Deliverable 1

Double Tournament

What is Bloat in Genetic Programming?

“The search space of GP is virtually unlimited and programs tend to grow in size during the evolutionary process. Code growth is a healthy result of genetic operators in search of better solutions, but it also permits the appearance of pieces of redundant code that increase the size of programs without improving their fitness. Besides consuming precious time in an already computationally intensive process, redundant code may start growing rapidly, a phenomenon known as bloat. Bloat can be defined as an excess of code growth without a corresponding improvement in fitness. This is a serious problem in GP, often leading to the stagnation of the evolutionary process. Although many bloat control methods have been proposed so far, a definitive solution is yet to be found.”

- Sara Silva “Controlling Bloat: Individual and Population Based Approaches in Genetic Programming”, 2008

Double Tournament

In this first deliverable, you will implement one very simple bloat control method called Double Tournament. While the regular tournament selection only evaluates individuals based on their fitness, Double Tournament introduces a second round where individuals are evaluated based on their size.

The complete description can be found in *Sean luke et. al “Fighting Bloat With Nonparametric Parsimony Pressure” 2002 :*

“The double tournament algorithm selects an individual using tournament selection: however the tournament contestants are not chosen at random with replacement from the population. Instead, they were each the winners of another tournament selection. For example, imagine if the “final” tournament has a pool size of 7: then seven “qualifier” tournaments are held as normal in tournament selection, and the winners go on to compete in the “final” tournament. Double tournament has been previously used to select for both fitness and diversity . We suggest it may be used for parsimony pressure* by having the “final” tournament select based on parsimony while the qualifying tournaments select based on fitness (or vice versa). The algorithm has three parameters: a fitness tournament size S_f, a parsimony tournament size S_p, and a **switch** (do-fitness-first) which indicates whether the qualifiers select on fitness and the final selects on size, or (if false) the other way around.”

Summarizing, S_f is the tournament size, referring to how many single tournaments need to be performed. By the above example, if $S_f = 7$ then you perform 7 independent tournaments and store the winners.

S_p refers to the parsimony (size) tournament. It refers to the number if individuals which will be chosen for the second round of the tournament, where they are evaluated by their size. If $S_p = 3$ then out of the 7 winners from the first round that we have stored, 3 will be randomly chosen to compete and the smallest one will be the final selected parent. There is only one requisite which is $S_p \leq S_f$.

Lastly, there is the **switch** boolean. By default, it should be False, meaning that the order should be fitness → size, but if it is True, then the order should be reversed, going size → fitness, and the requisite changes to $S_f \leq S_p$

Instructions

Using the StdGP framework, make a new function called **double_tournament** which performs the Double tournament selection described above.
In the *GeneticOperators*** file of the framework, you already have the regular tournament implemented. You can make use of that as you want, as well as use any other function that is already implemented in the framework.

All you need to deliver code-wise is a new *GeneticOperators.py* file with the double tournament implemented and a modified *StdGP.py* file that uses the Double Tournament selection instead of the regular Tournament selection.
Additionally, you need to include a very short report (1-page **maximum**) explaining the logic behind the code that was written.

The projects need to be submitted on moodle by the end of the 11th of May (23:59).

*Parsimony pressure - a means to control code growth by penalizing larger solutions.

GeneticOperators - Selection methods are **not genetic operators, the fact that they are included in this file is purely a development choice by the author.