Biologically Inspired Computation - Coursework 2

Introduction

The aim of this coursework is to highlight how changes in parameter values within Genetic Programming (**GP**) can affect the overall fitness and solutions obtained by the method. Various tests were performed using the Java based evolutionary computation research system – ECJ.

For all tests, three different groups of parameter were explored: *evaluation budget, function set* and *depth constraint*. Tests were performed on two of the provided parameter files, *sextic regression* and *santa fe ant*. For both problems, the average fitness and hits were calculated over fifty runs over the same parameter. This number of runs was chosen for testing as it was performed by Machado et al (2015) in their experiments.

To make running the same parameters fifty times a less tedious task I created a Python script (pictured in Figure 1) to run ECJ a defined number of times and calculate the average number of hits, average standardised fitness and the standard deviation of all the obtained results. Additionally, the *runecj.sh* script was edited to calculate the time taken for each run, in milliseconds. My python script and the edited *runecj* script can be found at: https://github.com/mcallistertyler95/Bio2

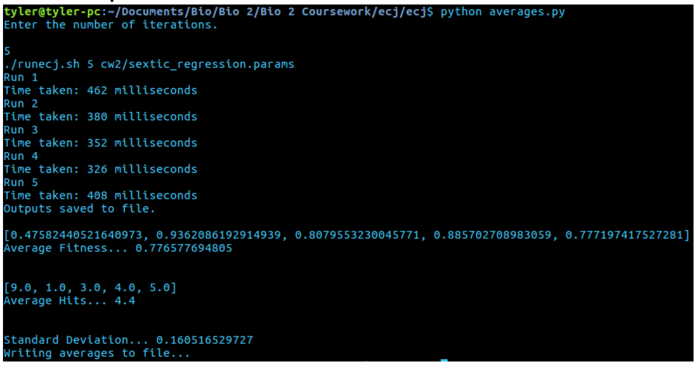


Figure 1 - Script running on 5 iterations

Population Size

The first parameter that was changed was the population. In genetic programming, population is the number of individuals, or solutions, that are created during each iteration. Some of these individuals are chosen for mutation and crossover and moved into the next generation. When testing, all other parameters were kept at their default values: Generations at 50, maxdepth of mutation and crossover at 17, the size of tournament selection kept at 7 and no changes made to the function set. Population was set to 10 and was gradually increased to 1000, increasing the increment to increase population by after a few iterations. From Figure 2 we can see that as the population was increased the average fitness generally always got lower in the *santa fe ant* problem. As the standardised fitness lowers, we get better results (higher hits). However, for “good” results (fitness below 30) the population had to be greatly increased, reaching 900 individuals. The same test was run on the *sextic regression* problem as shown in Figure 3. Here population had the same effect, steadily decreasing fitness as it was increased, although the fitness reached values much closer to zero by the time 1000 individuals was reached. Poli, Langdon and McPhee (2008) state that *“the most important control parameter is the population size”*, the results obtained reflect this, as solely raising the population leads to lower fitness.

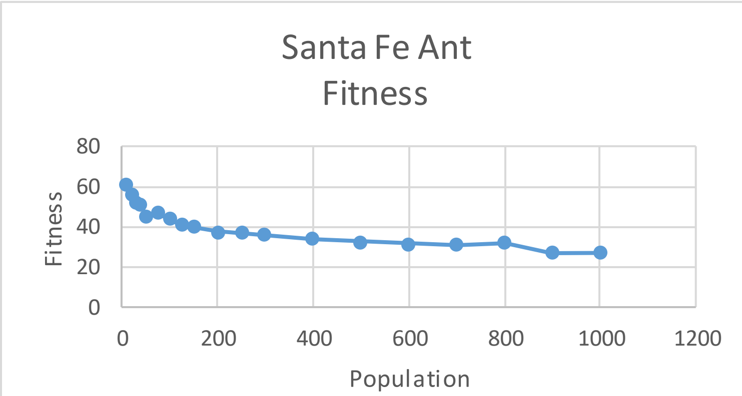


Figure 2 - Population Increasing

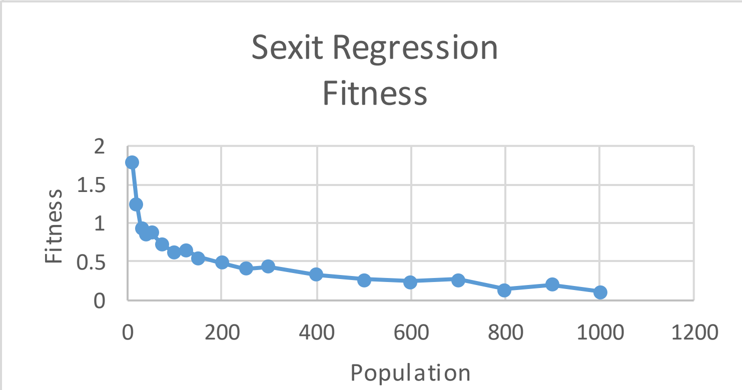


Figure 3 - Population Increasing

Generations

In genetic programming the generation size is the number of iterations of newly generated individuals. We use generations to carry over children from previous individuals that have been mutated and crossed over. By increasing generations we can increase the amount of genetic diversity in the population as well as converge towards an optimal solution. For these experiments the population was kept at 100 and every other parameter was kept at their default values mentioned previously. Generation size was set to 50 and increased by increments of 50 until reaching 550. This rate of increasing generations was done by Silva et al (2011), where the initial generation value was 50 and was increased to 500. Figure 4 shows the results of increasing generation size in the *santa fe ant* problem.

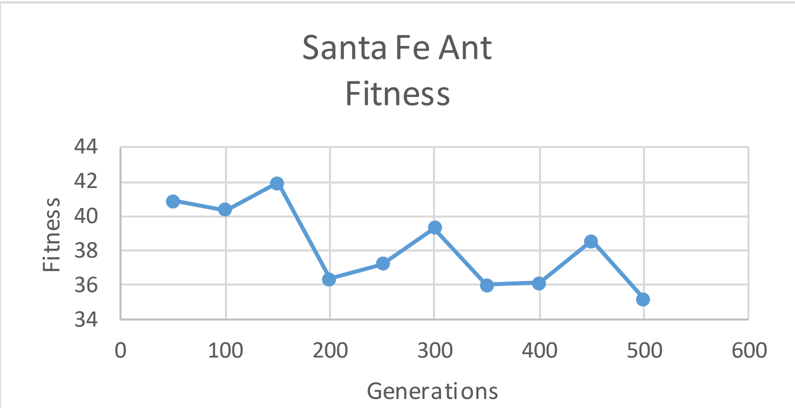


Figure 4 - Generations Increasing

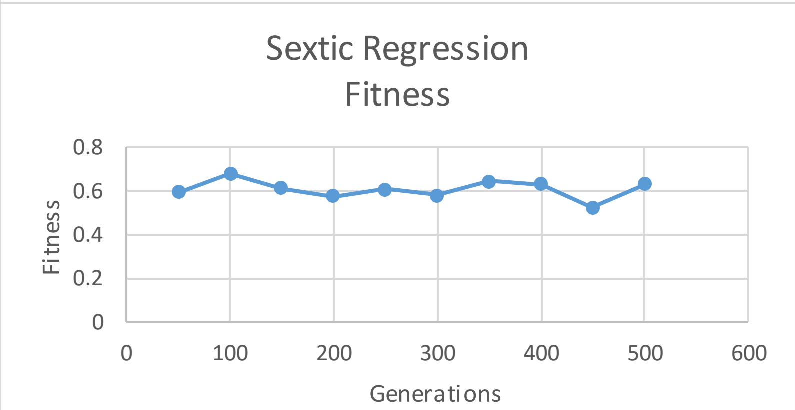
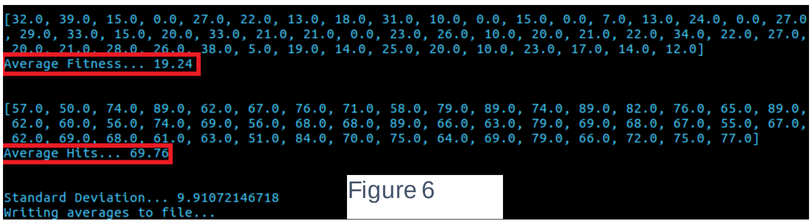


Figure 5 - Generations Increasing

As the generation size was increased the fitness took a lot longer to fall. The fitness would only steadily fall every 100 generations meaning the number of generations was not as effective at improving the results of the program than the population size was. The average time to run was around *2414.6 milliseconds* with generations set to 500 while the average was *1420.82 milliseconds* with the population set to 1000. From this we can observe that generations are more computationally expensive than population.

Figure 5 highlights our previous conclusion in that the fitness is overall not largely affected by the increase in generation alone. Even more interestingly the results in *sextic regression* appear to be almost unaffected by the increase in the number of generations. By increasing the number of generations, the number of iterations of newly generated population, mutation, crossover and tournament selection increases but if the number of individuals is low, as it was here then our results will generally not see a large change. By using ideal generation and population values together the fitness will improve overall because the search space for individuals increases as well as the number of iterations for evaluating the best individuals and creating new ones. To come to this conclusion, I ran two experiments using the santa fe ant problem with my script. The first experiment had generations set to 500 and population to 1000, the results are in Figure 6. The second experiment had generations set to 50 and population to 1000. With the generations and population at higher values, the average number of hits was much higher (69.76) than with a lower population where the average number of hits was much lower (58.3).



```
[32.0, 39.0, 15.0, 0.0, 27.0, 22.0, 13.0, 18.0, 31.0, 10.0, 0.0, 15.0, 0.0, 7.0, 13.0, 24.0, 0.0, 27.0
, 29.0, 33.0, 15.0, 20.0, 33.0, 21.0, 21.0, 0.0, 23.0, 26.0, 10.0, 20.0, 21.0, 22.0, 34.0, 22.0, 27.0,
 20.0, 21.0, 28.0, 26.0, 38.0, 5.0, 19.0, 14.0, 25.0, 20.0, 10.0, 23.0, 17.0, 14.0, 12.0]
Average Fitness... 19.24

[57.0, 50.0, 74.0, 89.0, 62.0, 67.0, 76.0, 71.0, 58.0, 79.0, 89.0, 74.0, 89.0, 82.0, 76.0, 65.0, 89.0,
 62.0, 60.0, 56.0, 74.0, 69.0, 56.0, 68.0, 68.0, 89.0, 66.0, 63.0, 79.0, 69.0, 68.0, 67.0, 55.0, 67.0,
 62.0, 69.0, 68.0, 61.0, 63.0, 51.0, 84.0, 70.0, 75.0, 64.0, 69.0, 79.0, 66.0, 72.0, 75.0, 77.0]
Average Hits... 69.76

Standard Deviation... 9.91072146718
Writing averages to file...
```

Figure 6

Mutation and Crossover Max Depth

Mutation and crossover methods are key in genetic programming as they focus on creating diverse individuals that explore the search space of solutions early on. The max depth for mutation and crossover were treated as one parameter so their values were kept the same during each experiment. Generations was set to a constant 500 for each experiment and population to 100, while other parameters were kept to default. Figure 7 shows the effect of mutation on the *santa fe ant* problem and 9 shows its effects on *sextic regression*. The mutation and crossover showed the most effect on decreasing the fitness values when the max depth was set between 1 to 5, once it reached 8 it began to converge for both programs.

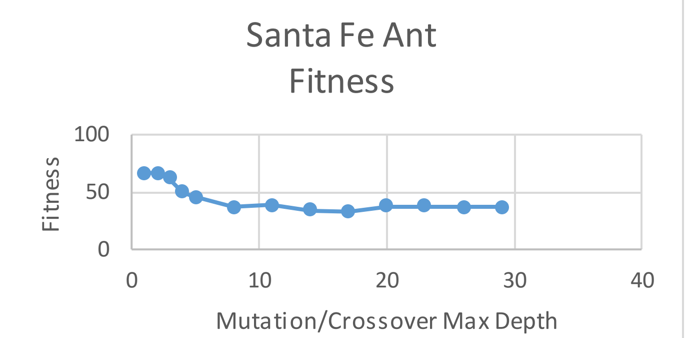


Figure 7

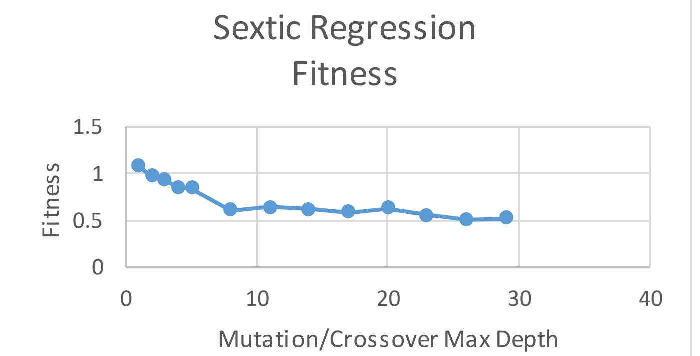


Figure 8

The largest jump in fitness decreasing was when the max depth for mutation and crossover was 8. The fitness does not dramatically decrease at any point after this. The reason for this may be that for both programs, a maximum depth of 8 is all that is needed for them to reach optimal solutions, anything more than this is unnecessary for finding the best solutions.

Once the maximum depth reaches 20 the execution time for one run of the program is also far greater than a lower max depth. This may be due to the

Function Set Size

The function set serves as the set of operations the genetic program can use when creating new individuals, these can range from mathematical functions to unique, program specific, operations. For experimentation, certain functions were removed. Figure 9 shows the removal of certain functions in the *sextic regression* problem. As described by, Alvarez (2000) basic operations such as addition, multiplication subtraction and division are capable of approximating sine, cosine, logarithmic and exponential operations. Therefore, I steadily removed each of these from the function set in

Figure 9

sextic regression. Generations and population were set to 250 and 200 respectively. Figure 9 shows the average fitness as each function is removed. Interestingly the fitness is better when sine is removed than when all functions are present. This may mean that in the *sextic regression* problem sine makes it more difficult to obtain optimal results.
As expected and stated by Alvarez (2000), as we remove mathematical operations from the function set the program performs worse overall as the genetic program becomes more limited with the amount of the functions it has and must approximate the missing ones using these functions. Here, sine, cosine, exponential, logarithm and multiplication can all be approximated with the addition and subtraction functions but as division is removed the standardised fitness greatly rises and we obtain extremely bad solutions. This may be due to division being impossible to approximate using addition and subtraction alone.
For the *santa fe ant* problem the experiment done was slightly differently. Only a few chosen members of the function set were removed in each experiment, instead of continuously removing one as done in Figure 9. The bar chart in Figure 10 below shows how the fitness is effected when the ant has some of its movement options are removed.

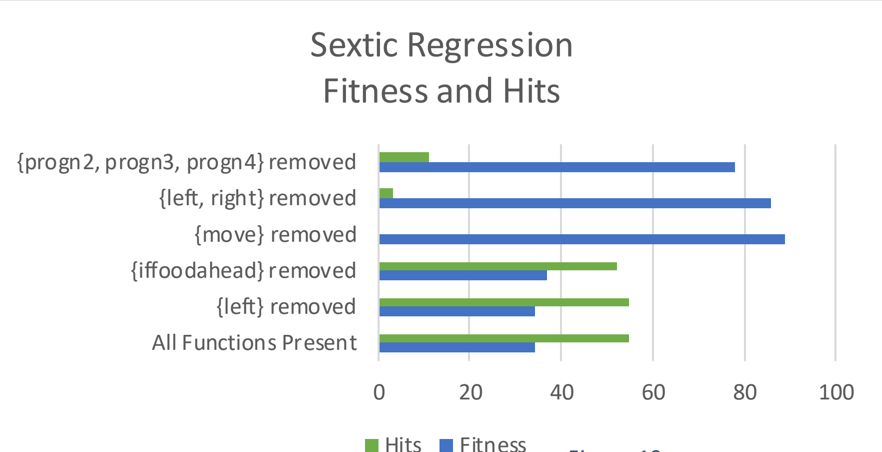


Figure 10

When *move* is taken out of the function set we can see that without any way of moving from its initial position the ant cannot eat anything on the path, so the fitness reaches its maximum, 89, and the number of hits becomes exactly 0 with each run. Similar results can be seen when *left* and *right* are removed as the ant is constrained to only moving forward from its starting position.
However, when only one function was removed, in this case *left*, the ant was still capable of producing similar results. This is due to a similar concept in the *sextic regression* experiments. With genetic programming, operations within the function set can still be used to approximate actions that are not within it. In this case, when left is removed the ant can still move left in theory, it just needs to take more actions, just as when *iffoodahead* is removed the ant can still move forward and eat food it will just have to use more actions to do it. Ultimately I believe the main differences in these problems lie in their function set. Mathematical operations are much easier to approximate so performance takes longer to drastically decrease than in the *santa fe ant* problem where movements can be approximated but removing too many results in immediate bad results.

References
Poli, R., Langdon, W. and McPhee, N. (2008). *A Field Guide to Genetic Programming*. 1st ed. p.26.
Machado, Penousal et al. *Genetic Programming*. 1st ed. Copenhagen, Denmark 2015. p.161-163
Silva, Sara et al. *Genetic Programming: 14th European Conference*. 1st ed. Torino, Italy 2011 p.30
Alvarez, L. (2000). *Design Optimization Based On Genetic Programming*. Ph.D. Department of Civil and Environmental Engineering, University of Bradford. p.50-52