Evolutionary Robotics
Prof. Dario Floreano

Evolutionary Robotics Laboratory Exercises

Evolutionary Algorithms: part 1

Davide Zappetti (davide.zappetti@epfl.ch**)**
Anand Bhaskaran(anand.bhaskaran@epfl.ch**)**

Goal. The goal of this lab is to understand some advantages and pitfalls of evolutionary algorithms (EAs). In particular, you will observe the effects of the mutation rate, crossover rate, selection pressure, population size and number of generations in artificial evolution, and reflect on how to change these parameters when performing an evolution according to different types of fitness landscapes**.**

Rules:

- **Every exercise poses you some questions on which you should think and elaborate written answers. No official submission of these written answers is required; however, this work is crucial in understanding the basics of evolutionary algorithms and propaedeutic to complete the following Labs, the Robogen project and the final written exam.**
- **Next week, before next class, a PDF with the answers will be uploaded and you will be able to compare them with yours. If you have any question you can discuss with the teaching assistants in class or send an email to us.**

Some information: The labs (TPs) for the Evolutionary robotics course are held in **CM 1 103**. The computers of this lab have dual operating system and all the exercises of this course will be performed in **Linux Operating System**. Login into the system with the following credentials:

Username: Student
Password: EPFLepfl

We will use the Python programming language in the first two lab sessions.

For the first two lab sessions, you will not have to do much programming yourself. However, a basic understanding of Python is recommended. **If you are not familiar with Python, you can find numerous info and tutorials on the** official python website**: or through the web.**

Getting Started. Once you are logged in and comfortable with the computers in CM 1103, you will have to download the file EA_exercises_part1.zip from Moodle and unzip it. Open Visual Studio Code or VSCode, a code editor installs in your computer (you can do this by typing "VSCode" in the search box). Open the File menu and select "Open Folder" then navigate to the downloaded exercise directory.

This lab employs the *inspyred*[1] and other python framework which you should install. To do this, firstly open VSCode Terminal by selecting ***Terminal-> New Terminal*** and execute the following command

```
pip3 install --upgrade –r requirements.txt
```

This should automatically install all the required frameworks for you. To check if everything is fine execute the following command.

```
python3 test.py

>> All good.
```

If you see "**All good.**" Then everything is working properly. However, if you see an error it means something went wrong and you should ask one of TAs for assistance.

Each exercise has a corresponding .py file (named after the exercise number: exercise_1_1.py, exercise_1_2.py, etc.). To solve the exercises, you will have to open, edit, and run these .py files. You can run all the exercises as follows:

```
python3 <<exercise_i_j>>.py
```

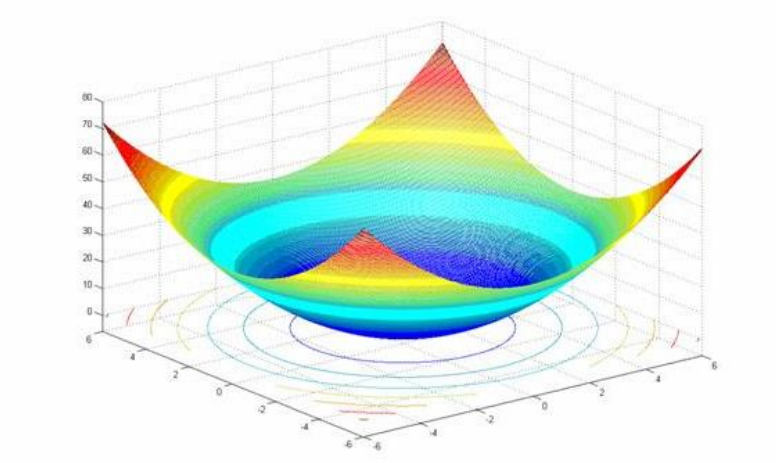
Note. In this lab, the genome of an individual is always a vector of real-valued parameters $\mathbf{x} = [x_1\ x_2 \ldots x_N]$ (keep in mind that there are other types of encodings, e.g. bit strings, which will not be explored in this lab). The fitness of an individual is given by the fitness function $f(\mathbf{x})$. Here, the aim of the EA is to ***minimize*** the function $f(\mathbf{x})$, i.e., to find the vector $\mathbf{x}_{min}$ that has the lowest value $f(\mathbf{x})$. In other words, ***lower* values *f*(x) correspond to a *better* fitness**.

[1] http://pythonhosted.org/inspyred/

Exercise 1

In this first exercise, we will not yet run a complete EA. Instead, we consider a single parent individual $\mathbf{x}_0$, from which a number of offspring individuals are created using a Gaussian mutation operator (which adds a random number from a Gaussian distribution with mean zero and standard deviation σ to each parameter x_i of the parent). The fitness function is defined as:

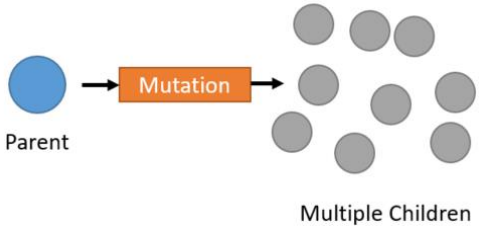
$$f(x) = \sum_{i=1}^{n} x_i^2$$



This fitness function is unimodal, it has a single global minimum at the origin. We will analyze the effects of mutations on the fitness depending on the value of the parent $\mathbf{x}_0$, the mutation magnitude (the standard deviation σ), and the number of dimensions N of the search space.

Exercise 1.1

In this exercise, single parent (at a random location) is mutated to generate multiple offspring. To generate offspring from a single parent $\mathbf{x}_0$ using a Gaussian mutation operator (which adds a random number from a Gaussian distribution with mean zero and standard deviation σ to the parent). Generate offspring from different parents (e.g. $\mathbf{x}_0$=0.1, 1, 10) using different mutation magnitudes (standard deviations σ). First consider the one-dimensional case, then two dimensions, and finally many dimensions (e.g. N =100). For one or two dimensions, the fitness landscape with the parent and the offspring is shown. For more dimensions, a boxplot[2] with the fitness of the offspring is shown where the green, dashed line is the fitness of the parent. As our objective is to find the global minimum, try tuning the parameters such that at least one of the children has near zero fitness. Try to answer the following questions:



[2] Visit http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.boxplot if you are unfamiliar with boxplots

- Do the mutations tend to improve or worsen the fitness of the parent?
- Are low or high mutation magnitudes best for improving the fitness? How does this depend on the initial value of the parent and on the number of dimensions of the search space?

Exercise 1.2

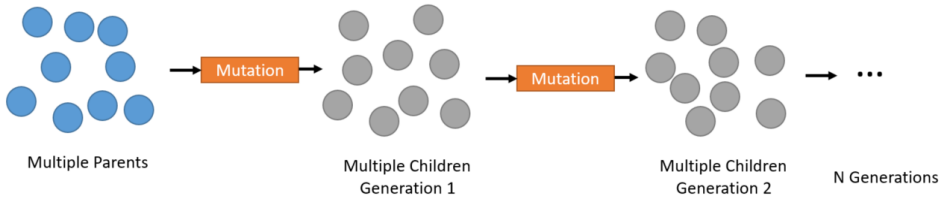
Run the exercise_1_2.py file to confirm the observations that you did qualitatively in the previous exercise by plotting boxplots side-by-side to evaluate the statistical significance of observed differences. Compare different values for:

- The number of dimensions of the search space
- The value of the parent (how close it is to the optimum)
- The mutation magnitude

If you vary one of these three parameters, *make sure that you set the other two at a constant value* (otherwise it may be difficult to interpret your results). Try to confirm the answers that you gave in the previous exercise.

Exercise 2

We will now use an EA to find the minimum of the unimodal fitness function defined in the previous exercise and analyze the effect of the mutation magnitude and the dimensionality of the search space and number of generations on the results.



Exercise 2.1

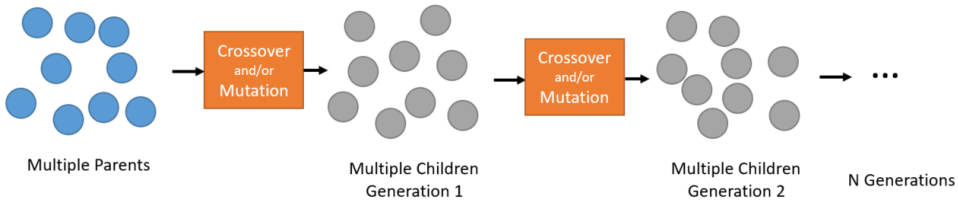
Run exercise_2_1.py to run a basic, mutation-only EA on the one-dimensional function first. How close is the best individual from the global optimum? Increase the dimensionality of the search space to two and more. How close are the best individuals now from the global optimum? Can you get as close as in the one-dimensional case by modifying the mutation magnitude and/or the number of generations?

Exercise 2.2

Run the exercise_2_2.py file to do three batches of 30 runs of the EA with different mutation magnitudes (it may take a minute). The boxplot compares the best fitness values obtained in the three conditions. Try to explain the result.

Exercise 3

In this exercise we will analyze the effect of crossover in the EA. An offspring individual is formed from two parent individuals $\mathbf{x}_1$ and $\mathbf{x}_2$ by randomly taking the value for each entry x_i either from $\mathbf{x}_1$ or $\mathbf{x}_2$. The EA has a parameter defining the fraction of offspring that is created using crossover at each generation (the remaining individuals are created via asexual reproduction).



Exercise 3.1

Using the same setup as in the first two exercises, exercise_3_1.py file does 30 runs using mutation only (as in the previous exercises), and 30 runs using crossover only. The boxplots compare the best fitness values obtained in the two cases. See if you can explain the results.

Exercise 3.2

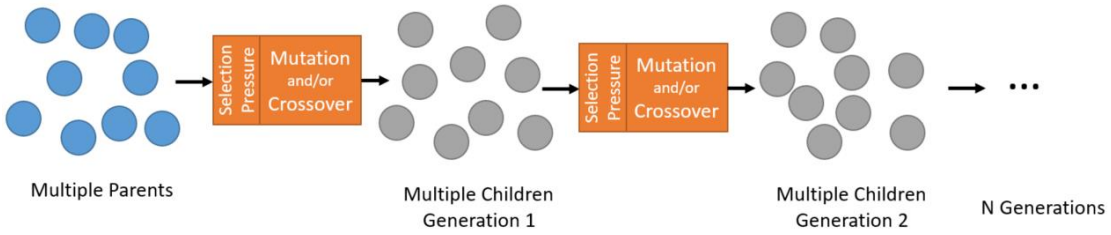
Run exercise_3_2.py to compare the best finesses obtained by varying the fraction of offspring created using crossover (while using a fixed mutation probability of 0.5 i.e. each loci of each genome will have a 50% chance of being mutated).

Is there an optimal crossover fraction for this fitness function? Interpret the results.

Note: You can also set mutation rate to zero in this exercise to realize that effect of crossover only.

Exercise 4

We will now investigate the effect of the selection pressure.

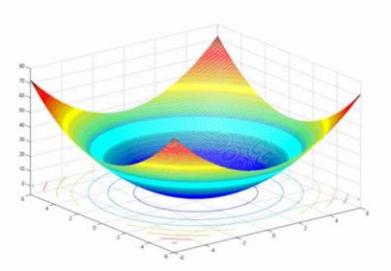


Run the exercise_4.py file to compare the best fitness values and the distances from the global optimum obtained using tournament sizes 2 and 10. **Note:** In the previous exercises, we were using tournament selection with a tournament size of 2.

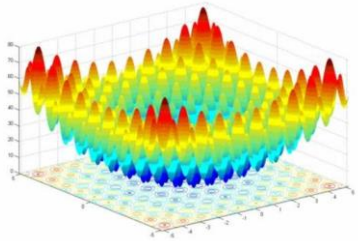
- Which tournament size gives better results for the fitness function sphere and why?
- Which tournament size is better for the fitness function Rastrigin[3] and why?

Exercise 5 : Let's explore!

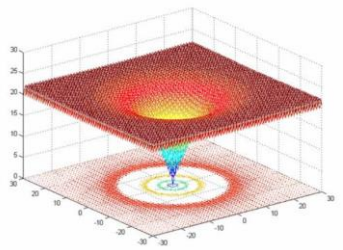
In this exercise you will investigate running the EA on many test functions commonly used to benchmark optimization algorithms. Run the EA on the benchmark functions shown in the below (especially the multimodal functions) and adapt the mutation magnitude, crossover rate, selection pressure, and population size so as to get the best results. If you run the code as provided it will initialize and bound the values of your population vectors to suitable ranges. You may comment/uncomment certain lines to alter this behavior. See the comments in exercise_5.py for further details.



Sphere Function

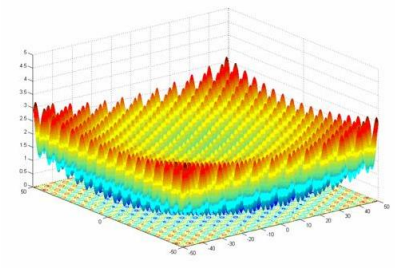


Rastrigin Function

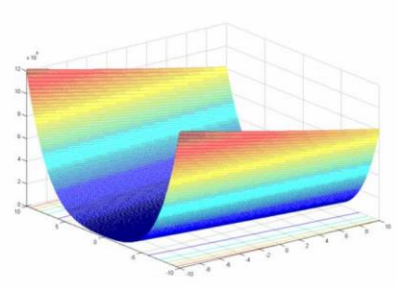


Ackley Function

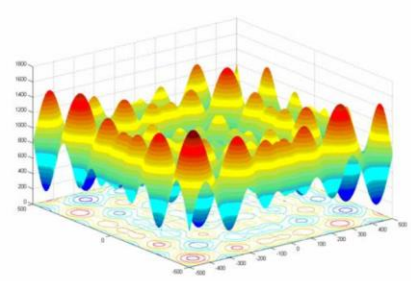
[3] http://pythonhosted.org/inspyred/reference.html#inspyred.benchmarks.Rastrigin



Griewank function



Rosenbrock Function



Schwefel Function

You may first try the 1D or 2D case, which has the advantage that the fitness landscape can be visualized. However, keep in mind that sometimes the resolution of the plot is not sufficient to accurately represent a function.

IF YOU MADE IT HERE IN THE FIRST TP SESSION GOOD JOB, WE WILL CONTINUE WITH THE FOLLOWING EXERCISES NEXT WEEK!