Bio-Inspired Artificial Intelligence

Prof. Giovanni Iacca
giovanni.iacca@unitn.it

Module 3–Lab Exercises

Introduction

Goal. The goal of this lab is to familiarize yourself with some advanced forms of evolutionary computation (EC). In particular, you will explore the use of Evolution Strategies (ES) and Covariance Matrix Adaptation Evolution Strategies (CMA-ES). You will observe the effects of different forms of self-adaptation and how these are useful for black-box optimization.

Getting started. Download the file `03.Exercises.zip` from Moodle and unzip it. This lab continues the use of the *inspyred* framework for the Python programming language seen in the previous labs. If you did not participate in the previous labs, you may want to look those over first and then start this lab's exercises.

Each exercise has a corresponding `.py` file. To solve the exercises, you will have to open, edit, and run these `.py` files.

Note once again that, unless otherwise specified, in this module's exercises we will use real-valued genotypes and that the aim of the algorithms will be to *minimize* the fitness function $f(\mathbf{x})$, i.e. lower values correspond to a better fitness!

Exercise 1

In this exercise you will explore the use of Evolution Strategies (ES). ES are a popular class of evolutionary algorithms used for optimization problems. In particular you will explore the popular $(\mu/\rho,\lambda)$-ES where μ denotes the number of parents, $\rho \leq \mu$ the number of parents involved in the producing a single offspring ("mixing number"), λ the number of offspring, and "comma" selection is employed.

To start the experiments, from a command prompt run[1]:

```
$>python exercise_1.py
```

[1]For all the exercises in this lab you may follow the name of the `.py` file with an integer value, which will serve as the seed for the pseudo-random number generator. This will allow you to reproduce your results. Also, please note that in this document `$>` represents your command prompt, do not re-type these symbols.

This code will attempt to optimize the 10-dimensional Rosenbrock function[2] using an Evolutionary Strategy without self-adaptation or recombination (which is very similar to the GA you used in the first module's exercises).

Try adjusting the various parameters: μ, λ, and ρ, (by changing, respectively, `args[‘‘pop_size’’]`, `args[‘‘num_offspring"]`, and `args[‘‘mixing_number"]` in the script `exercise_1.py`.)

- What happens if you make λ smaller e.g. $\lambda = \mu$?
- What happens if you increase the mixing number ρ?

Try out the different strategy modes (change the parameter `args[‘‘strategy_mode"]`), and observe how they affect the performance of the algorithm:

- `None` means that there is no self-adaptation
- `es.GLOBAL` means each genome encodes a global step-size (mutation standard deviation)
- `es.INDIVIDUAL` means each genome encodes an independent step-size for each gene.

Exercise 2

In this exercise you will systematically explore some of the intuitions you gained in Exercise 1. Open `exercise_2.py`. By default this code is configured to run 10 ES runs apiece with each different strategy mode. The best fitnesses of each run are shown in a boxplot similar to what you saw in the first module's exercises.

- How does the self-adaptation strategy influence performance on this problem?
- Does what you see here confirm what you suspected from the previous exercise?

Use the provided code as a template to systematically explore the other parameters.

Note: To study the effect of parameters, it's important to change only **one parameter at a time**, i.e. keep everything fixed except for the parameter you want to study, and for each parameter under study create box plots to compare different values.

Note: In order to make a fair comparison you must keep the number of function evaluations (i.e., $\lambda \times$ `max_generations`) fixed[3].

- How do the values of μ, ρ, and λ influence the performance given a particular self-adaptation strategy and other parameters?
- Can you come up with any rules of thumb for choosing these parameters?

[2]`http://pythonhosted.org/inspyred/reference.html#inspyred.benchmarks.Rosenbrock`

[3]When you compare multiple algorithms on the same problem, the computational budget allotted to any of them should be same, in order to get a fair comparison. So, if you run experiments with different values of λ, you should adjust `max_generations` accordingly, so that the total number of evaluations is consistent across experiments. E.g.: {$\lambda = 50$, `max_generations`=100}, {$\lambda = 100$, `max_generations`=50}, etc. One rule of thumb sometimes used in evolutionary computation papers and black-box optimization competitions is to run each algorithm for $5000 \times n$ fitness evaluations, where n is the problem dimension. Also bear in mind that some algorithms are not generational at all: in those cases, the stop criterion must be expressed in terms of absolute number of evaluations, or as a convergence condition.

In order to see how general your results are you can explore different benchmark problems[4] on different numbers of variables. You can change the problem by changing the parameter `args[‘‘problem_class’’]` in the script `python exercise_2.py`, and the problem dimension (number of variables) by changing the variable `num_vars` in the script. Note that most benchmark problems are “scalable”, i.e. they can be defined for any number of variables.

- Can you find a choice of parameters that work properly across several problems?

Exercise 3

In this exercise you will explore the use of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). As described in the lecture CMA-ES is currently one of the most successful evolutionary optimizers. By using statistics gathered over generations it is able to adapt a covariance matrix in a completely derandomized way (in contrast to the ES we were just exploring).

While *inspyred* does not include a native implementation of CMA-ES, I have made one available to you through a similar interface as you have been using so far. An example of proper usage is provided in `exercise_3.py`.

Using this and the previous exercise as templates compare the performance of CMA-ES to the other Evolution Strategies you were just investigating.

- Can CMA-ES find optima to different problems with fewer function evaluations?
- How do these differences change with different pop. sizes and problem dimensions?

Note: in CMA-ES ρ must equal μ, hence you cannot define a separate ρ value.

Instructions and questions

Concisely note down your observations from the previous exercises (follow the bullet points) and think about the following questions.

- Do the observations you made while varying μ, ρ, and λ confirm or contradict the conclusions you drew in the previous module’s exercises?
- What are the advantages of self-adaptation in evolutionary computation?
- In what ways might self-adaptation be occurring in biological organisms?
- Compare the different self-adaptation strategies explored in this exercise. In what ways are certain strategies better than others for optimization? In what ways are certain strategies more biologically plausible than others?
- Describe what reasons may contribute to better performance of CMA-ES and what can be the conditions when CMA-ES is not better than a basic ES.

[4] See `https://pythonhosted.org/inspyred/reference.html#single-objective-benchmarks` for a list of single-objective benchmark problems.