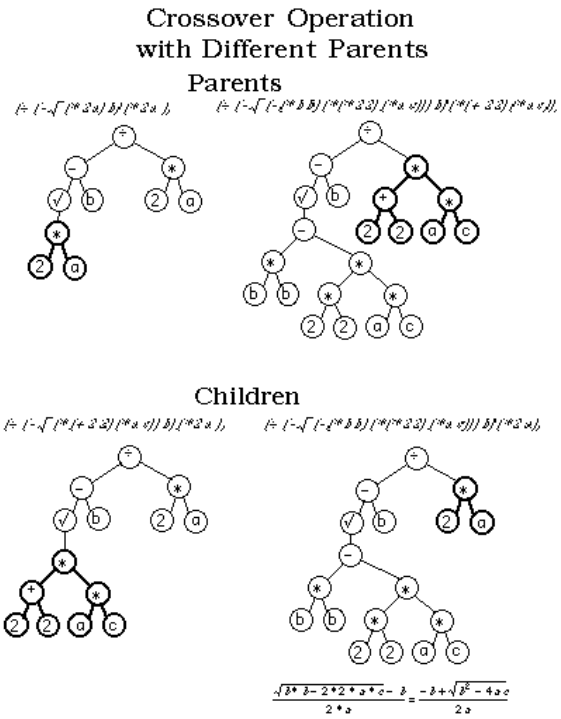
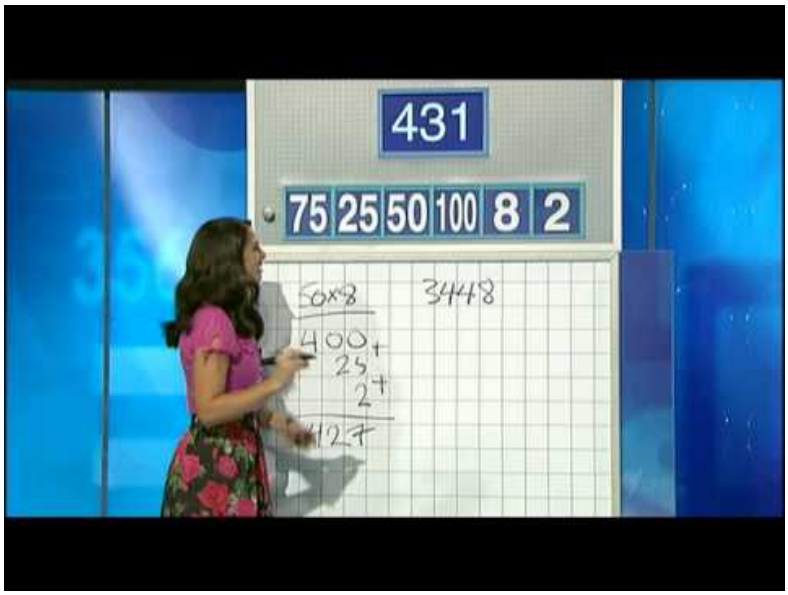
Last updated on Friday, April 30, 2021

Letters and Numbers Assignment

Genetic Algorithms





Key information

- Submission due at the end of **Week 13** (Sunday 06 June, 23.59pm)
- Submit your work via Blackboard
- Recommended group size: three people per submission.
  Smaller groups are allowed (1 or 2 people OK, but completion of the same tasks is required).

Overview

In the Australian TV show called “Letters and Numbers”, contestants played two games: one where they got a random collection of letters and had to create the longest word they can, and one where they got a random collection of numbers and a random target and had to create a calculation that produced the target. They did this in a 30 second timeframe, counted down by a giant clock on the wall. The Australian show was based on a show which in the UK is called “Countdown” in reference to the clock, which in turn is inspired by a French show called “Des chiffres et des lettres”.

The aim of this assignment is to design and implement a player for the number game using Genetic Algorithms (GA)

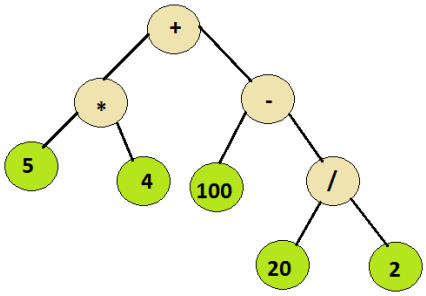
Problem formulation

The player will evolve *expression trees* to find solutions to instances of the problem played in the *Letters and Numbers* game show (https://en.wikipedia.org/wiki/Letters_and_Numbers).

Here is how this game is played (extract from a wikepedia description): "*one contestant chooses how many "small" and "large" numbers they would like to make up six randomly chosen numbers. Small numbers are between 1 and 10 inclusive, and large numbers are 25, 50, 75, or 100. All large numbers will be different, so at most four large numbers may be chosen. The contestants have to use arithmetic on some or all of those numbers to get as close as possible to a randomly generated three-digit target number within the thirty second time limit. Fractions are not allowed—only integers may be used at any stage of the calculation.*"

We can represent a candidate solution with an expression tree.

The expression tree on the right corresponds to the computation "(5*4)+(100-(20/2))". In the game, we are limited to the three operators "+, *, -".



Another useful representation derived from the prefix notation (aka Polish notation) of the expression is the nested lists representation: " [+, [*, 5, 4] ,[-, 100, [/, 20, 2]]] ".

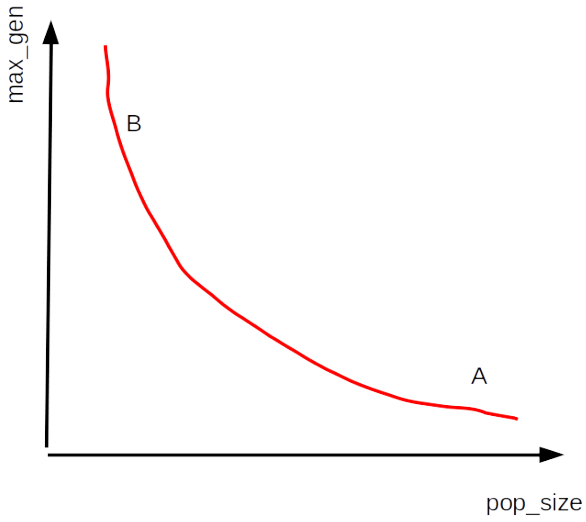
For the purists, the brackets are not needed in the Polish notation, but I added them for readability.

Using GA, you will evolve a population of expression trees whose fitness will be defined by the distance between the target and the values computed by the expression trees.

Given a computational budget, say two seconds, we have some discretion with respect to the values of the parameters *population size* and *maximum number of generations.*

You are asked to investigate what is a good trade-off between these two parameters.

In the diagram on the right, we have sketched in red the combinations of the parameters for a budget of two seconds. It is not obvious what is better; a large population size with a small number of generations (Option A in the diagram), or a smaller population with a larger number of generations (Option B in the diagram). You are asked to perform some experiments to answer this question.



Approach

Our approach is a standard GA that follows the pseudo-code below.

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

You are provided with some scaffolding code:

- **genetic_algorithm.py** contains a function *evolve_pop* implementing the above pseudo-code
- **run_experiment.py** shows how to call *evolve_pop*
- **tester.py** snippet to call the crossover function
- **number_game.py** this is the file that contains the functions that you have to implement

Your tasks

Your solution **has to comply to** the programming interface defined in the file ***number_game.py.***

All your code should be located in a the file called ***number_game.py***. **This is the only Python file that you should submit**. In this file, you will find partially completed functions and their specifications. You can add auxiliary classes and functions to this file. If you break the API, your code will fail the tests!

Task 1

Complete the following functions defined in ***number_game.py***.

- polish_str_2_expr_tree
- decompose
- mutate_num(T, Q)
- mutate_op
- cross_over

Task 2

Given the constraint of a budget of 2 seconds on your computer, investigate what is the best combination of values for the parameters *population size* and *maximum number of generations.*

Hints: For 20 different values of *population size* in an appropriate range, compute what is the corresponding *maximum number of generations* that can be computed within the time budget of 2 seconds. For each of the 20 pairs, gather about 30 game results to evaluate the success rate of the pair. For Task 2, you will need to decide what extra functions to write.

Deliverables

You should submit via Blackboard only two files

1. A **report** in **pdf** format **strictly limited to 4 pages in total** (be concise!) containing
    - a description of your investigation with results clearly presented in tables and figures.
    - a recommendation for the values of the investigated parameters
2. Your version of the **Python file *number_game.py***

    The file should include the functions required to repeat your experiments. It should be no more complicated than uncommenting a function call in a main block.

Marking Guide

- **Report**: 30 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
- **Code quality**: 20 marks
  - Readability, meaningful variable names.
  - Proper use of Numpy and Python idioms like dictionaries and list comprehension.
  - No unnecessary use of loops when array operators are available.
  - Header comments in classes and functions. In-line comments.
  - Function parameter documentation.

- **Functions of *number_game.py* :** 50 marks

  The markers will run python scripts to test your functions.
  - my_team(): 1 mark
  - polish_str_2_expr_tree: 4 marks
  - decompose: 25 marks
  - overall performance: 20 marks

Marking criteria

- **Report**: 30 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).

Levels of Achievement

30 Marks	**24 Marks**	**18 Marks**	**12 Marks**	**1 Mark**
+Report written at the highest professional standard with respect to spelling, grammar, formatting, structure, and language terminology.	+Report is very-well written and understandable throughout, with only a few insignificant presentation errors. +Recommendation derived from experimental results are clear	+The report is generally well-written and understandable but with a few small presentation errors that make one of two points unclear. +Clear figures and tables.	The report is readable but parts of the report are poorly-written, making some parts difficult to understand. +Use of sections with proper section titles.	The entire report is poorly-written and/or incomplete. +**The report is in pdf format.**

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all $j \leq i$. For example, if your report is not in pdf format, you will not be awarded more than 1 mark.

Levels of Achievement

[15-20] Marks	[10-15] Marks	[7-9] Marks	[4-6] Marks	[1-3] Mark
+Code is generic, well structured and easy to follow. For example, auxiliary functions help increase the clarity of the code.	+Proper use of data-structures. +No unnecessary loops. +Useful in-line comments. +Header comments are clear. The new functions can be unambiguously implemented by simply looking at their header comments.	+No magic numbers (that is, all numerical constants have been assigned to variables with meaningful names). +Each function parameter documented (including type and shape of parameters) +return values clearly documented	+Header comments for all new classes and functions. +Appropriate use of auxiliary functions. +Evidence of testing with assert statements or equivalents	Code is partially functional but gives headaches to the markers.

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all $j \leq i$.

Miscellaneous Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions or use MS Teams. If you don't get a satisfactory answer that way, email me f.maire@qut.edu.au
- Enjoy the assignment!
- Don't forget to **list all the members of your group in the report and the code**!
- Only one person in your group should submit the assignment. Feedback via Blackbloard will be given to this person. This person is expected to share the feedback with the other members of the group.

How many submissions can I make?

- You can make multiple submissions. Only the last one will be marked.

How do I find team-mates?

- Use Blackboard groups. Note that the groups are only used to facilitate group formation.
- When marking the assignment, we simply look at the names that appear in the report and in the code. We ignore the Blackboard groups.
- Use the unit MS Teams. If you are not registered, please contact HiQ. You should have been automatically added because of your enrollment in the unit.
- Make sure you discuss early workload with your team-mates. It is not uncommon to see groups starting late or not communicating regularly and eventually submitting separately bits of work.