COSC 5P71 Assignment 1 Part A Report

Nick Aksamit
Department of Computer Science
Brock University
St. Catharines, Canada
na16dg@brocku.ca

I. Introduction

Genetic programming is a problem-solving approach that incorporates notions of biological evolution into creating computer programs [1]. Some of these notions include the evolutionary standpoint of genetic inheritance from parents, with both crossover and mutation of the inherited genetic material. Following its conception, genetic programming has been applied to a wide variety of problem regions [2]. In this work, genetically evolved programs are applied to solve a symbolic regression problem, which involves constructing a mathematical model that closely imitates a given function.

II. Experimental Setup

Four sets of experiments are completed to study the effects of parameter selection on the performance of genetic programming application. A problem is defined beforehand which the evolved programs attempt to solve to best of their ability, following a specified fitness function. For this work, the problem is to construct a program that closely imitates the polynomial $x^4 + x^3 + x^2 + x$. To determine the quality of a solution, the fitness function is a squared difference between the generated program and the polynomial previously stated. Thus, the objective is to minimize the fitness. The objective function is calculated using 11 equidistant decimals within $[-1, 1]$, or $-1, -0.9, ..., 1$. Each non-leaf node in the genetic tree may consist of the following functional symbols: $x + y$, $x - y$, $x \times y$, x/y, $-x$, $cos(x)$, and $sin(x)$, where x and y are children nodes. The two potential terminal nodes are an ephemeral constant which takes the value of a random integer between $[-1, 1]$, and an argument x which takes the values of the aforementioned 11 equidistant decimals during fitness calculations.

Parameter	Value
Population Size	300
Number of Generations	40
Number of Executions	10
Tournament Size	3
Min Tree Size (Init.)	1
Max Tree Size (Init.)	2
Min Tree Size (Mut.)	0
Max Tree Size (Mut.)	2

TABLE I
Default Parameters

A set of standard parameters are used, as is defined in Table I. The population size, number of generations, number of executions, and tournament size all remain constant throughout the experimentation process. The parameters that are exploited and their values are seen in Table II. As their name suggests, both crossover probability and mutation probability define the possibility of performing their respective genetic operators. Number of elites defines the number of programs with the best fitness that are immediately transferred into the next generation.

The genetic operators, however, remain consistent throughout executions. For crossover the one point method is used, where one point is selected on two trees, and the subtree at that selected point is swapped between the two. In addition, mutation consists of also selecting a random point on one tree, but replacing the entire subtree with a freshly generated one, within a predefined size range. This size range is listed in Table I.

When initializing the population, a half and half approach is used. In this approach, half of the time a full generation is performed, while the other half a grow generation is carried out. Similarly to mutation, the size range to initialization is expressed in Table I. Subsequently, tournament selection is used for electing programs into the next generation. Using this method, some k solutions are selected at random, and the most fit is transferred. The process repeats until a new population of the same size is created, where k is pre-defined in Table I.

The purpose of this experimentation is to determine the affect a change in parameter value has on the genetic program result. It is known that a change in parameters may affect the quality of solutions obtained, along with the stability of a completed experiment [3]. Firstly, comparisons are made between a change in crossover rate and mutation rate. These genetic operators are predicted to have a large impact on the stability of a run, with mutation traditionally occurring at a low percentage, and crossover high. Secondly, further contrasts are developed between elitism, with static crossover and mutation rates. Table II illustrates the various altered parameters, where comparisons are drawn between sets 1, 2, and 3, followed by sets 1 and 4.

III. Results and Discussions

A. Crossover and Mutation

Following Section II, four sets of experiments are completed where the parameters of the genetic program are modified.

Parameter	Set 1	Set 2	Set 3	Set 4
Crossover Probability	90%	100%	0%	90%
Mutation Probability	10%	0%	100%	10%
Number of Elites	0	0	0	2

TABLE II
PARAMETER CONFIGURATIONS

Figure 1 illustrates both the average and best fitness of parameter sets 1, 2, and 3, over their 10 executions, per generation. The plot uses a logarithmic scale on the y-axis to stress any discrepancies. Experimentation on these sets expresses the effect of crossover and mutation on performance. From Table II, there is a clear extreme in always performing crossover (Set 2) and the same, but with mutation (Set 3).

On Figure 1, it is clear that having good parameter values can contribute to a stable average performance. For example, sets 1 and 2 does not often rapidly differ in fitness between sequential generations, however set 3 has widely varying values. This is likely due to mutation constantly altering each program inserted into the new generation. Each generation, every solution will have generated a new subtree or completely removed one, in a random location. Although this may be moderately advantageous for the sake of evolutionary diversity, excessive usage could lead to a reduction in performance.

From the graph, it confirms this notion as appropriate. The best fitness for 100% mutation (Set 3) is initially low, and decreases with generations, but not with abundance. Compare set 3 with set 2, and there is a massive difference in outcome. Near the final generation, set 2 falls to a best fitness of below 10^{-3}, while set 3 is not far from 10^{-1}. With a fruitless result with constant mutation in comparison to the other experiments, it cannot be recommended for practical use.

As was previously mentioned, a moderate amount of mutation could lead to diversity in solutions, which in turn may also lead to rewarding programs. Set 1 uses a high crossover rate, but low mutation (see Table II), and has a relatively stable average performance throughout execution. Its average fitness is constantly decreasing, and the best fitness per generation also presents adequate searching of good quality programs. Set 2 on the other hand, performed poorly in some of the generations, on average. This may be due to some bad crossover applications that drag the average to a less favourable value. Be that as it may, the best fitness from set 2 still outperforms both set 1 and 3. Therefore even with some instabilities on average, always and only using crossover remained superior during this experimentation.

B. Elitism

The next set of experiments determine the effect that elitism has on genetic program performance. Figure 2 illustrates the average and best fitnesses per generation when only elitism is changed. Set 4, where elitism is introduced, can be seen with a more erratic average fitness than set 1, where no elitism is used. This leads to the belief that when elite individuals are

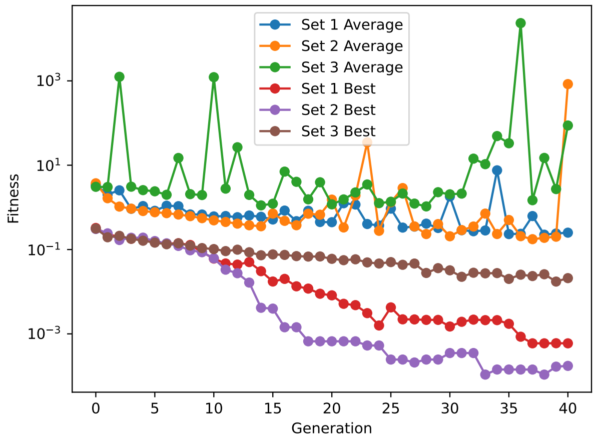

Fig. 1. Generational Performance (Sets 1, 2, and 3)

preserved, some instability appears in the genetic program. These fluctuations arrive in the later generations, rather than soon after initialization. It makes the appearance of a system that grows unstable after generations increase, rather than one that starts with volatility, but stabilizes. However, it should be noted that a similar trend arises with set 1 as well, but to a lesser degree.

Although the average performance tends to illustrate set 1 operating better than set 4, the best fitness per generation is relatively similar between the two. Both parameter configurations have a best fitness that gradually decreases. Set 4 tends to decrease quicker, while slowing down in the later generations. Set 1 decreases at a slower pace, but eventually overtakes set 4. This quick improvement of best fitness in set 4 is likely due to the elite program being directly copied into a new generation, rather than relying solely on tournament selection randomness, as in set 1. That being said, it cannot be disregarded that even with elitism, set 4 could still not produce a best individual that is better than set 1. An inference is alluded to, where elitism does not seem to provide a better answer overall, and on average may introduce instability into the genetic programming system.

IV. CONCLUSION

Genetic programming is a powerful tool that can be used for symbolic regression problems. In this work, a system is presented to reproduce the polynomial $x^4 + x^3 + x^2 + x$. Certain parameters were modified in attempt to describe their effect on the overall performance of the genetic programming system. These were: crossover rate, mutation rate, and elitism. Results express that using a lower mutation rate and higher crossover is favourable. This may very likely be due to the high possibility of replacing subtrees that contain good nodes, reducing the quality of the overall program. In addition, while 100% crossover may achieve a better overall result, on average some instabilities are introduced on average (see Figure 1).

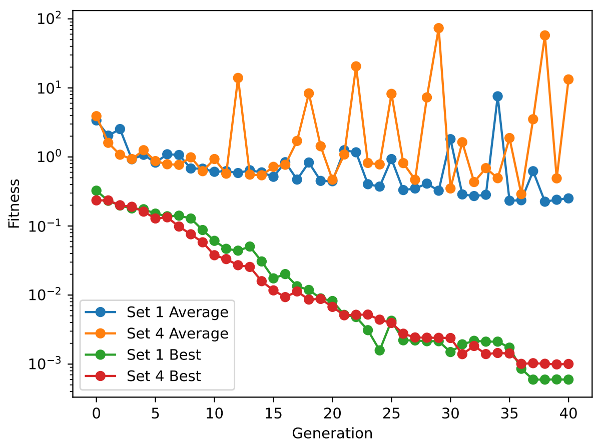

Fig. 2. Generational Performance (Set 1 and 4)

In addition, experimentation was completed to determine elitism and its affects. In early generations, it appears that elitism works well. Shortly afterwards, performance declines as instabilities start to occur in the average fitness. Various plots of high fitness value appear in one generation, and decrease immediately after. This leads to the allusion that elitism promotes instability. It is important to note that while fluctuations occur, the best fitness overall gradually decreases in a similar fashion to no elitism, but still does not achieve a fitness of the same quality.

Future work should attempt to determine the affect of parameters on different functions. This work used a relatively simple polynomial that is recreated using two elements in the function set. Perhaps the findings noted here are contradicted by the use of a more complex function. Alternatively, different mutation and crossover strategies might find a solution more effectively. Last but not least, this study did not provide any study on bloat which may have been introduced into the program trees. Further work should discuss these issues when attempting to solve a symbolic regression problem.

REFERENCES

[1] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[2] A. H. Gandomi, A. H. Alavi, and C. Ryan, *Handbook of genetic programming applications*. Springer, 2015.

[3] R. Feldt and P. Nordin, "Using factorial experiments to evaluate the effect of genetic programming parameters," in *European Conference on Genetic Programming*, pp. 271–282, Springer, 2000.