University of Essex
School of Computer Science and Electronic Engineering
CE310 – Evolutionary Computation and Genetic Programming
Spring 2023

Coursework Assignment: CE310 - Programming Assignment and Mini Project

Set by: Riccardo Poli (rpoli@essex.ac.uk)

The coursework 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 then document and discuss your work.
2. Part 2 (research task) asks you to perform computational Genetic Programming experiments in Python and to produce a report that discusses the results obtained.

Each part accounts for 50% of the coursework.

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 rpoli@essex.ac.uk if you have questions.

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 use the implemented GA to 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 Jupyter Notebooks (Python). Select and implement **crossover**, **mutation**, and **selection** operators appropriate for the problems to be solved. Select appropriate standard values for the hyper-parameters (population size, generations, crossover rate, etc.).

Marking criteria: Code and code documentation [25]

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.

Not sure how to start? Little or no experience with Python?
Note that in the lecture we will implement a basic framework for the GA method together. Please watch the associated video and download the Jupyter notebook from Moodle as soon as it is available. You can use the provided code as a basis for your implementation.

1.3 >> Combinatorial optimisation problems

Solve the problems below to test your implementation. Note that to get full marks the code must be executable and either documented via comments (or Markup sections) or self-documenting (clean code, extreme-programming style, for a concise summary see here). Additionally, you must make observations about GA performance and hyper-parameters, answer questions about the various problems and make observations about GA performance and hyperparameter settings for each problem. *Observations to be made/questions to be answered are written in blue.*):

1. **ASCII art**:
   Test your code by using a binary array or bitstring representation (chromosome) and evolve the representation to match a given ASCII art image (e.g., Fig. 1). The ASCII art is given as a text file that contains the symbols '0', '1', and '\n' (new line). You find an example on Moodle, but you can create your own. Each line in the text file represents a row of pixels. Multiple lines of 0s and 1s need to be concatenated to form a complete binary representation (genotype) of the 2-D ASCII art matrix (phenotype). The opposite operation is needed to convert genotypes evolved by the GA into phenotypes. To evolve solutions, use the fitness function (pseudocode)

   ```
   fitness(ind) = sum over i of (ind[i] == ASCII[i]),
   ```

   where `ind` is an individual to be evaluated, `ASCII` is the target binary pattern we want to evolve and i is the i^{th} gene (bit) in the individual.

   ```
   0001111000001100
   0010000100010110
   0100000000100011
   0010011100111111
   0001110100100001
   ```

   Fig. 1. ASCII art image example. The 1s form the letters GA.

   Marking criteria: Code and code documentation [4]

Perform different GA runs, i.e., execute your code several times, and evolve individuals by firstly **maximising** and secondly **minimising** the above fitness function. *What solutions do you expect when maximising and minimising the fitness function?*

Observations [1]

2. **Hyper-parameter Analysis**:
   To get a better picture about the GA performance, experiment with the GA **hyper-parameters** such as **population size**, **generations**, **crossover rate** and **mutation rate.** Also explore hyper-parameters (if any) of the **selection operator** that you decided to use. Start with standard values that we discussed in the lectures.

   Code and code documentation [6]

   Briefly describe and discuss the behaviour of the GA based on the selected hyper-parameters.
   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?

   Observations and discussion [14]

3. **Fitness function**:
   Every problem has its own fitness function. Finding suitable fitness functions can be a challenge. The GA's task is to find the minimum or maximum of the fitness function. In general, we do not know the structure (e.g., the shape of the graph) of the fitness function, i.e., we do not know how many local or global minima and maxima a curve has. In this task we explore the impact of different representations and fitness functions (see below) on the performance of your GA. Remember that optimization involves finding the inputs $x = (x_1, \dots, x_N)$ to a fitness (or more generally objective) function f(x) that result in the minimum or maximum output of f(x). To solve this task, you need to define an appropriate representation for x_i (e.g., binary, real-valued or mixed) and then evolve this representation (by adapting or creating binary or real-valued genetic operators as needed) to find the global minimum or maximum of f(x).

   Code and code documentation [10]

   Select **one** single-objective and **one** constrained optimisation function from https://en.wikipedia.org/wiki/Test_functions_for_optimization and implement them as your fitness functions. Then run simulations to find the minimum (or maximum, depending on the problem) of the function.

   Briefly describe and discuss the behaviour of the GA.
   Which representation did you use?
   Did the GA find the minimum or maximum?
   Did you have to adjust the hyper-parameters?
   How precise is the solution?
   How can you improve the solution?

   Observations and discussion [10]

4. **Free choice (interesting task for you):**
   Come up with your own combinatorial optimisation problem and solve it using a GA. Be creative and impress the module supervisor

   Code and code documentation [20]

   Briefly describe the problem, the representation, and the fitness function, and discuss the behaviour of the GA.

   Observations and discussion [5]

5. **Self-assessment:**
   Discuss and motivate how your work in Tasks 1-4 meets the marking criteria.

   Discussion [5]

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

1.4 >> What must be submitted to FASER?

Submit your Jupyter/iPython notebook including code, results, and observations and discussions to FASER.

Please 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. Structure the notebook in such a way that the different tasks can be clearly distinguished.

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.**

1.5 >> Problems

If any general problems arise in the assignment, please start a thread on the Moodle forum. For specific inquiries, please use the academic support hour or email me at r.scherer@essex.ac.uk

1.6 >> Late Submission and Plagiarism

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

1.7 >> How to start?

This depends on how familiar you are with algorithmic thinking and on your coding skills. A good way to start is to make sure that the task is clear. This includes reading the whole assignment and making sure you are familiar with the content discussed in units 1-3.

Check the learning material on Moodle, use resources available at the university library or use internet search engines to find relevant material. Of course, asking question during the lectures and classes is another good way to get answers and have a good discussion. You can assume that if you struggle, your fellow students will also struggle. So, I encourage you to ask questions and engage in constructive conversations.

Once it is clear what you need to do start with the planning. You can start top-down, i.e., plan the development by staring from the general Evolutionary Algorithm and then implement the different functions, or bottom-up, i.e., implement basic functionality and then combine everything. Be systematic and add functionality step-by-step. Also, be sure to define the way you want to represent binary arrays/string before implementing the genetic operators.

Keep in mind that there are many ways how to address and complete the tasks. There is no correct or best way to do this. Everything depends on your skills and the decision you make. So, make decisions based on objective evidence and not on subjective feelings. This is a learning opportunity and learning means making errors and correct them.

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 understanding of GP, then you can modify the representation and crossover/mutation operators of your Genetic Algorithm code and implement a GP system for symbolic regression yourself. Alternatively, you can use an implementation of GP that I will provide shortly. Finally, you can use the third-party DEAP Python GP toolbox to run the experiments. Below I provide 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. For the experiments you could Jupyter-lab. Otherwise Anaconda (https://www.anaconda.com/) or Google Colab (https://colab.research.google.com) that 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:*

  $p_1(x) = 5 \cdot x^5 + 4 \cdot x^4 + 3 \cdot x^3 + 2 \cdot x^2 + x$

  $p_2(x) = 4 \cdot \sin\left(\frac{5 \cdot \pi}{4} \cdot x\right)$

  $p_3(x) = G(-1.7, 0.5) + 7 \cdot G(1.3, 0.8)$ where $G(\mu, \sigma)$ is a gaussian: $G(\mu, \sigma) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$

  $p_4(x) = \cdots$ 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 GP 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 600 words** (without Table or Table/Figure captions).

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

Marking criteria (50% of coursework):

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

2.5 >> Problems

If any general problems arise in the assignment, please start a thread on the Moodle forum. For specific inquiries, please email me at rpoli@essex.ac.uk

2.6 >> Late Submission and Plagiarism

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