IASCA
Programming Evolutionary Algorithms with inspyred

Departamento de Automática, UAH
http://atc1.aut.uah.es

Objectives:

- Introduce `inspyred` programming model
- EAs solving non-trivial problems

Pseudo random numbers generation

EAs, like any other stochastic search algorithms heavily depends on randomness to solve problems. As a consequence, the algorithm needs to generate a large number of random numbers, which is itself a serious problem. Fortunately, Python's default random number generator works well, and is suitable for using in EC. Given that a computer is a deterministic system, getting randomness from it is complex. For this reason random number are not random, but instead pseudorandom.

In order to properly use a pseudorandom numbers generator we need to feed the algorithm with a seed, i.e., a number that is used to initialize the algorithm. The seed should be as much random as possible, so a common practice is to use the current time, in milliseconds. From an EC perspective, it is important to note that two equal seeds will generate the same pseudoranom number sequence, it means that **two executions of an EA with the same seed will give the same results**. It is a very important feature, because eases the repetivity of the execution.

For this reason, any well-implemented EA will contain some code to initialize the pseudorandom numbers generator. In the case of `inspyred`, you will always find something like

```
1  from random import Random
2  from time import time
3
4  # Code here
5
6  # Properly initialize pseudorandom numbers generator
7  prng = Random()
8  prng.seed(time())
9
10 # More code here
```

Getting into inspyred programming API

Inspyred is a well-documented project. Go to the project website[1] and read the following texts:

[1] `http://pythonhosted.org/inspyred/index.html`

- Read the "Overview". Pay special attention to the section "Design Methodology"
- **Do not** read the tutorial (the following exercises are inspired in those examples)
- Examine the examples "Genetic Algorithm" and "Evolution Strategy" within the "Examples" section
- Read carefully the subsection "Customized Algorithms", within the "Examples" section

Once you got a general understanding on the structure of *inspyred*, you should get used to handle its reference manual and source code. Much of the features are not documented, and many times there are doubs that requires reading the source code. Based on the reference library [2], answer the following questions.

1. Identify all the *mutation* methods available by default.
2. Identify all the *recombination* methods available by default.
3. Identify all the *parent selection* methods available by default.
4. Identify all the *survivor selection* methods available by default.
5. Identify all the *termination* methods available by default.
6. Identify all the methods available to inspect the population throughout the evolutionary process.

Playing with inspyred

Based on the code of the one-max problem introduced in the previous practical assigment, perform the following tasks.

1. Change the selection method to fitness proportionale.
2. Change the crossover to uniform crossover.
3. Change the mutation to scramble mutation.
4. Show the basics statistics (best, worst and average fitness) for each generation. Use the proper observer.
5. Store all the individuals in a CSV file for futher analysis. Use the proper observer.
6. Bonus track: Visualize in real-time the basic fitness statistics in a plot. Use the proper observer.

[2] Located in `http://pythonhosted.org/inspyred/reference.html`

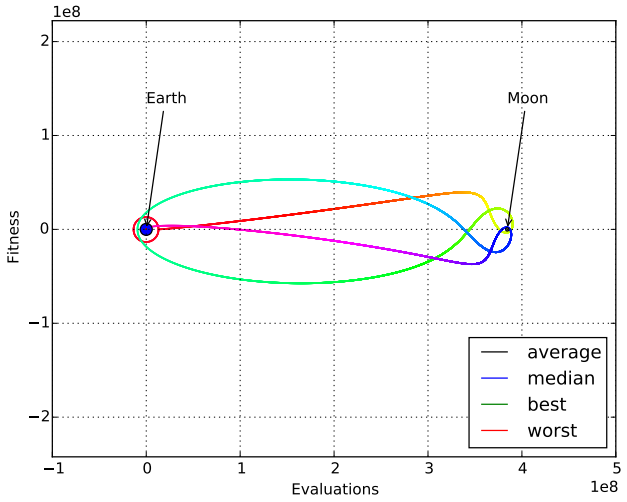

Figura 1: Example of evolved trajectory to the Moon and return.

Genetic operators customization

We are going to use a variant of the one-max problem. Instead of an all-ones chromosomes, we want to generate a chromosome with the pattern $[0, 1, 1, 1]$ repeated k times, for instance, $[0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1]$ with $k = 3$.

Implement a GA to solve the previously described problem. Divide this task into the following steps.

1. Implement an skeleton of the algorithm with *inspyred* with default components.
2. Define and implement the fitness function.
3. Implement a customized crossover operator that only splits chromosomes in the boundaries of the repeated pattern $[0, 1, 1, 1]$. Compare this operator with one-point crossover.

Trajectory of a spacecraft to the Moon

(Based on the problem exposed in the inspyted tutorial, do not read it!)

This exercise deals with an optimization problem of a real-valued function. The objective is to design the trajectory of a spacecraft in orbit on Earth to the Moon using a gravity assist maneuver (see Figure 1). For simplicity the problem is reduced to 2D, and there is only one boost at the begining of the journey. With those considerations, the parameters to optimize are:

- **Orbital height**. The spacescraft begins in a circular orbit over the Earth. This parameter sets its height.

- **Satellite mass**.
- **Boost velocity**. Velocity after the engine boost. It is a bidimensional vector in form (x, y).
- **Initial y velocity**. Initially, the spacecraft moves in the Y axis with the velocity given by this parameter. It does not move in X.

The spacecraft must go to the Moon, transit it as close as possible, and return to land on Earth. It must not crash to the Moon (given by a distance equal to 0). To this end, we have the following fitness function:

$$f = \frac{d_m}{1000} + d + c - l \tag{1}$$

Where d_m is the mimimum distance from Moon, d the total distance traveled, c Moon crash penalty (1000) and l Earth landing reward (1000). Therefore, this is a minimization problem. Execute the following tasks.

1. Download the code that implements the fitness function (`moonshot`) and test it. The Boost velocity must be given in vectorial form (x, y).
2. Design and implement an EA that solves the given problem.
3. Modify the algorithm to consider the following constrains (Hint: Use Bounders)
   - Orbital height $\in (6e6, 8e6)$
   - Satellite mass $\in$ (10,0, 40,0)
   - Boost velocity $(x, y) \in ((3e3, 9e3), (-10000,0, 10000,0))$
   - Initial y velocity $\in (4000, 6000)$

Once you had finished, compare your solution with the one proposed by the tutorial[3].

[3] `http://pythonhosted.org/inspyred/tutorial.html#lunar-explorer`