Johan de Clercq
U19046121

# Assignment 3 – Grammatical evolution of assignment 2

## The Grammatical evolution algorithm

The grammatical evolution algorithm made use of a grammar to produce decision trees that could classify heart disease. The grammar consisted of:

<e> = <o4><e><e><e><e> | <o3><e><e><e> | <o2><e><e> | <v>

<o4> = <cp> | <ca>

<o3> = <thal> | <slope> | <restecg> | <age> | <trestbps> | <chol> | <thalach> | <oldpeak>

<o2> = <sex> | <fbs> | <exang>

<v> = <0> | <1> | <2> | <3> | <4>

And made use of 8bit codons for selection from the grammar. Therefore, each member of the population consisted of a string of codons used to build the expression that would be translated into a decision tree. There was a limit set to the depth an expression could extend to, that being 5, to stop the possibility of infinite tree size.

The fitness function made use of similar calculations as those meant for F1-score, that being the number of true positives (TP), right guesses, and false positives (FP), wrong guesses, each tree produced. These were then used to calculate the accuracy of each of the trees using the following formula:

$$acc = \frac{TP}{TP + FP}$$

a weighted F1-score was tried but it resulted in poorer overall search space searching and was thus not used.

Tournament selection was chosen as the selection method. It works by creating a tournament in which a certain number of population members are pitted against each other, using their fitness scores, and the best one chosen. This tournament is done with replacement so that each time the tournament is held each of the population has a probability to be selected, with the best fitness scores having the highest probability of being selected. The selected members are then passed on to the genetic operators.

The two genetic operators were crossover and mutation.

**Crossover** was used as a means for exploiting our search space. It made use of tournament selection to select two members of the population. Each of the members codon strings were taken and a random index selected. Then every codon to the right of the index would be swapped with everything on the right of the second members codon index. An example to make it clearer:

Pop1 = '11100111', '11011010', '10011001'

Pop2 = '10011000', '01110100', '00110001'

Now we randomise two indexes with pop1 getting 0 and pop2 getting 1. Now we swap everything to the right of those indexes leaving:

Pop1 = '11100111', '00110001'

Pop2 = '10011000', '01110100', '11011010', '10011001'

As our new codons for each of the pops.

**Mutation** was simpler than crossover to implement as it was only meant to explore the search space. As we want to keep a constant population size, mutation only gets applied to the first couple of members of the population, the exact number makes use of this formula: len(population) * (1-crossoverChance). Once we have the portion of our population that mutation is applied to, the mutation chance is used to see if any one of the codons of that member needs to change. If mutation happens, the codon is recreated and reapplied to the members chromosome.

Only one termination criterion is used for the GE. That being number of generations ran. Initially if the population stagnated to the point where more than 50% of the population had the same fitness as the best member then we would terminate, but I ran into a problem where invalid trees, or those who didn't guess one of the classifications at least once, were making too far into the generations. So, I changed the fitness score slightly to say, if a tree doesn't at least guess each classification once then give it a fitness of 0. This worked very well to increase the overall fitness but it led to a large amount of the initial population being classified as 0 fitness and would prematurely end the running of the GE. I know there are better solutions but I simply didn't have the time to implement them.

## Results for GE

The results documented below are from 10 runs. Each of the 10 runs has its accuracy listed below alongside their average, standard deviation, and best accuracies. All accuracies are in percentages.

Results from training:

| | |
|---|---|
| Training Accuracies | [49.7, 50.3, 50.89, 49.11, 52.07, 46.75, 54.44, 53.85, 49.11, 55.03] |
| Average accuracy | 51.125 |
| Standard Deviation | 2.68 |
| Best Accuracy | 55.03 |

Results from testing:

| | |
|---|---|
| Testing Accuracy | [44.26, 14.75, 50.82, 45.9, 24.59, 57.38, 55.74, 57.38, 26.23, 52.46] |

| | |
|---|---|
| Average accuracy | 42.951 |
| Standard Deviation | 15.47 |
| Best Accuracy | 57.38 |

## Results from GP

The results from the last assignment, to make discussion easier.

Results for Training:

| | |
|---|---|
| Training Accuracies | [48.21, 50.0, 47.62, 52.38, 50.6, 50.0, 50.0, 49.40, 51.19, 45.23] |
| Average accuracy | 49.46 |
| Standard Deviation | 2.01 |
| Best Accuracy | 52.38 |

Results for Testing:

| | |
|---|---|
| Training Accuracies | [57.38, 47.54, 45.90, 47.54, 44.26, 59.02, 45.90, 42.62, 55.74, 6.56] |
| Average accuracy | 45.25 |
| Standard Deviation | 14.77 |
| Best Accuracy | 59.02 |

## Comparison

To start the comparison the best place to start is to look at their overall performance as a measure of accuracy. When we do this, we can see that GE outperformed GP, only slight, in the training data. Its average accuracy and best accuracy were higher. But on the other hand, GP outperformed GE when it came to the testing data. In all honesty this is the result that I expected when only looking at accuracy. The reason I think this is expected is that both algorithm’s stem from the original idea of GP by Koza. They, therefore, should perform to a similar degree when looking at pure output.

Where GE starts to outperform GP is when we look at implementation and time complexity for running the entire algorithm. In my opinion, GE is by far easier to implement as there is no bloating of trees as all the genetic operators are performed on the chromosomes. With no tree manipulation taking place either, the algorithm is significantly faster to run through each iteration than GP. Overall, I feel like GE is the better option as its easier to implement leaving more time to do genetic operator checking and fitness function tweaking.

(Thank you for the awesome semester Prof, I really enjoyed learning and am excited for the research and heuristics next semester :D )