Mutation: Does it play a significant role in Genetic Programming?

Michael Andrew Nolan (03024121)

michael.andrew.nolan@gmail.com

ABSTRACT

It is thought that the greatest tool in Genetic Programming is crossover. In nearly all Genetic Programs it is crossover that is used, at many different levels, to try to find the best solution for given problems. Mutation is seen as a secondary level operation that yields very little in the way of good results to most Genetic Programs. In most Genetic Programming experiments, mutation is only used in small amounts or left out altogether. This paper lays out two of Genetic Programming's most widely known problems and explores what results can be obtained when high levels of mutation are introduced with crossover, and when mutation is used alone to find results. This paper will test between mutation alone, crossover alone, and both mutation and crossover together on "The Artificial Ant Problem" and symbolic regression with an aim to "The Quartic Polynomial". The results are then presented with some interesting outcomes.

1 Introduction

How or why sex was invented is still to this day unclear [6]. However it has many important features, it firstly allows for diversity in a population, as children are genetically different from both their parents. Also if two individuals in a population have high fitness genes then a child of these parents may end up containing both sets of high fitness genes. Hence a population grows stronger over time.

In nature beings that are more capable of adapting to their current environment, survive longer than those who do not and hence reproduce at a much higher rate. Darwin showed through the theory of natural selection that it was only the fittest individuals that went forward to reproduce.

Genetic Programming arose in the late 1980's from research being carried out in the field of *Machine Learning*. It was the method which showed the most potential two automatically write programs, inspired by biology and Darwin's ideas. It achieved this by genetically breeding populations of computer programs using the idea of natural selection. The operations it uses include reproduction, recombination and mutation.

It has been known for years in science that crossover is the one of main genetic operations that has allowed nature to evolve. It is through crossover that the fitness is passed on. It stands to reason then, that when it came to Natural Computing this theory would also hold true. Indeed it was proven in 1989 [3] and again in 1992 [4] for Genetic Algorithms. It was later proven for Genetic Programming (GP) in 1992 by Koza [5].

In the early 1950's Arthur Samuel posed the following question, which has since been the central question in computer science:

How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?

It was this statement that inspired Koza [5] to create a type of GP to solve this problem. Koza popularised LISP S-Expressions, which were tree like structures generated form Function Sets (boolean, arithmetic, loops) and Terminal Sets (inputs, constants, variables). Because these tree structures were acting like chromosomes for the programs, the logical step was to start performing crossover between sub trees of two parent trees. The results were found to be an improvement from generation to generation, and crossover became main operator in GP.

However since then the effectiveness of crossover been brought into question. Jones [1] showed that on problems that did not contain building blocks ([3], [4]), a crossover between one parent form the population and one random parent preformed as well as the standard crossover between two parents form the population, sometimes even better. This crossover using randomly selected parents is what Jones termed to be "Headless Chicken Crossover".

Angeline [2] took this theory and put it to the test on three genetic problems. He ran tests between standard sub tree crossover and headless chicken crossover over the "6-Bit Even Parity problem", the "Intertwined Spiral problem" and the "Sunspot Prediction problem". His findings over these three problems where in keeping with the findings of Jones [1], that headless chicken crossover did in fact do as well as or even better that standard sub tree crossover for these problems as they all lacked building blocks.

However throughout all this time, mutation was left to one side, and seen as of very little use in GP. Crossover was still seen as the dominate operation and it was this operation that was continuing to be looked at and tried to be improved. Koza [5] excluded mutation form his genetic programs, as he believed it caused very little difference to the outcome when used in genetic-based systems. Indeed in the experiments of Jones [1] and Angeline [2] mutation was again left out. They both simply tried to improve the crossover operation by introducing new random trees to the population.

This paper will therefore test the hypothesis that mutation causes very little difference to the outcome of GP's. Tests will be run on both the "Artificial Ant Problem" and the "Quartic Polynomial" using crossover only; mutation only; and finally both crossover and mutation together. It can then be seen from the results whether or not mutation has a significant effect.

2 Experiments

The purpose of the following experiments is to compare standard sub tree crossover when used on it own, against mutation when used on its own, and against crossover and mutation when used together on two CP problems. These problems are the "Artificial Ant Problem", and the "Quartic Polynomial". These problems have been chosen as they are widely known and relatively well understood problems often used in experiments of similar nature to those done here.

For the purpose of this paper crossover will be defined as the random selection of node in each of the two parent trees and swapping the sub trees rooted at these points between the two parents. Mutation is defined as the random selection of a node in the parent tree, followed by the deletion of said node, and then growing a new sub tree in its place, as per initializing.

2.1 Experimental Setup

All the experiments in this paper were run under the exact same conditions. Each problem had a population size of 50. As both problems are relatively simple the population size has been kept small to add some difficulty to the problems. Each problem was then run for 30 runs over 50 generations. Those experiments containing crossover had a crossover probability of 0.9. Those experiments containing mutation had a mutation probability of 0.2. All the experiments were ran using ECJ version 14 [7]. The random number generator used was the standard one that comes with ECJ. The initial seed used with this generator was randomly selected by using the system clock.

2.2 The Artificial Ant Problem

The Artificial Ant Problem evolves a program which navigates a near-sighted and motion-constrained artificial ant along an irregular trail of food [5]. The well known Santa Fe trail was being used in these experiments. The trail uses a 32 x 32 board which has 89 pieces of food laid upon it in an irregular trail. The ant was only allowed to make at most 400 moves. Moving left, moving right, or moving straight ahead by one place is considered as one of these 400 moves. The parameters for the "Artificial Ant Problem" can be seen in Table 1.

Table 1. Artificial Ant Parameters

Function Set	{if-food-ahead, begin2}
Terminal Set	`{(move), (turn-right), (turn-left)}`
Fitness Measure	(food eaten, size)

2.3 The Quartic Polynomial

The Quartic Polynomial $x^4 + x^3 + x^2 + x$ [5] was used as a target for a symbolic regression problem. The parameters for the "Quartic Polynomial" can be found Table 2. The fitness of each individual was calculated using the triple (difference, hits, size). Difference was the sum of the absolute value of the error between the individual and the actual result at any given time. Hits is the number of fitness cases where "difference" is less than or equal to 0.05.

Table 2. Quartic Polynomial Parameters

Function Set	{+, -, *, /, sin, cos, exp, log}
Terminal Set	{x}
Fitness Cases	20 evenly spaced points between the interval of [-5, 5)
Fitness Measure	(difference, hits, size)

3 Results

Each graph shows the mean avg fitness per generation over 30 runs plotted against generations.

Each table shows the mean best of run results over the 30 runs for each method; the corresponding standard deviations for the means; and the number of times each method actually found the ideal solution.

3.1 The Artificial Ant Problem Results

Figure1. graphs the results associated with the “Artificial Ant Problem”. From generation four onwards mutation alone out performs the other two methods for GP on the ant problem, reaching a highest value of 71.8 hits. Crossover alone out performs both crossover and mutation together but only by a small amount. Crossover reaches a high point of 66.6 while the method using both reaches 64.6. All three methods start to level out at there maximum fitness from approximately generation 35 onwards.

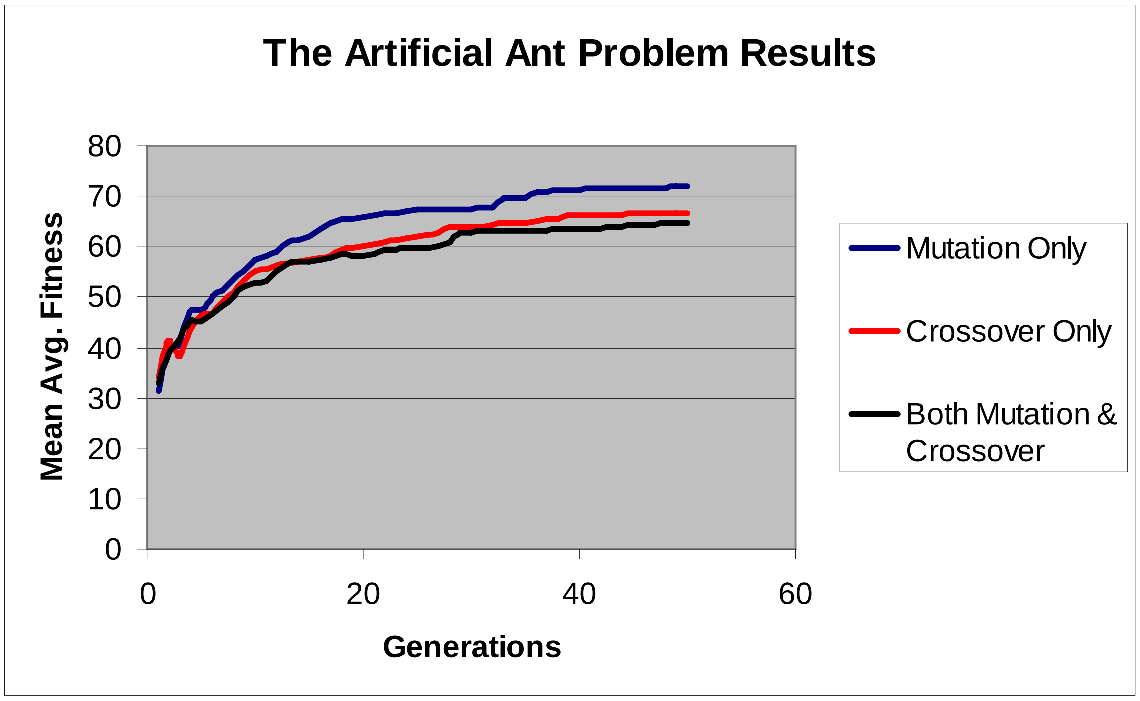

Figure 1. *Graph comparing the results of the Artificial Ant Problem when run with mutation only; crossover only; and both mutation and crossover together.*

Table3. shows the mean scores for all three methods, their standard deviations and the number of times the ideal score was actually found. Mutation alone had the highest mean hits, and found 2 ideal solutions. The method using crossover and the method using both had mean hits with less than 0.5 differences from each other. However crossover alone never found the ideal solution where as the one using both crossover and mutation found 2 ideal solutions.

Table3. Mean Best-of-Run for the Artificial Ant Problem

METHOD	MEAN HITS	Std	# IDEAL SOLUTIONS
Mutation only	60.9	9.92	2
Crossover only	56.07	8.68	0
Both Mutation & Crossover	55.66	7.73	2

3.2 The Quartic Polynomial Results

Figure2. graphs the results associated with the "Quartic Polynomial". Both Mutation and Crossover perform about as well as each other over the entire run, with crossover yielding only slightly higher results from generation 35 onwards as the methods start to level out. The method using both mutation and crossover is consistently lower than its two counterparts but not by much. Crossover yields a high of 17.93, Mutation is just behind with 16.93 and the method using both only ever gets as high as 15.46. Like the "Artificial Ant Problem" the methods here start to level out at there peak performances around generation 35.

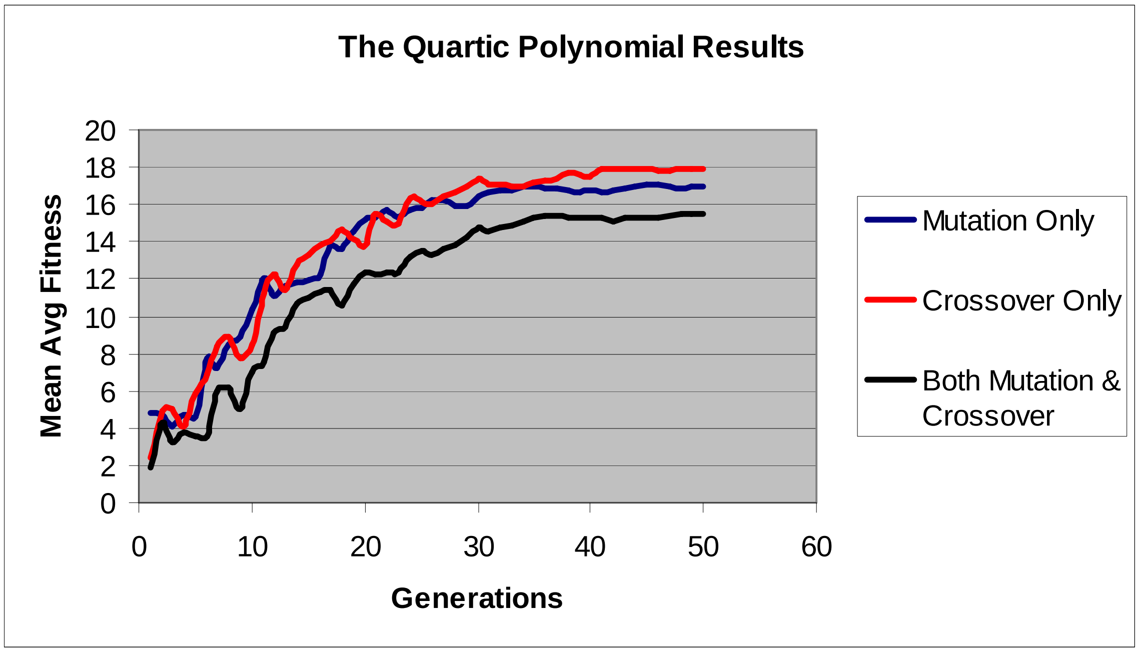

Figure 2. Graph comparing the results of the Quartic Polynomial Problem when run with mutation only; crossover only; and both mutation and crossover together.

Table4. shows the mean scores for all three methods, their standard deviations and the number of times the ideal score was actually found. Here we see that although lacking behind in overall performance in the chart the method using both mutation and crossover has a high mean hit rate. Both mutation alone and crossover alone have very close mean hits with identical standard deviation. All three methods this time returned 8 ideal solutions.

Table4. Mean Best-of-Run for the Quartic polynomial

METHOD	**MEAN HITS**	**Std**	**# IDEAL SOLUTIONS**
Mutation Only	8.73	6.48	8
Crossover Only	9.35	6.48	8
Both Crossover & Mutation	10.33	4.39	8

4 Discussion

In the “Artificial Ant Problem” mutation was shown to out perform crossover for the problem when both were tested against each other. It was mutation which found two cases of the ideal solution while crossover found none. When both were put together in the third method, the ideal solution was again found twice. This leads to the idea that this was due to the mutation being present rather than crossover. The third method lacks behind mutation but stays close to crossover, which might suggest crossover is limited and holding the method back.

From this problem it is easy to see why people thought mutation had no significant impact on results in GP as both crossover on its own and crossover with mutation are so close in comparison. But what seems to have been over looked is what happens if you only use mutation. This test shows that perhaps it is the crossover that has little significant value rather than the mutation. After all it was mutation on its own preformed the best.

In the symbolic regression problem with an aim to the “Quartic Polynomial” mutation no longer out performs crossover but it does perform relatively as well as crossover on its own. This helps show Koza’s [5] idea that mutation has little significant value, but does not prove it. The levels of mutation used here were unusually high. Indeed this helped show the power of mutation on the ant problem, but may have limited mutations true potential on this problem. Mutation introduces randomness to a population, too high a level of randomness can destroy any good work being done by the population. This may be what is happening here, and another test is needed with a slightly lower level of mutation to see if at that level can mutation out perform crossover for this problem too.

5 Future Work

There are a number steps that can be taken to follow up on the findings of this paper. These experiments can run again with larger populations and/or over longer generations to observe if mutation holds up under different conditions.

The rate of mutation was fixed at 0.2 in these experiments. This number can be increased/decreased as necessary to try to find the optimal level of mutation for each individual problem.

Other GP problems can be tested to see how mutation affects their results.

6 Conclusion

This paper has provided the first steps toward showing that mutation may have a significant part to play in Genetic Programming after all. When adjusted to the right levels mutation can, on its own, out perform crossover for particular GP problems. The initial results generated by this paper suggest that Koza's point that mutation caused very little difference to the outcome when used in genetic-based systems [5] may be wrong.

7 References:

[1] **Jones, T.** (1995). Crossover, Macromuation, and Population-based Search. *Proceedings of the Sixth International Conference on Genetic Algorithms.*

[2] **Angeline, P.J.** (1997). Subtree Crossover: Building Block Engine or Macromutation. *GECCO-99 Proceedings of the Genetic and Evolutionary Computation Conference, pp. 361-368.*

[3] **Goldberg D.E.** (1989). *Genetic Algorithms in Search, Optimization and Machine Learning,* Reading, MA: Addison Wesley.

[4] **Holland J.H.** (1992). *Adaptation in Natural and Artificial Systems,* Ann Arbour, MI: University of Michigan Press.

[5] **Koza J.R.** (1992). Genetic Programming: On the programming of computers by means of natural selection. Cambridge, MA: MIT Press.

[6] **Langdon W.B, Poli R.** Foundations of Genetic Programming. ISBN 3-540-42451-2

[7] http://cs.gmu.edu/~eclab/projects/ecj