SET10107 Coursework

40056761@live.napier.ac.uk
SET10107 - Computational Intelligence
School of Computing, Edinburgh Napier University, Edinburgh

***Index Terms*—Neural Network, Genetic Algorithm, Multi-Layer Perceptron**

I. Introduction

THE aim of this project was to investigate how the diversity of an genetic algorithm's population effects the performance of Multi-Layer Perception Neural Network trained by said algorithm. A substantial amount of research has already been conducted into how the effectiveness of Hyper-heuristic's can be improved by encouraging diversity in the individual sub-heuristics - this investigation was designed to asses if a similar approach to Hyper-heuristic diversity could be applied to an genetic algorithm to improve it's fitness

II. Approach

Investigation was conducted by exploring how different types of Parent Selection, Child Crossover, Mutation, and Immigrant Injection can alter the overall diversity of an genetic algorithm's population - and how in turn that effects the overall fitness of the algorithm.

A. Background

Stuff about hyper-heuristics goes here. Get those references, son. [1] [2]

B. Algorithm Design

A simple genetic algorithm of similar style to supplied version was used throughout this investigation. Unless otherwise stated, a single generation of the algorithm was as follows:

Algorithm 1 Genetic Algorithm Pseudocode

A random population is generated
Two individuals are selected from the population as parents
Two children are generated from the parents
The children are mutated
Children replace two worst individuals in the population

Upon completion, the fittest chromosome produced by the algorithm were used as the weights for a multi-layer perception neural network, which attempted to perform a function estimation using pre-supplied training data. Once that algorithm was run for a predetermined number of generations, a final test using a separate set of data was used to determine the overall effectiveness of the neural network, and in thus, the effectiveness of the genetic algorithm. Details of selection, crossover and mutation methods are presented below.

Project available at: github.com/neaop/set10107_coursework

C. Algorithm Operators

1) Selection

While the provided `selectRandom` function was a perfectly valid solution to increasing the diversity of selected parents - alternative selection functions had to be designed to allow for comparison.

`selectRandom`: The provided selection method operates by simply choosing two random individuals from the population. This selection function should be considered the most diverse, as every individual in the population has an equal chance of being selected to become a parent - regardless of their fitness

`selectElite`: This selection method operates by choosing the two individuals with the best and second-best fitness. From a ignorant approach, this may seem to be a valid section tactic, in reality however this method is likely to cause the population to reach a local maxima within a relatively small number of generations. `selectElite` was included in this investigation to provide evidence that a lack of diversity in the population may be deferential to its potential fitness.

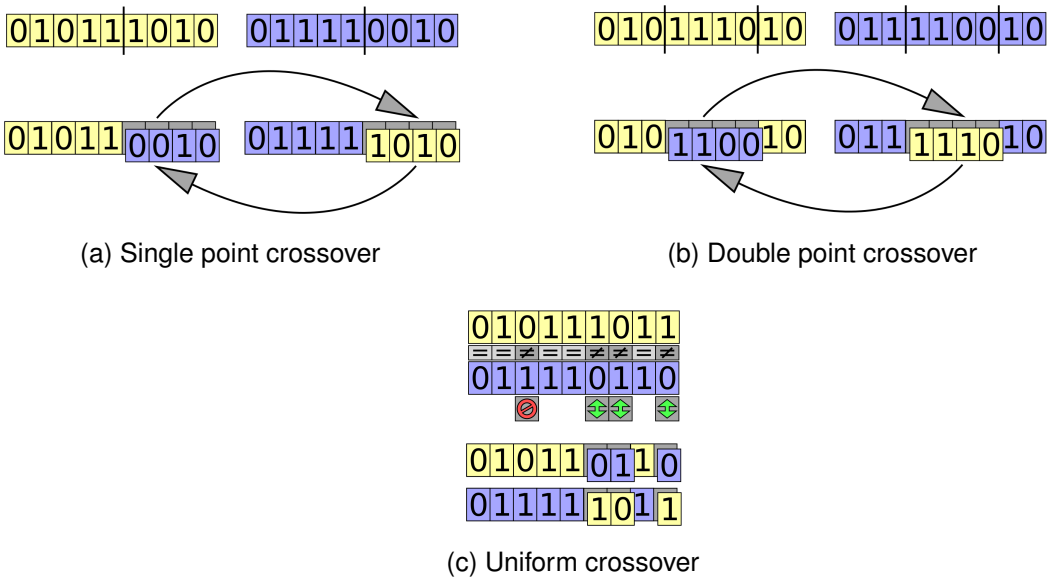
`selectTournament`: This function was designed to randomly choose a designated number of individuals - and the individual in the group with the best fitness goes forward as a parent. The number of individuals in the tournament directly alters the selection pressure - lager tournaments have a smaller chance of a weaker individual being chosen for crossover. This selection method should prove adequate in diversifying the population - however it is unlikely to be as pronounced as the `selectRandom` function.

`selectRoulette`: A roulette selection has to potential to pick any individual in the population, proportional to its fitness - e.g. an individual with a fitnesses of 10 is twice as likely to be picked over an individual with a fitness of 5. While individuals with high fitness are more likely to be selected by this function, it is not guaranteed - ensuring this function is suitable for the diversification of the population.

2) Crossover

All crossover methods utilised in this project produced a pair children which were then introduced to the algorithm population by replacing the to two individuals with the worst fitness. Crossover is the most typical way to add diversity to an genetic algorithm, however the children's similarity to their parent's is determined by the type of cross over used.

`crossoverClone`: The provided crossover function produces a pair of children, each identical to one of the parents.



(a) Single point crossover

(b) Double point crossover

(c) Uniform crossover

Fig. 1: Examples of three types of crossover function.

The use of this kind crossover will result in the eventual homogenization of the algorithm's population - significantly reducing the diversity of the chromosomes.

`crossoverSingle`: A single-point crossover method uses a random number to determine a crossover point in two newly created children chromosomes. Each child is then given a number of genes from one parent, up to the crossover point, when they then receive genes from the other parent. This crossover method produces 2 inversely mirrored children - in extreme cases can produce two children identical to the parents, reducing the overall diversity of the population by introducing duplicates.

`crossoverDouble`: Extremely similar to the `crossoverSingle` function, with the caveat being that two crossover points are generated. With the addition of a second pivot point on which to swap which parent is passing genes, the chances of producing children similar to the parents is reduced - assuming that the two parents are not already similar in terms of genes.

`crossoverUniform`: Unlike the previous segment based crossovers, a uniform crossover function operates on a gene level. Each gene in a child is copied from one of the two parents depending on the value of a randomly generated number e.g. on the value of 0 the first parent's gene is used, on the value of 1 the second parent's gene is copied. Like the other crossover functions, `crossoverUniform` produces a set of mirrored children. This cross over method has the potential to produce the most diverse children from a set of parents.

3) Mutation

Mutation of chromosomes is yet another area of a genetic algorithm were diversity can be introduced. No form of mutation was provided, so a simple boundary mutation function was designed within the algorithm presented, mutation was only applied to child chromosomes produced by a cross over function.

`mutateBoundary`: This mutation function operates over every gene in a chromosome, producing a random `double` between 0 and 1 for each one, if the double is less than a predetermined value (`mutateRate`) then the current gene is altered by adding or subtracting a second predetermined value (`mutateChange`) to the gene. Once all the genes in a chromosome were been considered for mutation any genes that were pushed outside their maximum or minimum boundary are then brought back in line.

D. Neural Network

As the primary goal of this report was to investigate how the diversity of the genetic algorithm's population changed its effectiveness, very few of the Neural network's parameters were changed or investigated.

`geneMax` & `geneMin`: The values of 5 and -5 were used respectively for the maxim and minimum possible values of gene for each chromosome. While the systematic altering and testing of these values could have led to a super solution - leaving them at the default values ensured the other test could be compared fairly.

III. Experiments & Analysis

What I did

IV. Results

What I found

V. Conclusion

Summary

VI. Future Work

What to do in the future

REFERENCES

[1] L. Hong and S. E. Page, "Groups of diverse problem solvers can outperform groups of high-ability problem solvers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 46, pp. 16 385–16 389, 2004. [Online]. Available: http://www.pnas.org/content/101/46/16385.abstract

[2] E. Hart and K. Sim, "On constructing ensembles for combinatorial optimisation," *Evolutionary Computation*, 2017.

APPENDIX A

The crossover images used in this report are courtesy of User: `Yearofthedragon` of https://commons.wikimedia.org. These images are free to use under the Creative Commons legal code and the GNU Free Documentation License.