University of Essex
School of Computer Science and Electronic Engineering
CE310 – Evolutionary Computation and Genetic Programming
Summer 2020/21

Coursework reassessment: CE310 - Programming Assignment and mini project

Set by: Reinhold Scherer (r.scherer@essex.ac.uk)

The coursework resit consists of two parts:

1. Part 1 (coding task) asks you to implement basic functionality of Genetic Algorithms in Python, solve a few combinatorial optimisation problems and document your work.
2. Part 2 (research task) asks you to perform computational Genetic Programming experiments in Python and to compile a report that discusses the computed results.

Please complete both tasks and submit a ZIP archive containing the code and documentation for both parts to FASER.

Please contact me by email at r.scherer@essex.ac.uk if you have questions, need further clarifications or if you would like to schedule a Zoom meeting.

1. Part 1 – Implementation of the basic functionality for Evolutionary Algorithms

1.1 >> Assignment Objective

The objective of part 1 is to familiarize yourself with evolutionary algorithms, specifically problem solving using **genetic algorithms** (GAs).

1.2 >> Task definition

Design and implement the core functions needed for **a steady-state binary GA** and solve combinatorial optimisation problems (see below for more details). Software architecture, design and development decisions are entirely up to you. The only requirement is the use of Python (Jupyter/iPython notebook). Select and implement **crossover**, **mutation**, and **selection** operators appropriate for the problems to be solved.

Note that you do not need to install additional libraries. Software libraries included in the standard Python installation are sufficient. You can use *Matplotlib* to visualise results or *NumPy* if you plan to implement more efficient array operations.

1.3 >> Combinatorial optimisation problems

Solve the following problems to test your implementation:

1. **Fifty-Fifty**: Test your code by creating individuals that are represented as 120 bit long arrays or bitstrings (chromosome) and evolve individuals using the following fitness function (pseudocode):

   fitness(individual) = sum(individual[bit 1 to 60] == 0) + sum(individual[bits 61 to 120] == 1)

   Compute different runs and evolve individuals by firstly **maximising** and secondly **minimising** the above fitness function. *What solutions do you expect in both cases?*

   To get a better picture about the GA performance, please experiment with the GA **hyper-parameters** such as **population size**, **generations**, **crossover rate** and **mutation rate.** Also explore hyper-parameters of the **selection operator** that you decided to use. Start with standard values that we discussed in the lecture. *Which parameter combination achieves optimal performance? Why do you think is the parameter combination successful? Are there any drawbacks when using the identified parameter combination?*

2. **Knapsack Problem:** Design and implement binary representation with appropriate fitness function and solve the Knapsack problem. Please test your implementation with the below configuration. Weight and value of the i-th element are located at the i-th position in the respective array. The maximum weight is 50. Run a few simulations. *Did the GA find the optimal solution? Did you have to adjust the hyper-parameters?*

   ```
   weight = [10,4, 7, 4, 5, 5, 5,19,15,10,3,2,18,6,7,4,4,4,4,4,7, 6,7]
   value  = [10,2,30,14,15,16,15,15, 9,10,5,5, 5,5,5,3,3,3,3,3,7,12,5]
   ```

3. **Function optimisation**: Explore the impact of different representations and fitness functions (see below) on the performance of your GA. Remember that optimization involves finding the inputs $x = (x_1, \ldots, x_N)$ to an objective function f(x) that result in the minimum or maximum output of f(x). So, to solve this task you need to define an appropriate representation for x (e.g., binary or real-valued) and then evolve this representation (using the genetic operators you developed in the tasks above) to find the global minimum or maximum of f(x).

   Use the **Hölder table function** (single-objective optimisation problem) and the **Simionescu function** (constrained optimisation problem) and implement them as your fitness functions. You find more details about the functions at https://en.wikipedia.org/wiki/Test_functions_for_optimization. Run simulations to find the minimum of both functions. *Did the GA find the minimum? How precise is the solution? How can you improve the solution?*

4. **Free choice (interesting task for you)**
   You can get even better acquainted with GA by solving additional problems, do more in dept-analysis. For example**:**
   - **Function optimisation**: You can explore multi-objective fitness functions and/or implement a real-valued GA if you used binary Gas to solve the above problems.
   - **Time to move:** Design and implement the representation and the fitness function that solves the combined N Bin Problem and Knapsack problem that we discussed in the first week.
   - **Come up with your own optimisation problem:** Find or come up with your own combinatorial optimisation problem one and solve it.
   - **Impress me with your creativity and exploratory skills**

1.4 >> What must be submitted to FASER?

Submit your Jupyter/iPython notebook including code, results, and a brief description of **solution representation**, **fitness function** and **observations for each problem that you solve** to FASER. Finish the document with a short reflection on GA. Please use a few whole sentences and not just key words for statements. Use the Markdown feature to improve structure and presentation of your code and documentation in the notebook. More information, tutorials and instructions about Markdown for Jupyter notebooks can be found on the Internet.

The Jupyter/iPython notebook is what will be assessed. We will **test the operability of the code** and **assess your documentation, observations on behaviour of GA/hyper-parameter, and answers to the questions.**

Marking criteria (50% of coursework):

- Fifty-Fifty 25%
- Fifty-Fifty hyper-parameter analysis 20%
- Knapsack Problem 15%
- Function optimisation 15%
- Free choice 25%

Note: To get full points as listed above, the code must be executable and documented. Additionally, you must answer the questions about the various problems and make observations about GA performance and hyperparameter settings for each problem.

2. Part 2 - Mini project

2.1 >> Assignment Objective

The objective of part 2 is to familiarize yourself with **genetic programming** (GP) by running several experiments (symbolic regression problems, see the teaching material on Moodle or https://en.wikipedia.org/wiki/Symbolic_regression for more details). While part 1 focused on the implementation of the basic algorithms to gain insights in the underlying mechanisms of evolutionary computation, part 2 focuses on **enhancing** your **research** and **analysis skills**. More precisely, like a **scientist** you will **collect** and **analyse** empirical **data**, **summarise,** and **interpret** your **results,** and based on the **evidence** gathered draw **conclusions**.

2.2 >> Own code or third-party GP toolbox?

If you enjoy coding and want to get a deeper glimpse into GP, then I encourage you to extend the functionality of your Genetic Algorithm code and implement GP for symbolic regression yourself. Alternatively, you can use the third-party DEAP Python GP toolbox to run the experiments. This document will provide you some guidance on how to use the toolbox. Please check the online documentation if you have further questions. **Note, that the choice of code does not impact on the mark**.

DEAP user	The Jupyter notebook file 'CE310-GP-Mini-Project.ipynb' with an example of how to run the experiments can be found in the Assessment Information Tile on Moodle. Here a link to the file https://moodle.essex.ac.uk/mod/resource/view.php?id=797673. For the experiments I recommend using either Anaconda (https://www.anaconda.com/) or Google Colab (https://colab.research.google.com). Both are available for free and you can use them in the labs or on your own computer. Gain a little bit of experience with the system. The Jupyter file has documentation on how to setup the different environments. Please familiarize yourself with the code and start a set of runs (the default parameters are fine) and see what happens. The code generates a log that records data related to *fitness* and *size* of the evolved programs.

2.3 >> Task definition

You are asked to perform a series of GP runs and describe the results of your runs in a report. In your experiments you will need to use GP in different configurations (i.e., problems and parameters). Since GP is a stochastic searcher, you will see that performance varies from run to run. Therefore, to draw your conclusions, you should ensure you perform *at least 10 runs* in each configuration. The aim of the experiments is to get an intuition on how the population and tournament size impacts on *fitness* and *size* of the evolved programs for different symbolic regression problems, and on the *computational complexity* (based on how often the fitness function is being executed). Use the following parameter configurations:

- *Problems:*

  $p_1(x) = x^5 - 13 \cdot x^3 + x - 7$

  $p_2(x) = \sin\left(\frac{\pi}{4} + 3 \cdot x\right) + x$

  $p_3(x) = N(-1.7, 0.5) + 7 \cdot N(1.3, 0.8)$ with $N(\mu, \sigma) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$

  $p_4(x) = \cdots$ create your own function

  Select **two** of the above problems p_i and create symbolic regression data sets generated by associating the 65 x values between approximately -π and +π in steps of π/32 (that is -3.14159, -3.04341, -2.94524.... 3.14159) to 65 corresponding target values t computed via the formula $t=p_i(x)$.

- *Population size:* ***500*** *vs.* ***2000***
- *Tournament size:* ***2*** *vs.* ***5***
- Set the other parameters of the GP to the following values: *generations = 30, crossover rate = 0.7, mutation rate = 0.3*.

DEAP user	The parameters can be adjusted in the PARAMETERS section of the code that you find at the beginning of the third cell in the provided Jupyter file. The section also provides you an example on how to create functions to create target values. Please change and adapt the code as needed and implement the functions.

Note that analysing two problems with two population sizes and two tournament sizes result in eight combinations and consequently you need to run eight different experiments.

Once the runs are completed, analyse the data provided by the system and make sure you look carefully at the output produced in different runs. Is the GA maximising or minimising the fitness? Then **summarize the results in form of a table**, reporting statistics obtained in different configurations when solving your chosen problem. Please select statistics you think are most appropriate to characterize the behaviour of the GP.

Identify the parameter configuration that worked best for each problem and then do a further set of 10 runs with each configuration chosen.

You are now ready to start writing your *report*. In your report explicitly address these questions:

- Did GP show some change in behaviour as the problem, population size and tournament size were changed? Describe what happened.
- Can you provide a tentative explanation for the behaviours you have observed?
- Was the behaviour represented by the statistics (e.g., averages) in one configuration typical of all runs in that configuration or did you see ample variations in behaviour across the runs done with a configuration? If there were variations, can you explain why these happen?
- Explain what criteria you used to identify the best parameter configuration for the two problems.
- Were the statistics you got in the extra 10 runs performed with your chosen optimal configuration for each problem consistent with the statistics you obtained in the first batch of runs? If not, what do you think happened and how can we find out what's the best configuration?
- What conclusions can you draw about the suitability of GP to solve the given problems?
- Feel free to produce and report additional plots or tables (possibly including one or two typical runs) if this can provide support for your explanations.

This is a research focused assignment. This means you are encouraged to do further analyses and generate additional evidence that supports your conclusions. This will allow you to create a strong and conclusive report

2.5 >> What must be submitted to FASER?

Submit either a Jupyter/iPython notebook including the code, results, and the report or a PDF file of the report to FASER. Please be **short** and **concise**, and do **not exceed 500 words** (without Table or Table/Figure captions).

The Jupyter/iPython notebook or PDF (if applicable) is what will be assessed.

Marking criteria (50% of coursework):

- Structure and presentation of the report (including division into sections, provision of a table of contents, formatting, diagrams, etc.) 10%
- Accurate description of GP's behaviour in the runs performed 20%
- Depth and correctness of the analysis of the runs and plausibility of the explanations provided 30%
- Concluding section, including summary of what has and hasn't been learnt and possible further explorations 20%
- Supplementary analyses and further experiments 20%

3. Other information

3.1 >> Problems

For specific inquiries, please email r.scherer@essex.ac.uk

3.2 >> Late Submission and Plagiarism

Please refer to the Students' Handbook for details of the Departmental policy regarding submission and University regulations regarding plagiarism.