**CSC3423 – Biocomputing**

**Practical1: Genetic Algorithms**

The aim of this practical is to familiarise yourself with the working of a basic genetic algorithm using a software library called ECJ, which you have seen as an example in one of the previous lectures.

**Part 1: A basic GA using ECJ**

**Introduction to the framework and the files of the practical**

ECJ (https://cs.gmu.edu/~eclab/projects/ecj/) is a Java-based library for genetic algorithms and genetic programming. It is very comprehensive and has code for all aspects about evolutionary computation that we will cover in the module and many more. It can be used in stand-alone mode, with ECJ being in control of the whole program running, or it has an API that enables it to be integrated into your own code, similarly to what you will have to do in the module’s coursework. An introduction and tutorials on how to use it is available at <https://cs.gmu.edu/~eclab/projects/ecj/docs>. The full 200+ pages manual (with all the gory details) is available at <https://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf>

Download the ‘practical 1 files’ from Blackboard and unzip it. You will see two files: ecj.25.jar and basic-example.params.

ecj.25.jar is a Jar file that contains the whole ECJ library. Basic-example.params is a plain text file that specifies all details of the specific EC algorithm you want to run: what evaluation function, chromosome size, operators, probabilities of cross-over and mutation, etc.

If you open basic-example.params you will see this:

breedthreads = 1

evalthreads = 1

state = ec.simple.SimpleEvolutionState

pop = ec.Population

init = ec.simple.SimpleInitializer

finish = ec.simple.SimpleFinisher

breed = ec.simple.SimpleBreeder

eval = ec.simple.SimpleEvaluator

stat = ec.simple.SimpleStatistics

exch = ec.simple.SimpleExchanger

quit-on-run-complete = true

checkpoint = false

checkpoint-prefix = ec

checkpoint-modulo = 1

pop.subpops = 1

pop.subpop.0 = ec.Subpopulation

pop.subpop.0.duplicate-retries = 0

pop.subpop.0.species = ec.vector.BitVectorSpecies

pop.subpop.0.species.fitness = ec.simple.SimpleFitness

pop.subpop.0.species.ind = ec.vector.BitVectorIndividual

pop.subpop.0.species.crossover-type = one

pop.subpop.0.species.mutation-type = flip

pop.subpop.0.species.pipe = ec.vector.breed.VectorMutationPipeline

pop.subpop.0.species.pipe.source.0 = ec.vector.breed.VectorCrossoverPipeline

eval.problem = ec.app.tutorial1.MaxOnes

generations = 10000

pop.subpop.0.size = 100

pop.subpop.0.species.genome-size = 50

breed.elite.0=0

select.tournament.size = 2

pop.subpop.0.species.mutation-prob = 0.01

seed.0 = 1337

pop.subpop.0.species.pipe.source.0.likelihood = 0.60

pop.subpop.0.species.pipe.source.0.source.0 = ec.select.FitProportionateSelection

pop.subpop.0.species.pipe.source.0.source.1 = ec.select.FitProportionateSelection

#pop.subpop.0.species.pipe.source.0.source.0 = ec.select.TournamentSelection

#pop.subpop.0.species.pipe.source.0.source.1 = ec.select.TournamentSelection

There are lots of lines in this file. For this practical you will only need to modify the lines at the bottom of the file, starting in “generations”. For now, please notice the line “eval.problem = ec.app.tutorial1.MaxOnes”. It specifies that the optimisation problem that ECJ is trying to solve is the same one that has been used in the lectures, the OneMax problem (optimal solution should have all bits set to 1).

**Running the basic example**

To run the basic example, open a command line console, go to the folder where you have these files and type:

java -cp ecj.25.jar ec.Evolve -file basic-example.params

You will see an output like this:

| ECJ

| An evolutionary computation system (version 25)

| By Sean Luke

| Contributors: L. Panait, G. Balan, S. Paus, Z. Skolicki, R. Kicinger,

|               E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley,

|               A. Desai, A. Chircop, J. Compton, W. Haddon, S. Donnelly,

|               B. Jamil, J. Zelibor, E. Kangas, F. Abidi, H. Mooers,

|               J. O'Beirne, L. Manzoni, K. Talukder, S. McKay, J. McDermott

|               J. Zou, A. Rutherford, D. Freelan, E. Wei

| URL: http://cs.gmu.edu/~eclab/projects/ecj/

| Mail: ecj-help@cs.gmu.edu

|       (better: join ECJ-INTEREST at URL above)

| Date: July 1, 2017

| Current Java: 9 / Java HotSpot(TM) 64-Bit Server VM-9+181

| Required Minimum Java: 1.5

Threads:  breed/1 eval/1

Seed: 1337

Job: 0

Setting up

WARNING:

No statistics file specified, printing to stdout at end.

PARAMETER: stat.file

Initializing Generation 0

Generation: 0

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.68

01111001101011110111011110100110111101110101001111

Subpop 0 best fitness of generation Fitness: 0.68

Generation 1    Evaluations So Far 100

Generation: 1

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.7

10000111011111001011111001001001111111110111111111

Subpop 0 best fitness of generation Fitness: 0.7

Generation 2    Evaluations So Far 200

.

.

.

Generation: 9999

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.82

11001111110011111111110110010111111111101111111111

Subpop 0 best fitness of generation Fitness: 0.82

Total Evaluations 1000000

Best Individual of Run:

Subpopulation 0:

Evaluated: T

Fitness: 0.96

11111111111111111111111110111111111111111011111111

Subpop 0 best fitness of run: Fitness: 0.96

As you can see, the genetic algorithm in this example run for 10000 generations and did not even get to find the optimal solution. The reason for that is that the set of parameters that was used is not very good. We will fix that next.

**Part 2: Improving the performance of the Genetic Algorithm**

As mentioned before, the set of parameters in the initial file is not very good. Try to change the following parameters one by one, run the example and see if there is a difference:

1) Selection algorithm

To switch from Roulette-Wheel selection to tournament selection, comment out the lines:

pop.subpop.0.species.pipe.source.0.source.0 = ec.select.FitProportionateSelection

pop.subpop.0.species.pipe.source.0.source.1 = ec.select.FitProportionateSelection

and uncomment the lines:

#pop.subpop.0.species.pipe.source.0.source.0 = ec.select.TournamentSelection

#pop.subpop.0.species.pipe.source.0.source.1 = ec.select.TournamentSelection

Additionally, to change the tournament size, edit the line:

select.tournament.size = 2

2) Elitism

To ensure that the best individual(s) from a generation is copied to the next generation, edit the line:

breed.elite.0=0

Replace the 0 at the end of the line with an integer number specifying the number of best individuals to copy.

3) Generations & population size

To increase the maximum number of generations change the line

generations = 10000

Likewise with the population size:

pop.subpop.0.size = 100

4) Probabilities of crossover and mutation

To change these, edit the lines (for mutation and crossover, respectively):

pop.subpop.0.species.mutation-prob = 0.01

pop.subpop.0.species.pipe.source.0.likelihood = 0.60

5) Change the random seed.

If you don’t change the parameter file, each time you run the code you will obtain the same results. This is because the seed used for the pseudo-random number generator has been fixed. To change it, replace the line:

seed.0 = 1337

to

seed.0 = time

6) Making the problem more difficult

Finally, if rather than changing the parameters of the Genetic Algorithm you would like to change the definition of the OneMax problem, the line:

pop.subpop.0.species.genome-size = 50

specifies the length of the chromosome. That is how many bits are there in the problem. The larger the number, the longer the GA will take (if it can) to fine the optimal.

**Part 3: Thinking ahead for the coursework**

**How to measure performance**

As you will see later when the coursework specification is released, an important aspect to consider is to measure how quick can the GA find the optimal solution. We can see this objective from multiple points of view:

1) Identifying the number of iterations needed to find the optimal solution. This would give us a single number per run as performance measure. If you perform multiple runs with different random seeds, then there would be multiple numbers that would need to be averaged.

2) Rather than just counting number of generations, we can extract the fitness value (performance) of the best individual in each generation, and plot it as a curve. This is what I did in the lectures. Again, if multiple runs are performed, results would need to be averaged across runs.

3) Rather than looking just at the best individual of the population you can look at the average fitness of the population across generations.

**Extracting performance measures**

The output of the algorithm right now is just in the standard output (terminal). You can redirect (part of) the output of the algorithm to a file by adding a line in the config file such as

stat.file = results.txt

Think about how you can extract these performance measures from this file, how you can average results across runs and how you can plot curves (e.g. Excel). As an example, this is how you can extract the best fitness of each iteration from the Linux command line:

cat results.txt | grep "^Generation\|^Fitness" | cut -d: -f2 | grep -v ^$ | sed 'N;s/\n/ /'

This will generate a list of rows with “<iteration> <best fitness>”

**Integrating ECJ with other software**

In the coursework you will need to control ECJ not from the command line but through its API by another piece of Java code. You can see some examples of how this can be achieved in page 51 of <https://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf>