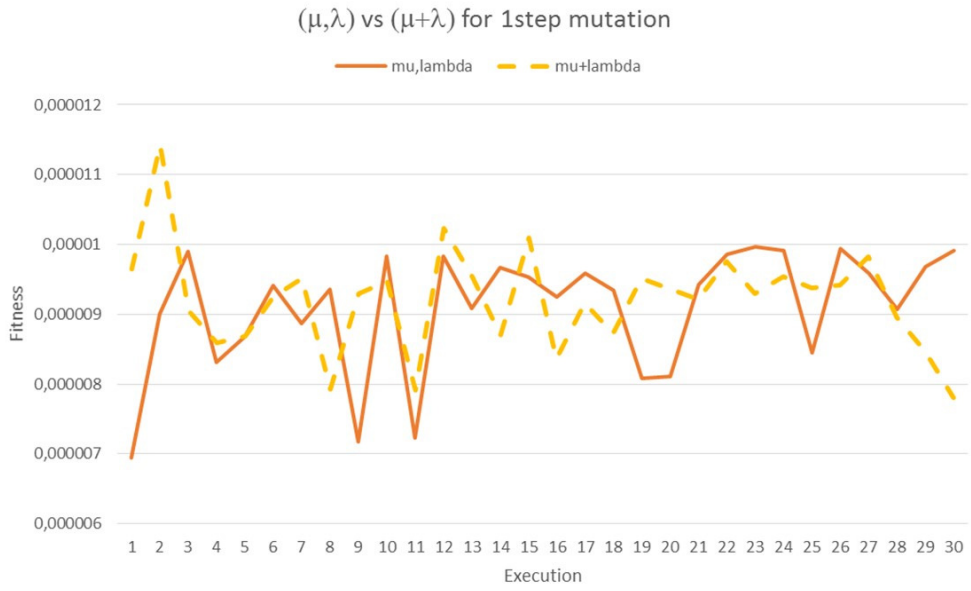

Figure 7: Best fitness solution in each execution with $(\mu + \lambda)$ selection and (μ, λ) selection for mutation 1step for Sphere.

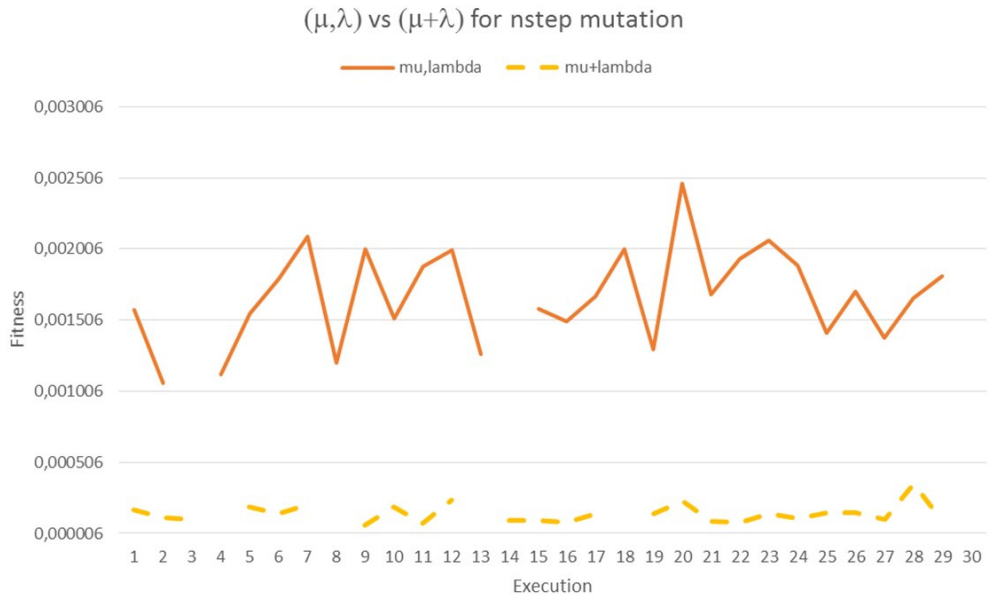

Figure 8: Best fitness solution in each execution with $(\mu + \lambda)$ selection and (μ, λ) selection for mutation nstep for Sphere.

deviations (without those cases when the ES does not evolve) for the configurations used. We can see that nstep mutation is always slower than 1step (1vs2 and 3vs4) and $(\mu + \lambda)$ is also faster than (μ, λ) (1vs3 and 2vs4). So with the information given by these 30 simulations and for the case of the Sphere, SphereCongfig3 is better than the others.

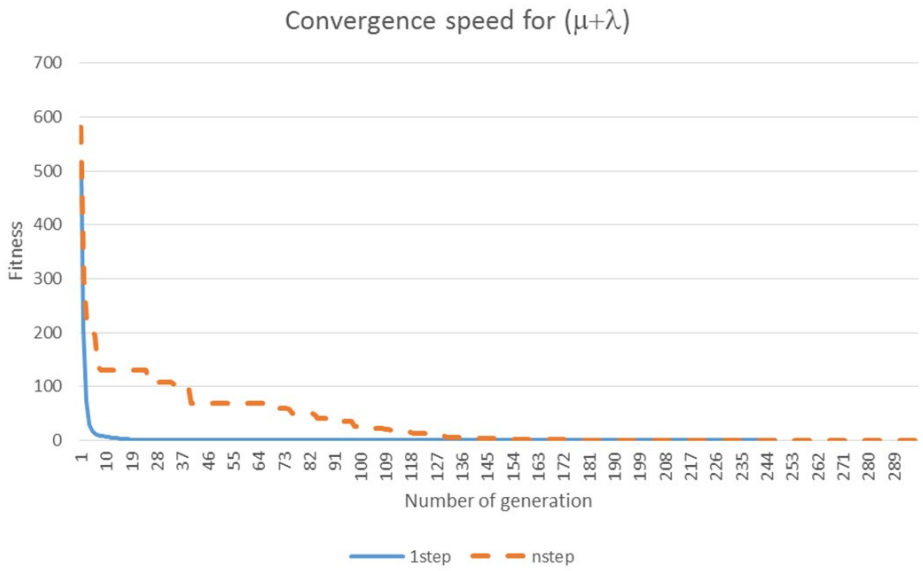

Figure 9: Progress curves for 1step and nstep mutation for a random simulation with $(\mu + \lambda)$ selection for Sphere.

Config	Generation Average	Std. Deviation	Fitness Average	Standard Deviation
SphereConfig1	163,93	79,39	9,1E-06	8,7E-07
SphereConfig2	1200,00	0,00	1,7E-03	3,4E-04
SphereConfig3	97,80	16,11	9,2E-06	7,4E-07
SphereConfig4	401,07	137,43	1,4E-04	6,6E-05

Table 3: Statistics of the 30 runs on the 4 configurations for Sphere.

5 De Jong N5

We are going to use the ES implemented to solve the minimum searching of the function De Jong N5. Table 4 summarizes the list of configurations and the name used to denote them. De Jong N5 has a big domain $[-65536, 65536]^n$ and there it is almost plane expect somewhere in the $[-50, 50]^n$ so the searching of the optimum with this function requires a lot of exploration. After some tests it was decided to grow the population size to $\mu = 200$ and $\lambda = 700$.

ConfigName	mu	lambda	nGen	selection	recomb_var	recomb_str	mut_type
DeJongConfig1	200	700	1200	mu.lambda	Discrete	NoRecomb	1step
DeJongConfig2	200	700	1200	mu.lambda	Discrete	Discrete	nstep
DeJongConfig3	200	700	1200	mu+lambda	Discrete	NoRecomb	1step
DeJongConfig4	200	700	1200	mu+lambda	Discrete	Discrete	nstep

Table 4: Configuration definition for De Jong N5 used in the following analysis.

Analysis on Uncorrelated mutation with (μ, λ)-Selection for different mutation methods (DeJongConfig1 vs DeJongConfig2):

To analyze the effect of different mutation method we compare 30 simulations of the ES applied to De Jong N5 function with the (μ, λ)-Selection. Figure 10 presents graphically the fitness for each kind of mutation. Some points were deprecated from the chart because they are stagnated in the ES. With this information it is difficult to say which one works better for this function. With the information about the average and standard deviation in Table 5 between DeJongConfig1 and DeJongConfig2 we can conclude that with this sample both methods are very similar.

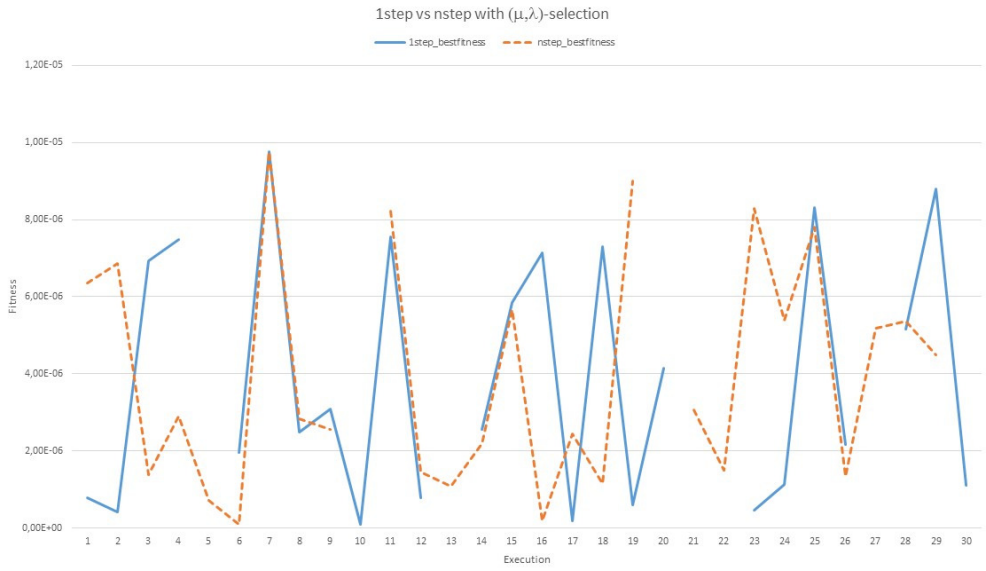

Figure 10: Best fitness solution in each execution with (μ, λ) selection and mutation 1step or nstep for De Jong N5.

Analysis on Uncorrelated mutation with $(\mu + \lambda)$-Selection for different mutation methods (DeJongConfig3 vs DeJongConfig4):

For the case of $(\mu + \lambda)$-Selection we concluded the same than the previous case. Both DeJongConfig3 and DeJongConfig4 work very similar with number of generations on the same order and fitness very close to zero. Figure 11 shows the fitness for this case. And also in Table 5 is presented the basic stats on generations and fitness.

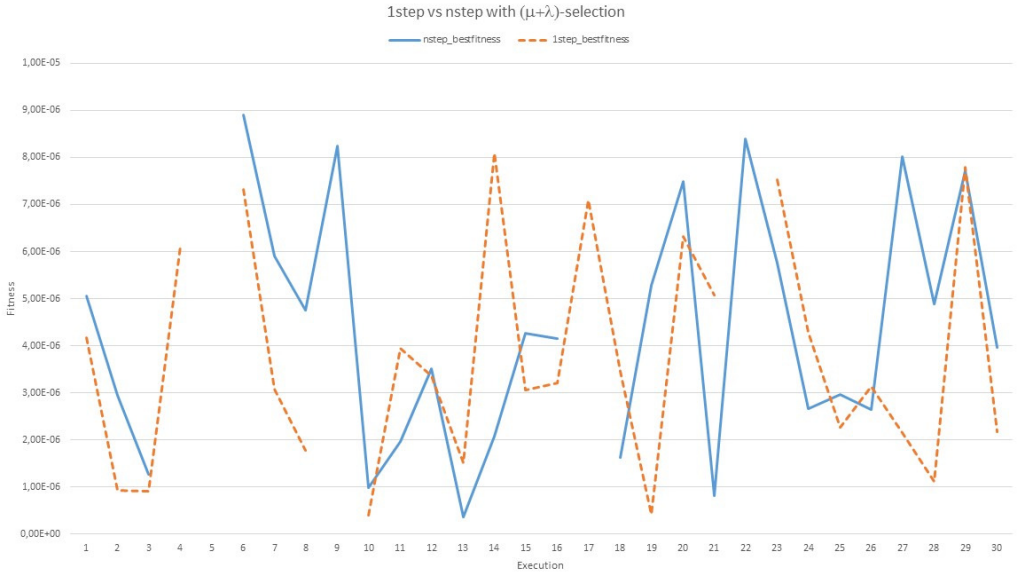

Figure 11: Best fitness solution in each execution with $(\mu + \lambda)$ selection and mutation 1step or nstep for De Jong N5.

Config	Generation Average	Std. Deviation	Fitness Average	Std. Deviation
DeJongConfig1	72,03	213,10	3,8E-06	3,2E-06
DeJongConfig2	109,30	296,59	4,0E-06	2,9E-06
DeJongConfig3	29,97	5,51	3,7E-06	2,4E-06
DeJongConfig4	27,17	7,86	4,3E-06	2,6E-06

Table 5: Statistics of the 30 runs on the 4 configurations for De Jong N5.

Analysis on Selection:

What it is really interesting is how different in terms of speed are those configurations based on selection $(\mu + \lambda)$ with respect to (μ, λ). However they are very similar when the fitness is analyzed (see Figures 12 and 13). In terms of number of generations when the ES stops changes from 72 to 29.97 in average with 1step and from 109.30 to 27.17 with nstep (see Table 5).

General Comments:

At the beginning of the analysis we make some test on the factor applied to the learning rate τ. In that test we conclude that changing that factor it does not provided better results than factor equals one. Maybe it was necessary some more statistic rigor with this but anyway we decided ro continue with facto one. Also it is interesting note that it is possible to initiate the ES with a point near the optimum and it provides a quick converge to the minimum, for example it was tested with $\mu = 30$ and $\lambda = 200$ with $X_0 = (-40, \ldots, -40)$.

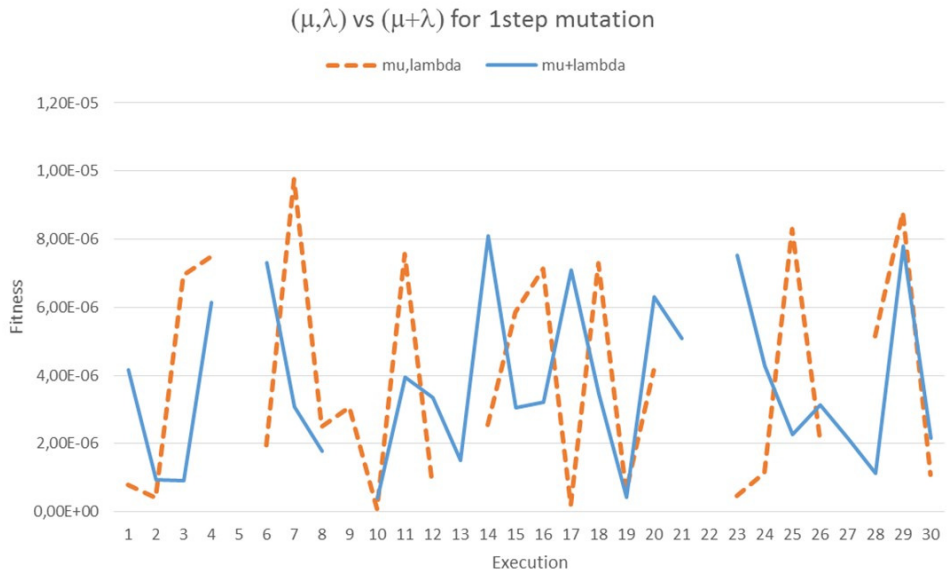

Figure 12: Best fitness solution in each execution with $(\mu + \lambda)$ selection and (μ, λ) selection for mutation 1step for De Jong N5.

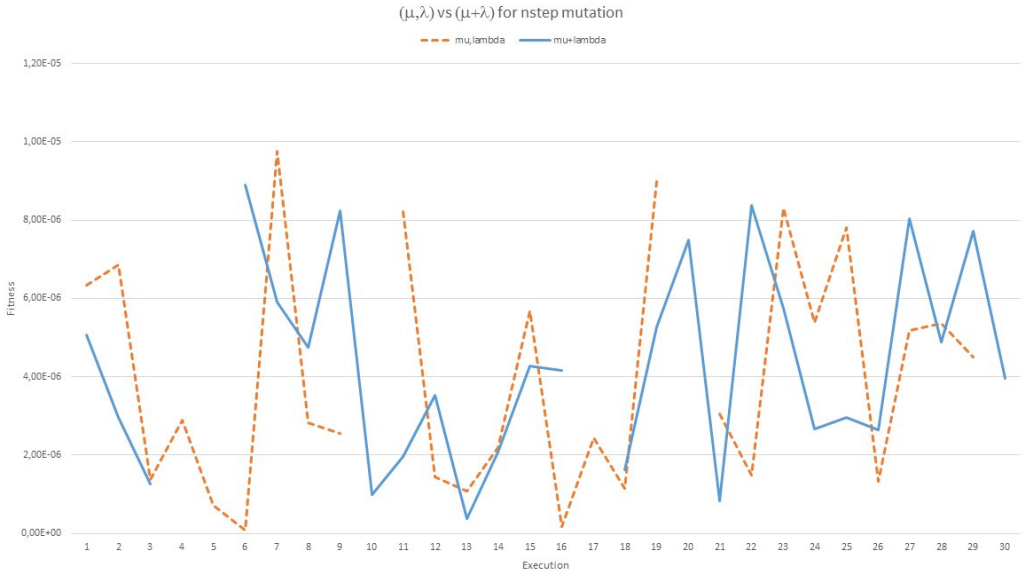

Figure 13: Best fitness solution in each execution with $(\mu + \lambda)$ selection and (μ, λ) selection for mutation nstep for De Jong N5.

6 Schwefel

We are going to use the ES implemented to solve the minimum searching of the function Schwefel. Table 6 summarizes the list of configurations and the name used to denote them. Schwefel functions has the problem of having many local minimums. Trying to avoid the stagnation in one of those minimums we must amply exploration using $\mu = 100$ and $\lambda = 700$. The execution of this ES applied to Schwefel function last much more than the other two function seen until now. Saddly due to deadline it was impossible to have a set of tests to analyze this function. Instead we will review with just one simulation.

ConfigName	mu	lambda	nGen	selection	recomb_var	recomb_str	mut_type
SchwefelConfig1	100	700	1200	mu.lambda	Discrete	NoRecomb	1step
SchwefelConfig2	100	700	1200	mu.lambda	Discrete	Discrete	nstep
SchwefelConfig3	100	700	1200	mu+lambda	Discrete	Discrete	1step
SchwefelConfig4	100	700	1200	mu+lambda	Discrete	Discrete	nstep

Table 6: Configuration definition for Schwefel used in the following analysis.

Analysis on Uncorrelated mutation with (μ, λ)-Selection for different mutation methods (SchwefelConfig1 vs SchwefelConfig2):

Figure 14 shows the progress curves for 1step and nstep mutation for (μ, λ)-Selection. The curve of the 1step is smoother and it does not has ridges as the nstep curve. And also is noticiable that reaches sooner zero.

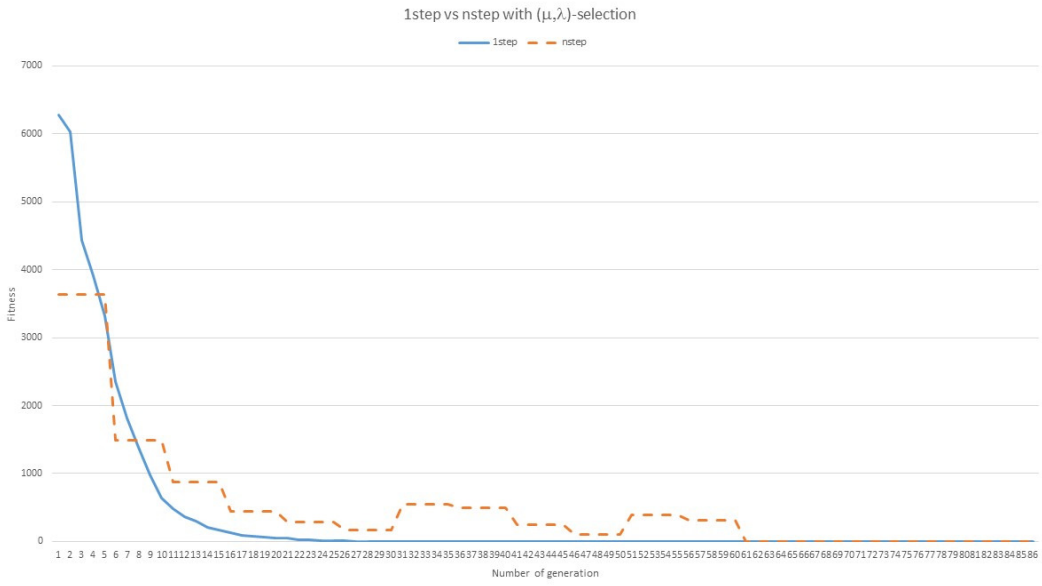

Figure 14: Progress curve with (μ, λ) selection and mutation 1step or nstep for Schwefel.

Analysis on Uncorrelated mutation with $(\mu + \lambda)$-Selection for different mutation methods (SchwefelConfig3 vs SchwefelConfig4):

In the case of having $(\mu + \lambda)$-Selection both 1step and nstep seems very similar in the two curves shown in Figure 15.

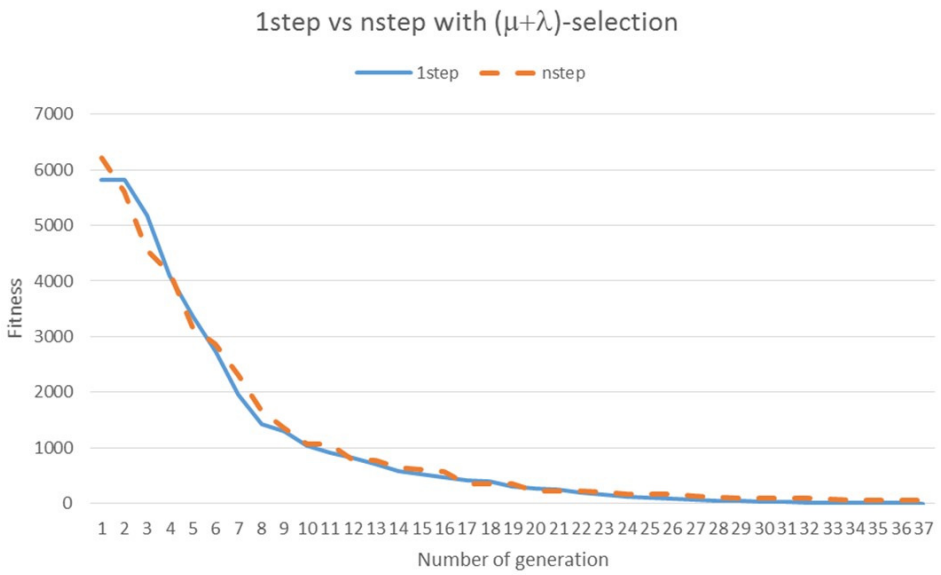

Figure 15: Progress curve with $(\mu + \lambda)$ selection and mutation 1step or nstep for Schwefel.

Analysis on Selection:

If now we analyze the effect of selection method fixing the mutation we can see a faster progress in the (μ, λ) for the 1step mutation (Figure 16) and a smoother progress for $(\mu + \lambda)$ with nstep mutation (Figure 17).

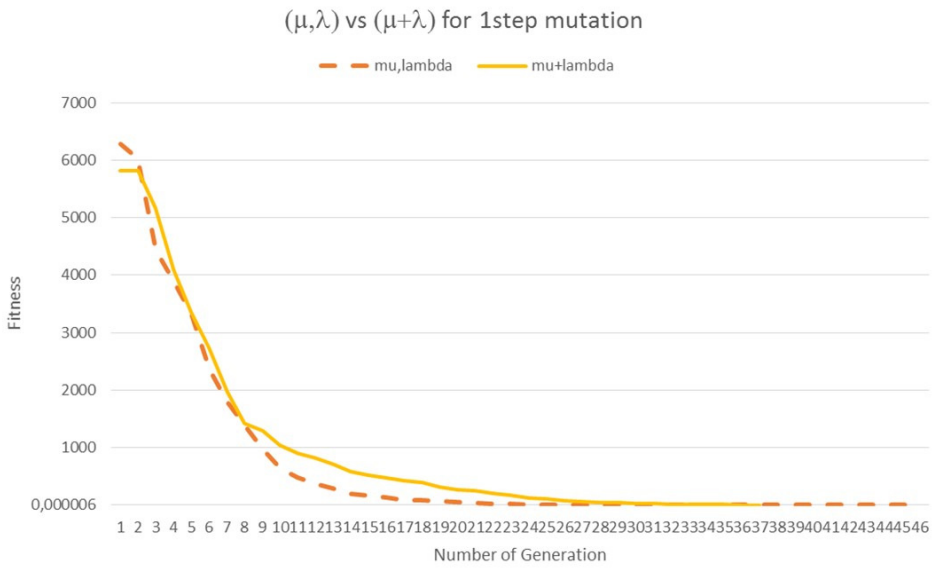

Figure 16: Progress curves with $(\mu+\lambda)$ selection and (μ, λ) selection for mutation 1step for Schwefel.

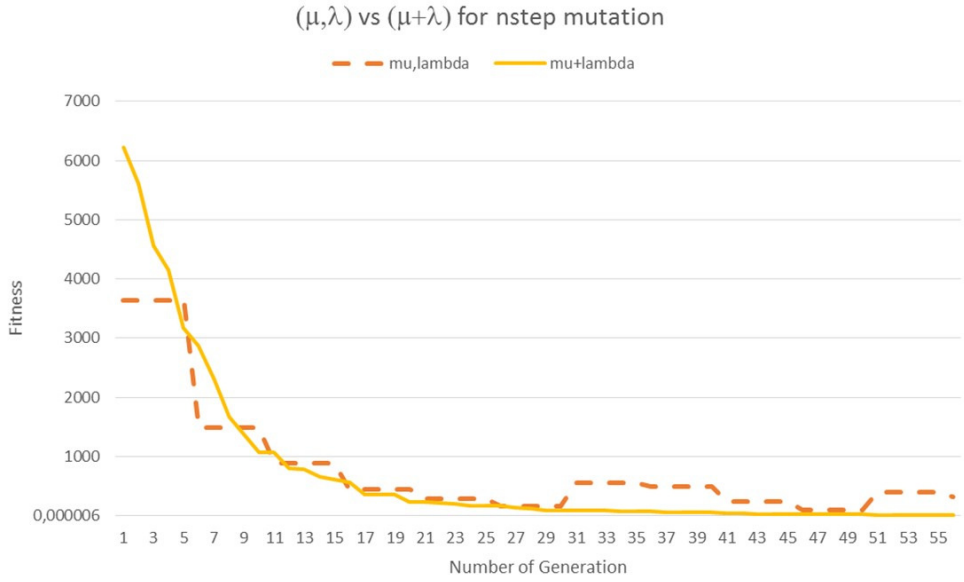

Figure 17: Progress curves with $(\mu + \lambda)$ selection and (μ, λ) selection for mutation nstep for Schwefel.

7 Conclusions

About Sphere:

- nstep uncorrelated mutation is much more slow than 1step. For nstep the maximum number of generations was achieved always.
- SphereCongfig3 is better than the others.

About De Jong N5:

- De Jong function requires a lot of exploration.
- Those $(\mu + \lambda)$-selection configurations outperformed the (μ, λ) configurations in terms of speed but all configurations reach more or less the same fitness.

About Schwefel:

- Mutation type nstep is very hard computationally.
- Author has not time to conclude the analysis (Excuses!) but he can conclude that is better having a good population size and use 1step mutation with (μ, λ) in order to improve the exploration without exhausting time consuming.

References

[1] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies. a comprehensive introduction. *Natural Computing 1: 352, 2002*, 2002.

[2] J.E. Smith A.E. Eiben. *Introduction to Evolutionary Computing*. Springer, 2007.