# Solution Stability in Evolutionary Computation*

Terence Soule
Computer Science
University of Idaho
Moscow, ID 83844
*tsoule@cs.uidaho.edu*

Robert B. Heckendorn
Computer Science
University of Idaho
Moscow, ID 83844
*heckendo@cs.uidaho.edu*

Jian Shen
Computer Science
University of Idaho
Moscow, ID 83844
*shen7547@uidaho.edu*

## Abstract

In this paper we show that evolutionary pressure towards *stable* solutions has a significant impact on the evolutionary process. More stable solutions are less likely to be degraded by variation operators such as crossover and mutation. In particular, our results show that the pressure towards stable solutions has a significant influence on the evolutionary selection of terminals in genetic programming. We solve the same problem using a genetic algorithm with a linear encoding. We observe similar effects with the GA showing that the behavior translates to a variety of evolutionary paradigms.

## 1 Introduction

The stability of a solution is a measure of the solution's average change in fitness under crossover and/or mutation. The most outstanding evidence of pressure towards stability is the phenomenon of code growth (or bloat) in genetic programming (GP) [1, 2, 3, 4, 5, 6, 7, 3, 8, 9]. Code bloat is a rapid increase in code size that does not result in fitness improvements. The extra code usually consists of introns (code that does not contribute to the program's fitness). It is generally accepted that programs generated using genetic programming grow as a means of protecting the useful code within good solutions. By adding introns the useful code (commonly known as exons) is less likely to be affected by crossover (or other similar operators). However, Luke [7] has argued that introns themselves are not the cause of code growth. Additionally, Smith and Harries have shown that growth can occur in code that does influence fitness (exons) if the exons only have a negligible effect on performance [10]. Recently, Soule [11] has shown that code growth can occur even with exons that have a significant impact on the programs' fitness. Current research on code growth in GP strongly suggests that it will occur in any evolutionary technique which uses variable size representations [4, 6]. Langdon has shown that growth can occur in non-population based search techniques [12].

Overall code growth can be viewed as a method of making the evolving programs more stable. In this paper we show that the pressure towards stable solutions also influences the selection of primitives, terminals in GP and genes in GA, in the evolutionary process.

## 2 Experiments

For this experiment we must be able to precisely control what types of exons and introns are possible. This necessitates very simple test problems. For the GP our test problem is to evolve an expression with a fixed target value T using the plus operation and a limited set of terminals. Fitness is the absolute value of the difference between the sum of the terminals and T.

For the GA our test problem is to evolve a set of genes from a limited set of genes who's sum is equal to the target value T. The fitness of the evolved solutions is $1/(|sum - T| + 1)$.

### 2.1 The Genetic Program

We use a simple, generational GP, written in C++. The programs are tree structured. The only function (internal node) used is a + and the trees are binary. The 90/10 rule is used in selecting crossover points to be consistent with most other GP paradigms. (90% of the selected crossover points are internal nodes, 10% are leaf nodes.) For mutation each node has

*This research supported by NIH NCRR P20 RR16448, NSF grant EPS80935, and a generous hardware donation from Micron Technologies.

a probability ($P_m$) of being mutated into a random node with the same airity. Because only a single non-terminal (+) is used mutation does not change the non-terminals. In practice, the effective mutation rate for terminals is somewhat lower than the given rate because the randomly selected value for a mutated terminal may be the same as the terminal's original value.

### 2.2 Function and Terminal Sets

In general, the functions and terminals determines what types of exons and introns can evolve. In our experiments the only function is + and we tested two different terminal sets. The first function/terminal set is {+,1.0, 4.0, T = 40}. Introns are not possible, but there is an option between 1's and 4's. Clearly a larger (optimal) program is possible using 1's instead of 4's. In particular, for a target value of 40 an optimal program of all 1's is of size 79 and an optimal program of all 4's is of size 19 (including +'s). Optimal programs with a mix of 1's and 4's will fall somewhere within this range. (The set is essentially identical to one of the test sets used in the paper [11].)

The second function/terminal set is {+,0,1,4, T = 29} This set allows introns in the form of branches consisting of all 0's. Thus, there is no limit on the possible code size.

The target value T is chosen so that the average initial program has a fitness close to T. If this were not the case code growth (or reduction) would occur as the initial, random programs adjusted their sizes to reach the target value. (E.g. consider a very large target value, the initial programs would have to grow rapidly just to reach the target, regardless of other evolutionary pressures.)

The ramped half-and-half generation technique used here results in initial programs whose average (including both functions and terminals) size is 32. For the set {1,4} the average expression value of these programs is 40. For the set {0,1,4} the average expression value of these programs is 29. Thus, 40 and 29 were chosen as the target values for the sets {1,4} and {0,1,4} respectively.

### 2.3 The GA

The GA is a steady-state GA using a variable length, integer chromosome. Each integer in a chromosome can have one of three values: 0, 1 or 4. The initial length of the chromosomes is fixed at 21. Thus, the average initial value of an individual is 35 (=7*4 + 7*1 + 7*0) and so the target value is 35.

Two point crossover is used. However, to create variable length chromosomes the crossover points are chosen separately in the two parents. Thus, during crossover one parent may lose a much smaller or larger section of chromosome than it gains. This imitates tree based GP in which different sized branches are exchanged during crossover. Mutation is applied to one third of the offspring with a 0.001 probability of changing each gene in the individuals selected for mutation.

The length of the GA individuals grew extremely rapidly. To avoid memory problems individual trials were halted when the average individual length reached 40,000. A total of 94 trials were run. The data is shown for the first 48,000 evaluations. By the 48,000th evaluation 45 of the 94 trials had been halted biasing the statistics towards the surviving trials.

## 3 Results

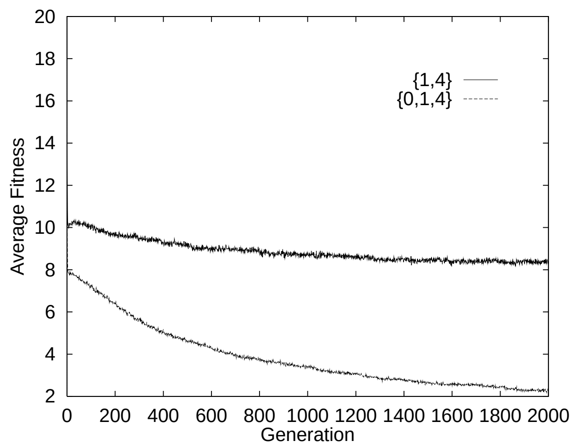


Figure 1: Average fitness for the sets {1,4} and {0,1,4}. Best fitness was zero (perfect) for all trials in all generations. The slow decline in average fitness shows that the programs are becoming more stable, resistant to the deleterious effects of crossover.

Figure 1 shows the average fitness for the terminal sets {1, 4} and {0, 1, 4} averaged across all 50 trials. (The best fitness is not shown because a perfect solution is found in the first generation and kept (elitism) in every generation. Thus, the graph of best fitness is simply a line at zero.) In both cases the average fitness improves (decreases) very rapidly for the first few generations and then improves very slowly.

For both terminal sets the population actually con-

Table 1: Summary of the genetic program and genetic algorithm parameters.

| | **GP** | **GA** |
|---|---|---|
| **Objective** | Find an expression with the value T | Find integers with sum T |
| **Function Set** | + | none |
| **Terminal Set** | 1, 4 or 0, 1, 4 | 0, 1, 4 |
| **Population Size** | 800 | 256 |
| **Crossover Probability** | 0.9 | 1.0 |
| **Mutation Probability** | 0.001 | 0.001 (1/3 of the time) |
| **Selection** | 3 member tournament | 3 member tournament |
| **Time** | 2000 Generations | up to 48000 Evaluations |
| **Maximum Size** | None | 40,000 (Average size) |
| **Elitism** | 2 copies of the best individual are preserved | none |
| **Initial Population** | Ramped, half-and-half | Random individuals of length 21 |
| **Number of trials** | 50 | 94 |
| **Crossover** | Subtree | 2 pt. variable length segments |

verges on the optimal solution in the first few generations. The average error is non-zero because it is measured after crossover and mutation. Thus, the average error actually measures how far crossover and mutation (the variation operators) move a solution away from the optimal. The decrease in average error shows that over time the solutions are becoming more stable; they are more resistant to the effects of the variation operators.

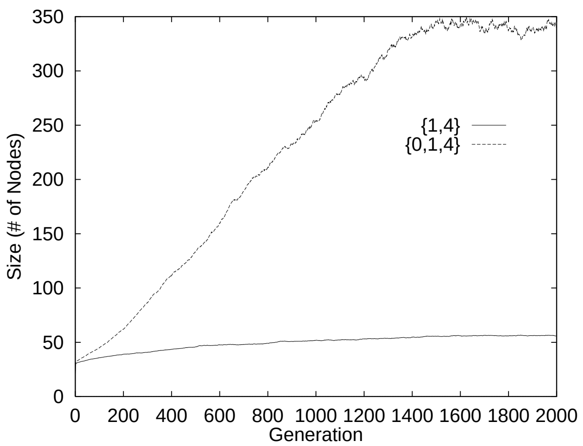


Figure 2: Average program size for the sets {1,4} and {0,1,4}. The slow increase in size for set {1,4} occurs because the 4's are being replaced by multiple 1's. The rapid increase in size for the set {0,1,4} occurs because the 0's are being added to the programs.

Figure 2 shows the average program size for both terminal sets, averaged across all 50 trials. The average size is increasing throughout the evolutionary process, albeit fairly slowly for the set {1 , 4}. For the terminal set {1, 4} this growth can only occur (without creating larger errors) if 4's in the solutions are being replaced by 1's. Thus, the data shows 4's being replaced by 1's, which leads to progressively more stable solutions.

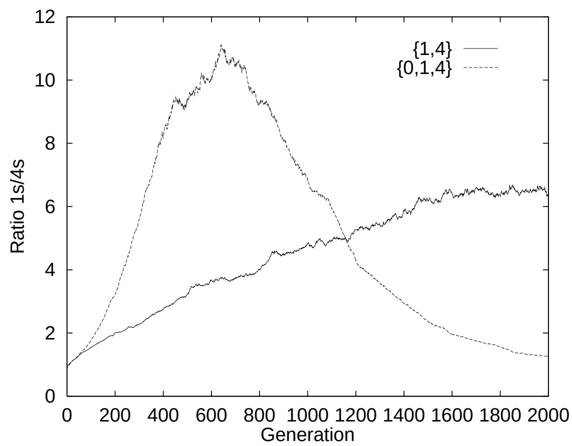


Figure 3: Ratio of 1's to 4's for the sets {1, 4} and {0,1,4}. For the set {1,4} 1's are favored to moderate the effect of crossover. For the set {0,1,4} 1's are also initially favored. However, when a critical number of 0's is reached 4's are favored to reduce the total number of exons (four 1's become a single 4) thereby better 'hiding' the exons from crossover.

This conclusion is confirmed in Figure 3, which shows the average ratio of 1's to 4's in each generation. Clearly the percentage of 1's is increasing while the percentage of 4's decreases. This change produces both the increase in size and the increase in stability.

We hypothesize that 1's are more stable than 4's because crossover is fairly likely to exchange branches with different numbers of terminals, causing a net change in the size of the offspring tree. If the parent tree consists of 4's, then fitness changes by roughly 4 for every terminal gained or lost during crossover, but fitness only changes by 1 per terminal if the terminals are 1's. Thus, 1's are more stable under crossover.

Growth with the set {0,1,4} may occur if 4's are replaced by 1's and if 0's are added. The ongoing growth makes it clear that 0's are being added throughout the evolutionary process. Figure 3 shows the ratio of 1's to 4's for the set {0, 1, 4}. Initially there is a rapid increase in the ratio of 1's to 4's. This accounts for some of the growth seen in Figure 2. However, around generation 400 this increase begins to decline and by generation 700 the trend has reversed itself; 1's are being replaced by 4's.

We hypothesize that the reversal from favoring 1's to favoring 4's occurs because of the changing proportion of 0's. Initially, the programs consist of an equal mix of 0's, 1's and 4's. Crossover is fairly likely to pick a branch that contains some of each of these terminals. As presented earlier 1's, are more stable under crossover than are 4's so there is evolutionary pressure favoring 1's. However, eventually the dominance of 0's means that crossover is fairly likely to affect only 0's. This is ideal (in an evolutionary sense) because the fitness remains unchanged if only 0's are crossed. The probability of only effecting 0's is increased if the number of non-zero terminals (exons) is minimized. (This is a standard argument; the pressure to protect exons leads to increased intron size and decreased exon size.) The number of exons can be minimized by replacing four 1's with a single 4.

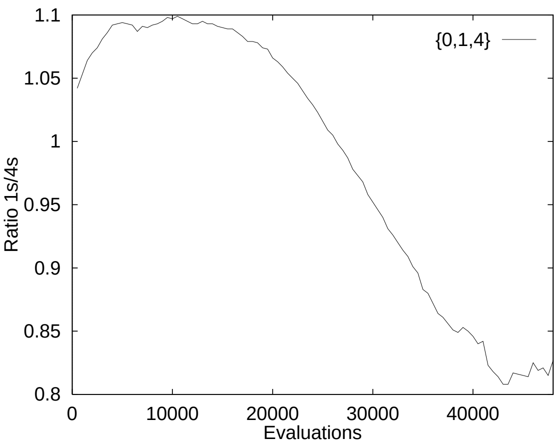


Figure 4: Ratio of 1's to 4's for the GA with 0, 1 and 4 as allowed gene values.

The results with the variable length GA were surprisingly similar. Figure 4 shows the ratio of 1's to 4's for the GA with 0, 1 and 4 as the allowed gene values. The GA chromosomes grew extremely rapidly through the addition of 0's. Initially 1's are favored followed by an eventual shift to 4's. These results are qualitatively very similar to those seen with the GP despite the significant differences between the two algorithms. However, there are some quantitative differences. The maximum ratio of 1's to 4's is much smaller for the GA (1.1:1 for the GA versus 12:1 for the GP) and the minimum ratio of 1's to 4's is smaller for the GA (0.83:1 for the GA versus 1:1 for the GP). Additionally, the changes in the ratio of 1s to 4s occur much sooner for the GA than the GP. For the GA the maximum ratio of 1's to 4's is reached at about 10,000 evaluations (roughly equivalent to 39 generations) versus 600 generations (480,000 evaluations) for the GP.

We believe these differences occur because the GA uses a steady-state model. As noted previously, this is a very simple problem; optimal solutions are found in the first generation and the population converges fairly quickly. In the steady-state model the offspring of crossover replaces the worst individual of the population. As long as a small percentage of the population remains sub optimal the optimal individuals are protected from replacement.

Thus, we believe that the observed results are due to a small sub optimal, subpopulation that is being rapidly modified, combined with a larger, stable subpopulation of optimal solutions. The changes in the ratios are relatively small because the stable subpopulation is skewing the averages. The changes in the ratios occur rapidly compared to the changes with the GP because only a few individuals are actually effected by the evolutionary process.

# 4 Conclusion

These results clearly demonstrate that there is evolutionary pressure for stable solutions. In both the GP and the GA 0's were rapidly added to the individuals which reduced the effects of crossover. This is a fairly typical example of code growth in GP. However, it is one of the few examples of 'code' growth in a GA.

More interestingly, our results demonstrate that the dynamic pressure for stable solutions influences which terminals are favored in a GP and which 'alleles' are favored in a GA. In both the GP and the GA the favored elements (terminals or alleles) changed during the course of a run, depending on the number of 0's that had been added to the evolving individuals. This represents a fairly complex dynamical change. For more difficult problems involving more and more complex elements we would expect correspondingly more complex dynamics.

Perhaps the most significant feature of these results is that they demonstrate that the entire search processes is biased by the evolutionary pressure for stable solutions. Elements are being added and removed from the evolving individuals not only to improve fitness, but also to improve stability.

## References


[1] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press, 1992.

[2] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In Jorn Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation*, pages 33 – 38. Saarbrucken, Germany: Max-Planck-Institut fur Informatik, 1994.

[3] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 310–317. San Francisco, CA: Morgan Kaufmann, 1995.

[4] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 303–309. San Francisco, CA: Morgan Kaufmann, 1995.

[5] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick R. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223. Cambridge, MA: MIT Press, 1996.

[6] Terence Soule. *Code Growth in Genetic Programming*. PhD thesis, University of Idaho, University of Idaho, 1998.

[7] Sean Luke. Code growth is not caused by introns. In *Late Breaking Papers, Proceedings of the Genetic and Evolutionary Computation Conference 2000*, pages 228–235, 2000.

[8] Peter Nordin, Wolfgang Banzhaf, and Frank D. Francone. Introns in nature and in simulated structure evolution. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Proceedings Bio-Computing and Emergent Computation*, pages 22–35. Springer, 1997.

[9] Peter Nordin. *Evolutionary Program Induction of Binary Machine Code and its Application*. Muenster: Krehl Verlag, 1997.

[10] Peter Smith and Kim Harries. Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation*, 6(4):339–360, 1998.

[11] Terence Soule. Exons and code growth in genetic programming. In James A. Foster, Evelyne Lutton, Julian F. Miller, Conor Ryan, and Andrea Tettamanzi, editors, *Genetic Programming, 5th European Conference, EuroGP 2002*, pages 142–151, 2002.

[12] W. B. Langdon. Fitness causes bloat: Simulated annealing, hill climbing and populations. Technical Report CSRP-97-22, The University of Birmingham, Birmingham, UK, 1997.