Directed Crossover within Genetic Programming

W. B. Langdon[1]

Taken from draft chapter in "Advances in Genetic Programming 2", dated 16 August 1995.

This chapter deals with evolving programs with several different requirements or tasks where a tree is allocated per requirement.

1 Directed Crossover

Often some tasks are easier than others and so evolve more quickly. It is obviously wasteful to perform crossover in code that is working correctly and the directed crossover mechanism succeeds in dynamically redistributing crossover locations to code in needed of improvement as the population evolves. The current mechanism only considers code at the level of individual operations or adfs but could obviously be refined.

In 90% of crossovers the first parent's fitness and execution path is used to bias the choice of crossover location. The location is chosen to avoid disrupting code that is working, to avoid wasting crossovers by changing code that is never executed and is biased in favour of changing code that appears to be performing poorly. The number of times the tree might be used by parts of the fitness cases which have not be executed is estimated ($unknown$). This is used as a dampening term to avoid emphisising results for trees which have been executed only a few times.

The tree in which crossover is to be performed is selected by choosing three trees at random (with reselection) from the 15. A tree cannot be one of these three if:–

- The tree is believed to be correct.

  This is decided by the fitness tests. If a tree has passed sufficient number of tests, it may be regarded as correct, even if when used in combination with other trees one or more consistency tests fail.

- The tree has never been executed,

- The tree has never been executed in a test sub sequence which subsequently failed a consistency check (nak) and it is anticipated that it would not be executed by any of the fitness test cases that have not been used ($unknown$).

However a tree may be selected as the crossover location immediately if either:–

- The number of times it was used in a test subsequence which subsequently passed its consistency check (ok) is less than nak, or

[1] W.Langdon@cs.ucl.ac.uk

- both it has never been run successfully and *unknown* is zero.

Otherwise, the following ratio is calculated:

$$\frac{nak + unknown}{ok + unknown} \tag{1}$$

When ratios for three trees have been calculated, crossover occurs in the tree with the highest ratio.

If 100 trees are examined before three meeting these requirements are found, the tree with the highest ratio found so far is selected. If none have been found, the tree is selected at random from the 15.

As far as is known, this is the first time program execution paths have been used to guide the choice of GP crossover location. (Koza [Koz92] and most others distinguish stochastically between terminals and functions, Rosca and Ballard's Adaptive Representation [RB94] uses subtree fitness measure to create new functions. Other selective crossover approaches ([D'h94] and [Kin93]) use the syntax of the code).

References

[D'h94] P. D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261. IEEE Press, 1994.

[Kin93] Kenneth E. Kinnear, Jr. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, volume 2. IEEE Press, 1993.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT press, Cambridge, MA, 1992.

[RB94] J. P. Rosca and D. H. Ballard. Learning by adapting representations in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence.* IEEE Press, 1994.