A Revised Comparison of Crossover and Mutation in Genetic Programming

Sean Luke
seanl@cs.umd.edu
http://www.cs.umd.edu/~seanl/

Department of Computer Science
University of Maryland
College Park, MD 20742

Lee Spector
lspector@hampshire.edu
http://hampshire.edu/~lasCCS/

School of Cognitive Science
Hampshire College
Amherst, MA 01002

ABSTRACT

In [Luke and Spector 1997] we presented a comprehensive suite of data comparing GP crossover and point mutation over four domains and a wide range of parameter settings. Unfortunately, the results were marred by statistical flaws. This revision of the study eliminates these flaws, with three times as much the data as the original experiments had. Our results again show that crossover does have some advantage over mutation given the right parameter settings (primarily larger population sizes), though the difference between the two surprisingly small. Further, the results are complex, suggesting that the big picture is more complicated than is commonly believed.

1 Introduction

The genetic algorithms and evolutionary programming fields have long been at odds over the proper chief operator for generating new populations from previous ones. Genetic algorithms proponents favor crossover, while evolutionary programming's philosophy emphasizes mutation.

Most justification for using crossover as a genetic algorithm's chief operator rests on the *building-block hypothesis* [Holland 1975]. This hypothesis argues that highly-fit individuals are formed from important building blocks ("schemata"), and that through crossover, these individuals can mix and match highly-fit building blocks to form even fitter individuals. Genetic algorithms typically uses only a tiny bit of mutation, relegated to the custodial job of making sure certain features aren't weeded entirely out of the population.

In contrast, evolutionary programming often uses mutation almost exclusively, partly because of philosophical differences, and partly from a much broader use of genomes which differ widely from the traditional GA-style vector chromosome (for which crossover is straightforward). But even when using vector chromosomes, new evidence and theory has cast some doubt on the building-block hypothesis and suggested that crossover may not be as useful for GA-style vector chromosomes as previously thought (see for example [Shaffer and Eshelman 1991], [Tate and Smith 1993], [Hinterding, Gielewski and Peachey 1995]).

Genetic programming's unusual tree-based genome is so distant from the genetic algorithm vector genome that it is very difficult to form a similar theoretic justification for favoring crossover over mutation. Still, crossover is the overwhelmingly popular operator in GP. Some of this of this popularity may be due to inertia: Koza's early experiments with the Boolean 6-multiplexer problem supported his argument for heavy use of crossover [Koza 1992, pp. 599–600], and most later GP work has followed closely in the Koza tradition. But the popularity of crossover may also be due to a latent belief that GP crossover, like GA crossover, must somehow transfer "things of value" from individual to individual. As such, GP literature freely uses, with little theoretical support, the overall building-block and schemata concepts used in GA (for example, [Iba and de Garis 1996], [Rosca and Ballard 1996], [Soule, Foster, and Dickinson 1996]). GP researchers have also attempted a GP building block hypothesis ([Haynes 1997], [Poli and Langdon 1997], [Rosca 1997]).

2 The Original Experiment

In [Luke and Spector 1997], we empirically compared GP point mutation and crossover. Our goal was to to determine if crossover had any significant utility (whether this was trading "things of value" or whatnot) over being an oddball mutation operator of sorts. This study was done in light of recent high-profile studies casting doubt on the notion GP schemata [O'Reilly and Oppacher 1995], and arguing against the merits of GP crossover (for example, [Angeline 1997]).

GP is a time-intensive method. The difficulty in obtaining broad data sets in GP means that much of the GP literature to date has yielded studies with relatively narrow experiments, often over only one (even custom) domain and set of parameters. Correspondingly, many arguments based on these studies may have missed the forest for the trees. We hoped that by providing a large, broad data set in our study we might be able to see the big picture.

Consequently, we identified the four parameter settings (problem domain, population size, number of generations, and selectivity) we felt might have the largest effect on the results in our comparison, and performed experiments over a wide range of parameter combinations. The result was one of the largest GP experiments to date, resulting in 572,947,200 GP individual evaluations, or the equivalent of about 12,000 GP runs. Unfortunately, the experiment was marred by two statistical flaws:

1. Measurements at each number-of-generations milestone were taken from the same run as it continued on. This meant that a these milestones were not statistically independent. This is a serious flaw.

2. For each data point, the sample size (25) should have been larger, according to standard statistical methods. This is a less serious flaw.

In this new paper we present the results of a revision of this experiment which fixes these two problems. The revision doubles the sample size (to 50) and performs statistically-independent measurements at every data point. The result weighs in at three times the size of the previous experiment: 1,674,446,400 GP individual evaluations, or the equivalent of about 34,000 runs of typical size in the GP community (say, 50 generations, population size 1000).

3 Run Parameters

The original experiment divided runs into two sets. The first set of runs compared a 90% crossover, 10% reproduction scheme with a 90% mutation, 10% reproduction scheme, looking for a "break-even point" beyond which one or the other approach began to be consistently more successful. The second set of runs compared various blends of mutation and crossover, trying to determine if a combination of the two might be more beneficial than each separately. While both sets of runs were performed with a sample size of 25, the serious flaw (statistical dependence) occurred only in the first set. Our revised experiment replaces only the first set of runs.

As discussed in the original paper, one of the difficulties in comparing features in Genetic Programming is the large number of external parameters which can bias the results. To cope with this, we identified the four parameters we thought would have the most dramatic bias on our data. We performed runs under combinations of the following parameters:

- **Problem domain.** We picked four different domains of varying difficulty. The domains ranged from the trivial (Boolean 6-Multiplexer), to the moderate (Lawnmower, Symbolic Regression) to the relatively more difficult (Artificial Ant).

- **Population size.** Unlike the previous experiment, in this study we performed separate runs with population sizes of 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048.

- **Number of generations.** Unlike previous experiment, in this study we performed separate, independent runs lasting 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512 generations long.

- **Selectivity.** We again chose to run all four domains using tournament selection, because it allowed us to rigorously vary selectivity simply by changing the tournament size. We used two different tournament sizes: 2 (because it is the standard in GA literature, and because it is not very selective) and 7 (because it is used extensively in GP literature, and also is relatively highly selective).

Our runs reflect all combinations of these parameter settings. We chose to hold constant the myriad of other possible domain parameters, setting them to traditional default settings. There is good methodological justification for this. By making large-scale changes to the traditional domain settings, the experiment risks losing relevance to the large body of work which has used these settings in the past. And while we could choose to improve any number of parameter settings, there would always be someone arguing that if one only tried improvement *x*, the results would have been different.

We used standard "point" mutation (in which a random subtree is replaced with a new random tree) as described by [Koza 1992, p. 106]. Similarly, we used the traditional GP crossover and reproduction operators described in [Koza 1992]. We included 10% reproduction, in order to stay closer to the classic GP mix. We imposed a maximum tree depth limit of 17. We used a depth ramp of between 2 and 6 for initial tree generation, and between 1 and 4 for subtree mutation. Subtree mutation picked internal nodes 90% of the time and external nodes 10% of the time. For both initial tree generation and subtree mutation, we used half-GROW, half-FULL tree-generation. Our runs did not stop prematurely when a 100% correct individual was found, but continued until each run was completed.

The function sets and evaluation mechanisms for the Artificial Ant, Symbolic Regression, and Boolean 6-Multiplexer domains were those outlined in [Koza 1992]. The Artificial Ant domain used the "Santa Fe" trail, and allowed the ant to move up to 400 times. The target function for the Symbolic Regression domain was $x^4 + x^3 + x^2 + x$. Our implementation of the Symbolic Regression domain used no ephemeral random constants. The function set, evaluation mechanism, and tree layout (with two Automatically Defined Functions or ADFs) for the Lawnmower domain are given in [Koza 1994], using an 8x8 lawn.

Each data point in the figures is the result of 50 random runs with the same set of parameters. We performed the runs using *lil-gp 1.02* [Zongker and Punch 1995], running on a 40-node DEC Alpha supercomputer.

4 Results

The results are shown in Figures 1 through 4. The landscape graphs show the mean standardized fitness at each data point. It is important to remember that standardized fitness is monotonic but not usually linear (depending on domain); a doubling in standardized fitness does not necessarily translate to some doubling in "real fitness". The comparison graphs are black where crossover is better than mutation, white where mutation is better than crossover, and gray where the difference between the two is statistically insignificant (using a two-sample, two-tailed t-test at 95%).

If you want to compare these graphs to the (invalid) ones in the previous study, it is important to note two differences. First, for the two "easier" domains (Lawnmower and Boolean 6-Multiplexer), the original study performed runs only up to 64 generations long, and for populations only up to 512 in size. In the new experiment, all domains have run data for the same combinations of population size and number of generations. Second, the graphs in the original study are logarithmic in terms of population size but *linear* in number of generations. The new results are logarithmic both in population size and number of generations. This can be very confusing when comparing the new figures to the original ones. The new experimental data can be found at http://www.cs.umd.edu/projects/plus/GP/gpdata.tar.gz

Our conclusions from this data are rather similar to the findings in the previous study:

- Crossover was more successful overall than mutation.

- Even when statistically significant, the difference between mutation and crossover is in many places surprisingly small (though one exception is Symbolic Regression). Often changing the tournament size will have a larger effect than picking crossover over mutation. From this we conclude that crossover is doing *something* positive beyond being just an odd mutation operator, though the utility of its additional effect is usually not all that high.

- The graphs are remarkably symmetrical with respect to choosing number of generations vs. population size. Certain domains favor one over the other only to a small degree (for example, Lawnmower favors number of generations, while Symbolic Regression favors population size). Traditional GP wisdom has been that favoring large populations (where crossover often works better) produces better results than favoring large numbers of generations (where mutation often works better); but our results do not really support this.

- In [Luke and Spector 1997] we speculated that for function sets with strong, global *domain dependencies* between functions, crossover might have less utility. Domain dependencies occur when nodes throughout a GP individual take turns manipulating the domain state or internal memory. As it turns out, there is an overall trend delimiting the areas where crossover or mutation is superior. The general trend is that mutation is more successful in smaller populations, and crossover is more successful in larger populations. It is interesting to note, however, that this trend is obvious only for those two domains (Symbolic Regression, Boolean 6-Multiplexer) with no global domain dependencies.

5 Conclusions and Future Work

What we've learned from this is that while we can draw some conclusions about overall trends in the data, the data is surprisingly complex. The difference between crossover and mutation is often small, and more often statistically insignificant. Further, where and why one is preferable to the other is strongly dependent on domain and parameter settings.

There are other issues to consider: for example, tree size has a tremendous effect on total evaluation time in these domains, especially for the Lawnmower domain. We have noted anecdotally that this seems to waste more time in crossover runs than mutation runs. In a future study we hope to compare this and other factors which contribute to overall computational run length.

We would also like to further examine the utility of crossover vs. mutation with respect to the overall number of GP individual evaluations. In our previous study, we placed <population size, num-generations> tuples into classes by total numbers of evaluations, and compared crossover and mutation by evaluation class; the result was that crossover was more successful overall, but the difference was usually small and almost always less than the difference caused by changing tournament size. In preparation for this study we also ranked the new data into evaluation classes. The results were remarkably similar. However, there is no rigorous statistical test to demonstrate the validity of these findings (a t-test does show statistically significant differences between the populations of grouped averages, but these are *averages* of averages, and further, there are at most ten of them per class and as few as one per class). In the future we hope to examine this issue more closely.

We realize that large studies are hard to produce, and as such we hope our data is of use to the GP community at large, both as evidence regarding the real utility of crossover, and as a demonstration that the big picture in GP is often more complex than it seems at first. And as proof that if first you statistically don't succeed, try, try again.

Acknowledgements

This research was supported in part by grants from ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), and ARPA contract DAST-95-C0037. Thanks also to Jim Hendler for his help in the preparation of this paper. Many Bothans died to bring us this information.

Bibliography

Angeline, P. 1997. Subtree Crossover: Building Block Engine or Macromutation? *In Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97).* J. Koza et al, eds. San Francisco: Morgan Kaufmann. 240–248.

Andre, D. and Teller, A. 1996. A Study in Program Response and the Negative Effects of Introns in Genetic Programming. In *Proceedings of the First Annual Conference on Genetic Programming (GP96),* edited by John Koza et al. The MIT Press. pp. 12–20.

Angeline, P.J. 1996. Two Self-Adaptive Crossover Operators for Genetic Programming. In *Advances in Genetic Programming 2,* edited by P.J. Angeline and K.E. Kinnear, Jr. The MIT Press. pp. 89–109.

Banzhaf, W., F.D. Francone, and P. Nordin. 1996. The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming Using Sparse Data Sets. In *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation,* edited by H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel. Springer Verlag. pp. 300–309.

Hinterding, R., H. Gielewski, and T.C. Peachey. 1995. The Nature of Mutation in Genetic Algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* edited by L.J. Eshelman. Morgan Kaufmann. pp. 65–72.

Holland, J. H. 1975. *Adaption in Natural and Artificial Systems.* University of Michigan Press.

Iba, H., and H. de Garis. 1996. Extending Genetic Programming with Recombinative Guidance. In *Advances in Genetic Programming 2,* edited by P.J. Angeline and K.E. Kinnear, Jr. The MIT Press. pp. 69–88.

Koza, J.R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection..* The MIT Press.

Koza, J.R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press.

Luke, S. and L. Spector. 1997. A Comparison of Crossover and Mutation in Genetic Programming. *In Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97).* J. Koza et al, eds. San Francisco: Morgan Kaufmann. 240–248.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms.* The MIT Press.

O'Reilly, U.-M., and F. Oppacher. 1995. The Troubling Aspects of a Building Block Hypothesis for Genetic Programming. In *Foundations of Genetic Algorithms 3,* edited by L.D. Whitley and M.D. Vose. Morgan Kaufmann. pp. 73–88.

Poli, R. and W.B. Langdon. 1997. A New Schema Theory for Genetic Programming with One-point Crossover and Point Mutation. *In Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97).* J. Koza et al, eds. San Francisco: Morgan Kaufmann. 278–285.

Rosca, J.P. 1997. Analysis of Complexity Drift in Genetic Programming. *In Genetic Programming 1997: Proceedings of the Second Annual Conference (GP97).* J. Koza et al, eds. San Francisco: Morgan Kaufmann. 286–294.

Rosca, J.P., and D.H. Ballard. 1996. Discovery of Subroutines in Genetic Programming. In *Advances in Genetic Programming 2,* edited by P.J. Angeline and K.E. Kinnear, Jr. The MIT Press. pp. 177–201.

Shaffer, J.D., and L.J. Eshelman. 1991. On Crossover as an Evolutionarily Viable Strategy. In *Proceedings of the Fourth International Conference on Genetic Algorithms,* edited by R.K. Belew and L.B. Booker. Morgan Kaufmann. pp. 61–68.

Soule, T., J.A. Foster, and J. Dickinson. 1996. Using Genetic Programming to Approximate Maximum Clique. In *Proceedings of the First Annual Conference on Genetic Programming (GP96),* edited by John Koza et al. The MIT Press. pp. 400–405.

Spears, W.M. 1993. Crossover or Mutation? In *Foundations of Genetic Algorithms 2,* edited by L.D. Whitley. Morgan Kaufmann.

Tate, D.M., and A.E. Smith. 1993. Expected Allele Coverage and the Role of Mutation in Genetic Algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms,* edited by S. Forrest. Morgan Kaufmann. pp. 31–37.

Teller, A. 1994. The Evolution of Mental Models. In *Advances in Genetic Programming,* edited by K.E. Kinnear Jr. pp. 199-219. Cambridge, MA: The MIT Press.

Teller, A. 1996. Evolving Programmers: The Co-evolution of Intelligent Recombination Operators. In *Advances in Genetic Programming 2,* edited by P.J. Angeline and K.E. Kinnear, Jr. The MIT Press. pp. 45–68.

Zongker, D., and B. Punch. 1995. *lil-gp 1.0 User's Manual.* Available through the World-Wide Web at http://isl.cps.msu.edu/GA/software/lil-gp, or via anonymous FTP at isl.cps.msu.edu in the /pub/GA/lilgp directory.