IASCA
Understanding parameters settings in Evolutionary Algorithms

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

Objectives:

- Introduce `inspyred`
- Understand the parameter settings of a basic GA
- Observe the basic behaviour of an EA
- Customize the parameter settings
- Gather and process the main statistics

Preliminary steps

Install *inspyred* following the instructions available on `http://pythonhosted.org/inspyred/`. Basically, *inspyred* is a Python module that can be installed like any other module. From an Unix console, just type *pip install inspyred*.

One-max problem with Genetic Algorithm

Copy or download the following script, which implements the one-max problem with a basic Genetic Algorithm. All the relevant algorithm parameters are contained in variables defined in the begining of the script.

```
1  from random import Random
2  from time import time
3  import inspyred
4
5  chrLength = 15  # Chromosome length
6  popSize = 50    # Population size
7  maxGenerations = 15     # Max. generations
8  mutRate = 0.1   # Mutation rate
9  elite=0         # Elitism size
10
11 def onemax_fitness(candidates, args):
12         fitness = []
13         for cs in candidates:
14                 fit = sum(cs)
15                 fitness.append(fit)
16         print("Best fit: {0}, avg. fit: {1}".format(max(fitness),
17                 float(sum(fitness))/len(fitness)))
18         return fitness
19
20 def generator(random, args):
```

```
21      return [random.choice([0, 1]) for _ in range(chrLength)]
22
23
24  prng = Random()
25  prng.seed(time())
26
27  ea = inspyred.ec.GA(prng)
28  ea.terminator = inspyred.ec.terminators.generation_termination
29
30  final_pop = ea.evolve(generator=generator,
31                                  evaluator=onemax_fitness,
32                                  pop_size=popSize,
33                                  num_elites=elite,
34                                  max_generations=maxGenerations,
35                                  mutation_rate=mutRate)
36
37  best = max(final_pop)
38  print('Best solution: \n{0}'.format(str(best)))
39  if (best.fitness == chrLength):
40          print("Solution found!")
41  else:
42          print("Solution NOT found")
```

The parameter setting used is the following one:

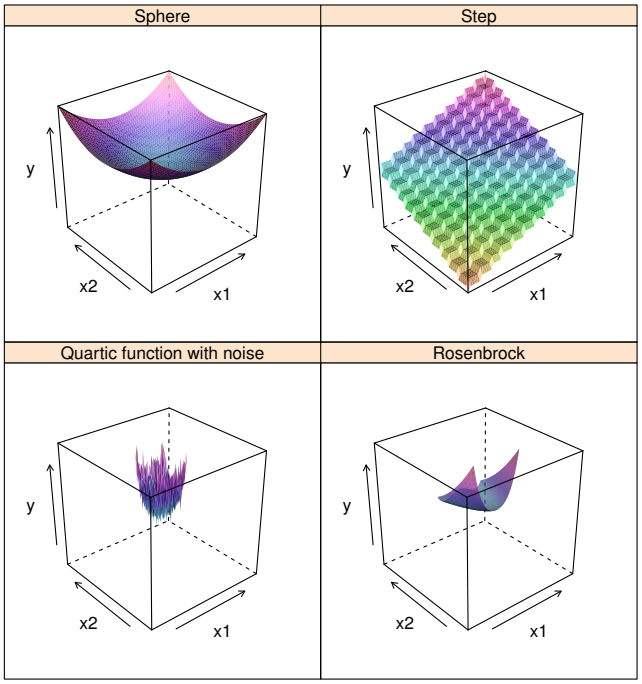
- Representation: Binary
- Chromosome length: 15
- Crossover: 1-point crossover
- Mutation: Flip mutation
- Mutation probability: 0.1
- Population size: 50
- Termination: 15 generations

Perform the following tasks:

1. Execute the script to validate the installation of *inspyred*.
2. Observe how average and best fitness evolve along the time. Explain their behavior.
3. Execute the script several times, did it always find the solution? Why?
4. Change the chromosome length to 30 and repeat the previous questions.
5. Customize the algorithm settings to increase the probability of finding a solution.
6. Set $p_m = 0{,}5$. What happen?
7. Set $p_m = 1{,}0$. What happen?
8. Set the chromosome length to 50 and customize the algorithm to increase the probability of finding a solution.

Optimization problem

Sound comparison of EAs is a thought problem. In order to make this task easier there is a number of theoretical problems[1] that are commonly used to assess the performance of new algorithms. These problems have different properties and, together, offer a base to make comparisons. The next figure shows some of the most common benchmark problems used in EC.

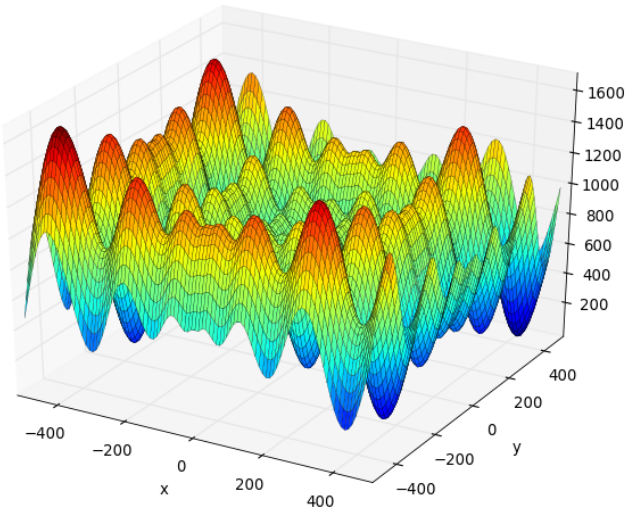


In this exercise we will use one of these, the Schwefel problem, that can be formally stated as follows:

$$f(x) = 418{,}9829n - \sum_{i=1}^{n} \left[-x_i \sin(\sqrt{|x_i|}) \right] \tag{1}$$

where n represents the number of dimensions and $x_i \in [-500, 500] \forall i = 1, ..., n$. The input values that optimizes the function is $[420{,}9687, 420{,}9687, ..., 420{,}9687]$, this a minimaization task and the best fitness is 0. A graphical representation of this problem for $n = 2$ (two dimensions) follows.

[1] You will find several names in the literature, such as benchmark problems or DeJong's functions, however the latter is a fixed collection of 12 functions.



The code that implemens a GA that solves the Schwefel problem is the next listing. Observe that the main algorithm parameters have been deleted.

```
from random import Random
from time import time
import inspyred

# Do not touch this value
maxEvaluations=8000

# Customize these parameters
popSize = X
mutRate = X
elitism = X
tourSize = X
xoverPoints = X

def showStatistics(population, num_generations, num_evaluations, args):
    stats = inspyred.ec.analysis.fitness_statistics(population)
    print('Generation {0}, best fit {1}, avg. fit {2}'.format(
                        num_generations, stats['best'], stats['mean']))

def main(prng=None, display=False):
    if prng is None:
        prng = Random()
        prng.seed(time())

    problem = inspyred.benchmarks.Binary(inspyred.benchmarks.Schwefel(2),
                                         dimension_bits=30)
    ea = inspyred.ec.GA(prng)
    ea.terminator = inspyred.ec.terminators.evaluation_termination
    ea.observer = showStatistics
    ea.selector = inspyred.ec.selectors.tournament_selection
    final_pop = ea.evolve(generator=problem.generator,
                          evaluator=problem.evaluator,
                          pop_size=popSize,
                          maximize=problem.maximize,
                          bounder=problem.bounder,
                          max_evaluations=maxEvaluations,
```

```
37                              num_elites=elitism,
38                                                  tournament_size=tourSize,
39                                                  mutation_rate=mutRate,
40                                                  num_crossover_points=xoverPoints)
41
42      if display:
43          best = max(final_pop)
44          print('Best Solution: \n{0}'.format(str(best)))
45      return ea
46
47  if __name__ == '__main__':
48      main(display=True)
```

The algorithm design is as follows.

- Representation: Binary
- Chromosome length: 30
- Crossover: n-point crossover
- Mutation: Flip mutation
- Mutation probability: X
- Population size: X
- Termination: 8,000 evaluations

Copy or download the code and perform the following task:

1. Set the parameters to get a perfect solution (fitness=0).
2. Set the parameters to get the solution as soon as possible.
3. Execute the algorithm 10 times and show a graph relating generation, best fitness and average fitness. To obtain the graph values, average across all the 10 runs. If necessary, change the code and use any external tool (Excel, Matlab, R, Gnuplot, ...) at your convenience.

Symbolic regression with Genetic Programming (non-inspyred exercise)

Regression is a classical problem in Machine Learning. If you understand the classical lineal regression, you already know what the regression problem is. Given a set of samples drawn from a function, the problem is to find the symbolic representation of that function. The range of applications is huge, financial modelling or whaether forecasting, may serve as examples. A function has a non-linear nature, and is well represented by trees, so, not surprisingly GP is widely use to tackle this kind of problems.

In regression the fitness uses to be the mean squared error (MSE), which is defined as $MSE = \frac{1}{n}\sum i = 1^n(\bar{Y}_i - Y_i)$ where $\bar{Y}_i$ is the observed value and Y_i the theoretical one. So, regression is a minimization problem.

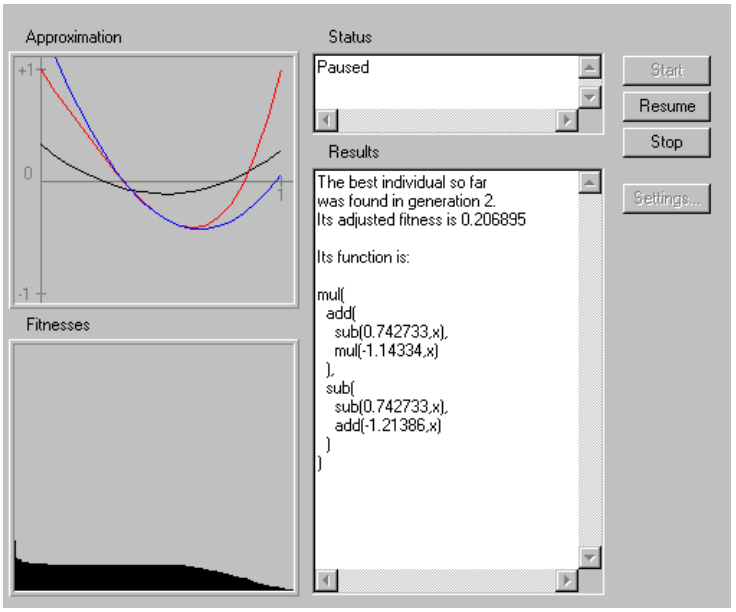


Figura 1: Applet with GP solving a symbolic regression problem.

Unfortunately, *inspyred* does not support GP, so we are going to use an on-line application. Open the site `http://alphard.ethz.ch/gerber/approx/default.html` with a Java enabled browser (i.e. do not use Chrome). If you have problems to execute the applet, open Java settings in your OS and give execution permissions to the applet.

The applet shows the target function (red), the best solution in run (blue) and the tree being assessed (black). It also represents the fitness histogram and a dump of the best tree found in the run. The most common GP parameters can be set with the button (Settings...).

Perform the follwing task:

- Work in groups of two or three people. Compete with your colleagues to get the best approximation (i.e. the lowest fitness). Try with different parameters, including terminal and function sets.