Evolutionary Computation 2013: Practical Assignment

Honours Course

Department of Computer Science
University of Cape Town, South Africa

Problem Description

The Griewank function (Griewank, 1981) has been widely used to test the convergence of optimisation algorithms because its number of minima grows exponentially as its number of dimensions increases. The Griewank function (GR_n) is defined as:

$$GR_n(x) = \sum_{j=1}^{n} x_j^2/4000 - \prod_{j=1}^{n} \cos(x_j/\sqrt{j}) + 1$$

The Griewank function is characterized by.:

- n variables
- Range of initial points: $-N < x_j < N, j = 1, 2, \ldots, n$
- Many local minima
- Global minimum: $x^* = (0, \ldots, 0), GR_n(x^*) = 0$

For example, the function graph for n = 2, $x_j \in$ [-50, 50] is depicted in figure 1, where the global minima: x = [0, 0].

Assignment

Working in pairs, implement a *selection operator* in Java for a *Genetic Algorithm* (GA) that optimises (finds the global minimum) for the Griewank function with the following parameters.:

- n = 100
- $-5000 < x_j < 5000, j = 1, 2, \ldots, 100$

For convenience, the other components of the GA have already been implemented, so only the selection operator needs be implemented for the GA to be tested with the Griewank function (*EC2013.class*). The basic structure of the *MySelection* class is provided as follows. You must implement your own selection operator within the skeleton *MySelection* class.

```
public class MySelection extends Selection
{
 public void initialize(int popsize)
 {
  this.popsize = popsize;
 }

 public Individual[] select(Individual[] population)
 {
  Individual[] newPopulation = new Individual[popsize];

  //  Code for your selection operator ...

  return newPopulation;
 }
}
```

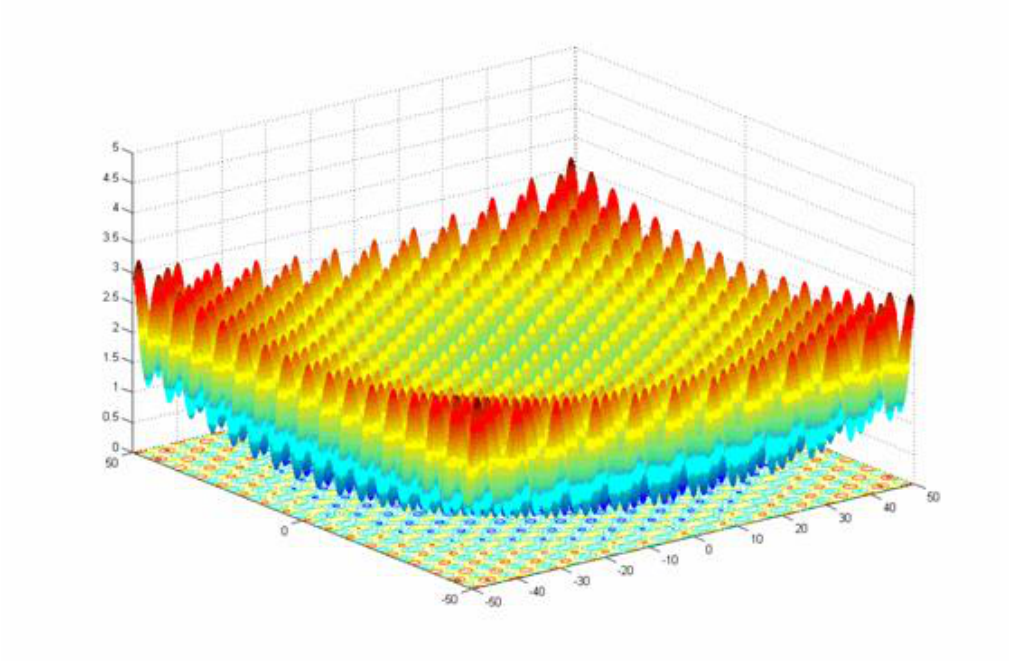

Figure 1: Griewank function graph for n = 2, $x_j \in$ [-50, 50].

In the *MySelection* class, *Individual* is a class that represents each genotype in a population, and *popsize* is the number of genotypes in the population. The *select* method is called from the *Evolve* class. *EC2013Prac.zip* (containing the following Java source and class files) can be downloaded from Vula.

```
EAMath
EC2013
Evolve
Function
FunctionException
Individual
MySelection // Class to include your selection operator
```

Only the *MySelection* class must be modified. *The other source files must not be modified. Submitted source code will be checked and re-run to ensure that this is the case.*

From the command line, the EA is executed (in the *lib* directory) as follows.

```
java Evolve <Fitness Landscape>
            <Population size>
            <Mutation probability>
            <Crossover probability>
            <Number of generations>
            <Selection>
            <Number of runs>
```

Fitness Landscape: Use the *EC2013* class.

Selection: Code a selection operator into the *MySelection* class.

Mutation and *Crossover probability* should be set in the range: $[0.0, 1.0]$, for mutation and crossover to work.

Population size, *Number of generations*, should be set to values that work well with your chosen selection operator.

Number of runs should be high enough in order to draw statistically sound results. That is, if the GA is run N times and yields a low average fitness (the Griewank function must be minimised) with a correspondingly low standard deviation then this a fair indication that N is good enough.

Running Experiments

Choose one of the following GA parameters as an independent variable that you will test *three* (3) different values for :

- *Population size*
- *Mutation probability*
- *Crossover probability*
- *Number of generations*

Alternatively, if applicable, you can choose a parameter associated with your selection operator. For example, if you implement tournament selection you can choose *tournament size* as the independent variable.

Three values of the independent variable must be tested, whilst keeping the other (controlled) parameter values the same. The affect of changing the chosen parameter to each of three chosen values, given your controlled parameter settings is what must be graphed. Such graphs should plot fitness versus generations for a given parameter set.

Parameter set: One of three independent variable parameter settings together with the other (controlled) parameter settings.

For each of the three parameter sets (that is, 3 experiments), N runs (1 run = x generations) must be executed, where N is large enough to ensure statistically sound results.

Controlled parameters should be set to values that work well with the selection operator. These controlled parameter values should be set in test runs, where you get an idea of how changing their values affects the minimisation process.

Submission Requirements

- **Source code** for your selection operator, coded into MySelection.java, together with all other (unchanged code) in $ECPrac/src$. The results files from N runs of each of three parameter sets (3 experiments in total) should be in $ECPrac/lib$. These directories must be placed in a ZIP file and uploaded to Vula. The name of the ZIP file must be: *EC2013-StudentName1StudentName2*.

- **10 Minute presentation**, plus 2-5 minutes for questions. One presentation is to be given per team. Either one or both team members may give the presentation. The presentations will be during the final lectures and should contain:
  - Average best final fitness and standard deviation for each of three experiments (1 experiment = N runs of the GA).
  - A description of the *selection operator* implemented, and why this operator was chosen.
  - The (independent) parameter you chose to test (with three different values).
  - Graphed average best fitness vs generations (over N runs) for each of the three independent parameter values tested.
  - Results of statistical tests that show if there is (or is not) a statistically significant difference between the N runs done for each of the three independent parameter values.
  - A working theory as to why there is (or is not) a statistically significant difference between the results of the three experiments.

  The presentation should also discuss the strategy of the selection approach. For example, if it tries to be exploitative versus explorative, and how appropriate the selection operator is for Griewank function minimisation.

Presentations: Tuesday, March 19, 10.00 - 13.00, 2013

Code submission deadline: Tuesday, March 19, 23.59

References

Griewank, A. (1981). Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34(1):11–39.