PPSN-2016, Julia Handl *et al.*, *Eds.*, LNCS 9921, p313–323, Edinburgh, 17-21 Sept. Springer. DOI:10.1007/978-3-319-45823-6_29 *Preprint*

Kin Selection with Twin Genetic Programming

William B. Langdon

CREST, Department of Computer Science,
University College London Gower Street, London WC1E 6BT, UK

Abstract. In steady state Twin GP both children created by sub-tree crossover and point mutation are used. They are born together and die together. Evolution is little changed. Indeed fitness selection using the twin's co-conceived doppelganger is possible.

Keywords: theory, genetic programming, artificial intelligence, emergent behaviour, tinyGP, steady state evolution

1 Introduction

In genetic programming it is very common to represent programs as trees (like Lisp s-expressions) and to use two point subtree crossover [Koza, 1992] to create new programs. Although subtree crossover can be symmetric and can be used to create two new programs, it is common to it to use it to create a single child [Poli *et al.*, 2008, page 15]. The offspring inherits its root node from one parent and a subtree from its other parent. Typically the second parent passes less genetic material to its offspring than the first. Also, being lower in the child's tree, typically the second parent's genes have less impact. Here we create and use both children of each two point crossover. The combined genetic contents of the two children is the same as the combined contents of their two parents. Although often subtree crossover creates a smaller and a larger child, it never changes the average size of programs. However, subtree crossover will typically mean the children are different from each other and from their parents. In many cases not only are they genetically different (their trees are different) but also the children are different sizes from each other and different from both parents.

For simplicity we restrict the GP function set to binary functions so that the trees have internal nodes with two outward facing edges. However theoretical results have also been provided for mix-arity as well as fixed arity trees [Dignum and Poli, 2010]. Without selection and using only crossover, GP populations of any initial distribution of sizes and shapes rapidly converge to a limiting distribution [Dignum and Poli, 2010]. The final size distribution depends on the initial total number of functions of each arity and the number of terminals (tree leafs). For typical GP populations, the limit distribution contains a few very small trees, the number of trees rises to a peak near the average tree size and then there is a long tail of rapidly decreasing frequency of bigger trees (see, e.g. [Dignum and Poli, 2010, Fig. 6]).

The tendency for GP populations to bloat, i.e. for evolution to produce progressively larger programs with little increase in their ability, is well known [Poli

et al., 2008, Sect. 11.3]. It has been suggested (e.g. [Dignum and Poli, 2010]) that bloat in tree genetic programming is due to subtree crossover producing small children with below average fitness. Such children are removed from the population by fitness selection. Since crossover does not change the average size, the remaining programs will tend to be bigger than average. Thus the remaining population after selection will on average be bigger than their parents. In the next round of crossovers, subtree crossover will again produce some small programs. It will keep doing this no matter how big the parent trees are. The limiting distribution still contains a sizeable fraction of small programs no matter how big its mean. This bloat theory says they will always have a tendency to be too small to be useful and so fitness will always cause selection to preferentially remove them, so the average size of trees will always increase.

Twin genetic programming was conceived with the idea (which failed) of foiling bloat. The idea being to force simultaneous fitness based removal of both the smaller and the larger child. Thus in twin GP, **two children are created** with the same average size as their parents. Although, whilst in the population, they can be independently selected to be parents, **when either child is deleted, she takes her twin with her**. (If the two parents were not selected independently but locked together, like the offspring, then the population would degenerate into independent lines and interbreeding would be impossible.)

The next section presents twin GP, Section 3 describes our experiments, whose results are given in Section 4. The failure to contain bloat and the success of kin selection are analysed in Section 5, including a mathematical model of the disruption caused by kin selection (in Section 5.3), before we summarise in Section 6.

2 Twin Genetic Programming

2.1 TinyGP

Our implementation of twin genetic programming is based on Riccardo Poli's C implementation of TinyGP [Poli *et al.*, 2008] for Boolean problems. TinyGP provides a steady state [Syswerda, 1989] fixed sized population evolutionary framework.

2.2 Two Offspring Sub-Tree Crossover

TinyGP's subtree crossover essentially provides Koza's subtree crossover [Koza, 1992] but without a bias in favour of choosing functions as crossover points. That is, the two crossover points (one per parent tree) are chosen independently at random from all internal and external (leafs) nodes in the tree.

In TinyGP once the initial population has been created, there are no size or depth restrictions on the evolving trees and after a few generations bloat is usually rampant.

The only change for twin GP is to use the two crossover points twice so creating two offspring (the twins).

2.3 Point Mutation

Children not created by crossover are created by applying point mutation [Poli *et al.*, 2008, pages 16-17] at 5% per node to a copy of their parent. Thus larger trees are proportionately more likely to be changed at multiple places. Notice this type of tree mutation does not change the size or the shape of the program.

For twin genetic programming, two child programs are created by two independent mutations and then locked together as twins. Since the parents are chosen independently the twins (like those created by crossover) are typically of different sizes and shapes.

2.4 Fitness 6 Multiplexor

As a demonstration we use TinyGP's Boolean six multiplexor problem. The goal is to evolve a program which takes six Boolean inputs and outputs a Boolean corresponding to 6-Mux [Koza, 1992, Sect. 7.4]. I.e. two inputs correspond to two address lines (giving 4 combinations) which select one of the remaining four inputs and connect it to the output. There are $2^6 = 64$ possible tests. We use them all. An individual program's fitness is the number of test cases it for which it gives the correct answer. I.e. fitness is an integer between 0 and 64.

2.5 Twin Selection in Steady State Populations

In twin GP both selection to be a parent and deciding which two programs are to be removed from the population is on the basis of the fitness of the two twins. We looked at five ways of combining the twins' fitnesses: twin) the default, just use the individual program's fitness. MEAN) use the mean of the fitness of the program and its twin. MAX) use the best fitness of the two programs. MIN) use the worse fitness. KIN) use the fitness of the twin, i.e. kin selection. Finally, as a sanity check, we ignore fitness of the twins entirely and select randomly. As expected under random selection, evolution does not solve the problem at all and the populations do not bloat.

3 Experiments

We tried each of the five settings described in Section 2.5 and the original TinyGP. (Results for TinyGP "no twin" are at the top of Table 2). The parameters of twin GP are summarised in Table 1. Initial results for kin selection were disappointing and no solutions were found. Hence kin was re-run with a population ten times as big. We run each experiment 30 times. The results are summarised in Table 2. (For completeness, all GP runs were also also made with the larger population size.)

Table 1. Twin Genetic Programming Parameters for Solving 6 Multiplexor

Terminals:	6 Boolean inputs D0–D5
Functions:	AND, OR, NAND, NOR
Fitness:	All 64 fitness cases
Selection:	Binary tournaments used for both parent and replacement selection
Population:	1000 (or 10 000)
Initial pop:	Grow, max depth 6
Parameters:	80% subtree crossover. Both crossover points chosen at random, i.e. no function bias. 20% point mutation. 5% chance of substitution with primitive of the same arity per primitive. Notice mutants are subjected to zero or more flips and larger programs have proportionately more changes. No depth or size limits.
Termination:	Problem is solved, or 100 generation equivalents

Table 2. Twin Genetic Programming Six Multiplexor (30 runs each)

Experiment	Successful runs		Best fitness		Mean size generation 100	
pop size	1000	10000	1000	10000	1000	10000
no twin	21	30	64	64	1518.2	-
twin	13	29	64	64	1357.7	1654.5
MEAN	17	30	64	64	1167.8	-
MAX	19	30	64	64	1136.4	-
MIN	6	28	64	64	960.6	1103.9
KIN	0	11	61	64	259.4	327.0
RAND		0		50		13.6

4 Results

Table 2 shows twin GP working surprisingly well.[1] Evolution of successful programs is even possible if we use the fitness of the worst of the twins. Although kin selection is obviously doing less well, if we increase the population size, evolution can proceed even if we totally ignore the fitness of the individual and always use instead the fitness of her twin. As the last pair of columns in Table 2 makes clear, twin GP has totally failed to address bloat.

5 Discussion: Why does twin GP bloat

5.1 Two can Bloat Too

Figure 1 plots the change in total program size each time a new pair of programs is created (and so a twin is removed from the population). Figure 1 suggests a near symmetric distribution but notice the rapidly increase in variation as the population evolves to contain bigger trees. What Figure 1 conceals is that the distribution is not exactly symmetric. Figure 2 plots the average change. In

[1] According to the binomial distribution the first three variants of twin GP, i.e. twin, mean and max, are not significantly worse than TinyGP without twins.

almost every generation, on average, smaller trees are replaced with bigger ones. I.e. binary fitness tournaments have a bias towards selecting larger trees to be parents than they select to be killed. This leads to bloat. (Binary tournaments have the lowest selection pressure or intensity of any simple tournament selection scheme [Blickle and Thiele, 1996, Fig. 4].) That is, despite twin GP's careful control of the genetic operations of crossover and mutation, to ensure they do not change the total size, fitness selection is still bloating the population [Langdon and Poli, 1997].

5.2 How different are twins?

A possible explanation for all twin selection runs evolving fitter trees (including sometimes finding solutions), might have been that the twins are identical or at least very similar. However as Figure 3 shows, there is more to it than that. Firstly we consider what do we mean by two trees are similar. Figure 3 considers four similarity metrics. Firstly we look at the trees themselves and then we look at two metrics based on their outputs.

We can consider if the trees are identical. (This is [Koza, 1992]'s population variety.) And secondly if they are the same size. As expected twin GP populations, like usual GP populations [Koza, 1992], do not converge in terms of their genotypes. Even in the early generations there are almost no tournaments between identical trees and, in a typical run, none at all after generation nine. If we look at a much loose definition of tree similarity: are the trees the same size, we see a similar picture. (Of course identical trees must also have the same sizes, but not vice versa.) In the early generation about of 5% of tournaments are between trees of the same size but this falls to less than 1% after generation 13.

We also looked at similarity of behaviour. As expected [Langdon *et al.*, 1999], the populations converge to some extent. Again Figure 3 looks at two types of (phenotypic) convergence. Do two programs return identical answers on all the test cases and secondly do they have the same fitness. (Since fitness is define by the test cases, two trees which give the same answers on all 64 test cases must have the same fitness, but not vice versa.)

Phenotypes (i.e. behaviour) in twin GP populations do show some convergence (e.g. Figure 3) and also twins' phenotypes (and so fitness) are slightly more similar than those of the population as a whole. For example at the end of the run in Figure 3 30.6% (+) of twins gave exactly the same answer on all 64 tests whilst the figure for random pairs was 29.4% (×).

5.3 Expected Impact of Kin Selection in Boolean Problems

We next show in the limit in random populations kin selection impacts half fitness selection tournaments and calculate the ratio for $n = 6$ (34.3% disrupted) and show it agrees with experiment (see Figure 6).

In GP it is common (as we do here, see Table 1) to use an unbiased set of primitives. Thus before fitness selection, in Boolean problems, like the 6 Multiplexor, the chance of getting any individual test right is 50%. Therefore for

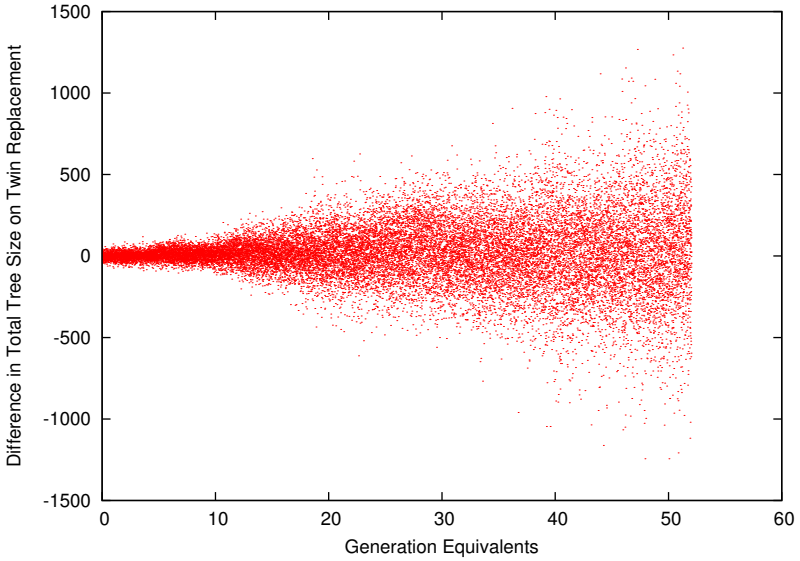

Fig. 1. Change in combined size (2×2 trees) per tournament in typical twin run (i.e. selection uses twin's own fitness). 6-Mux. Population 1000.

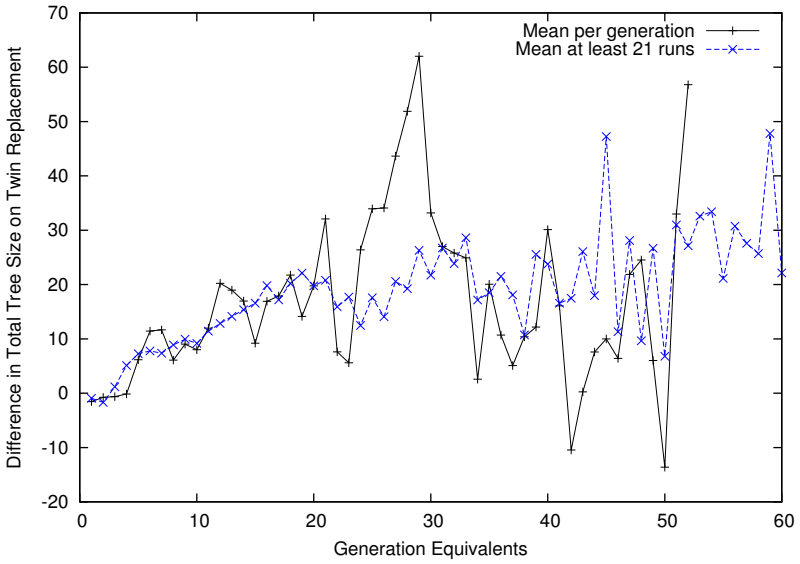

Fig. 2. Mean of size changes per generation. (+) same run as Figure 1

an n bit problem, in large trees the initial random distribution of fitness follows the binomial distribution $2^{-N}{C_i}^N$ (where $N = 2^n$). [2] Notice $2^{-N}{C_i}^N$ is symmetric about $N/2$. In the limit of large n the binomial distribution can be

[2] For the small trees the distribution is only approximately binomial [Langdon, 2009].

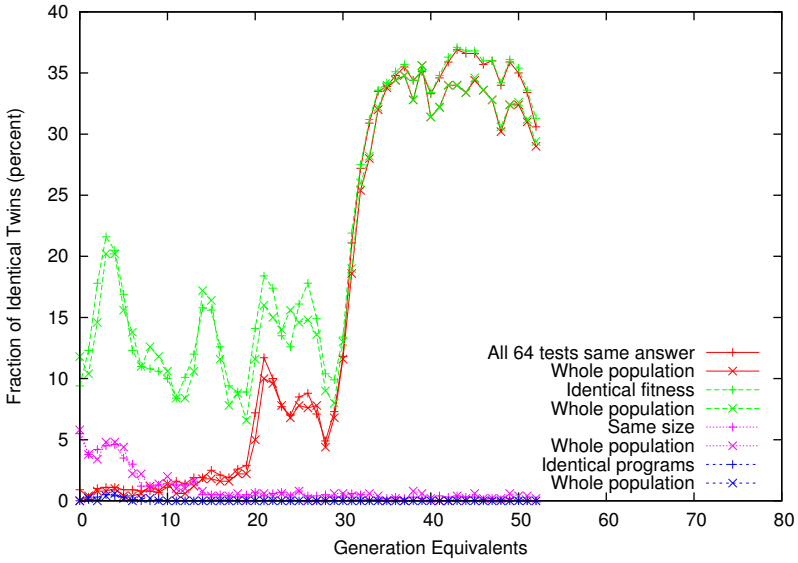

Fig. 3. Evolution of 4 measures of twin similarity in typical twin selection run (same run as Figure 1). + similarity of twins. × similarity of population. Twins (+) are slightly more similar than the population as a whole (×).

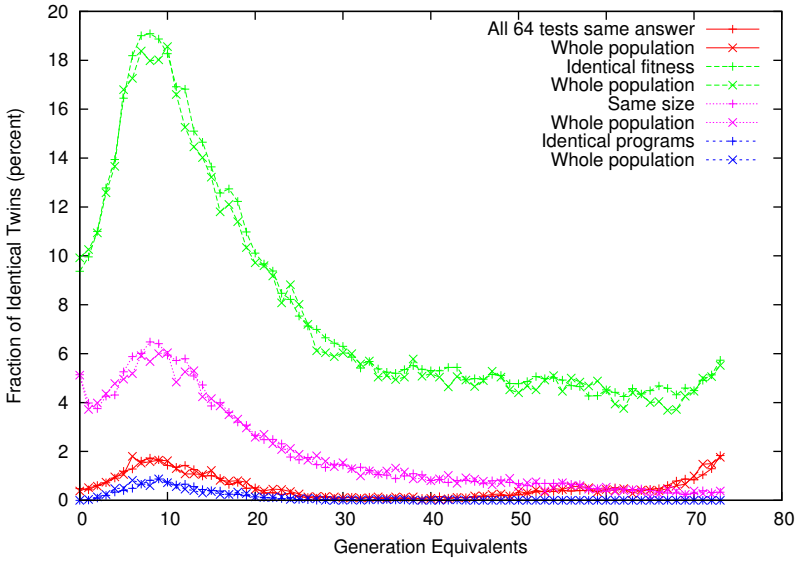

Fig. 4. Evolution of 4 measures of twin similarity in typical kin selection run with population of 10 000. Notice although the population convergence is smaller, otherwise population and twin similarity are much like selecting by the individuals fitness, Figure 3. Again twins are slightly more similar than the population.

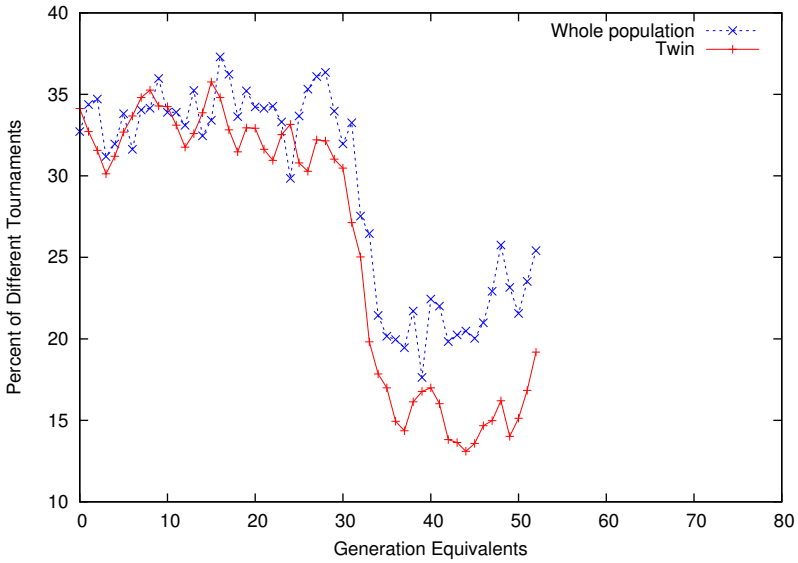

Fig. 5. Evolution of expected fraction of binary tournaments with different outcome in typical twin run with population of 1000 (+). (Same run as Figure 1.) Expected fraction for whole population plotted × for comparison. Ties are broken at random.

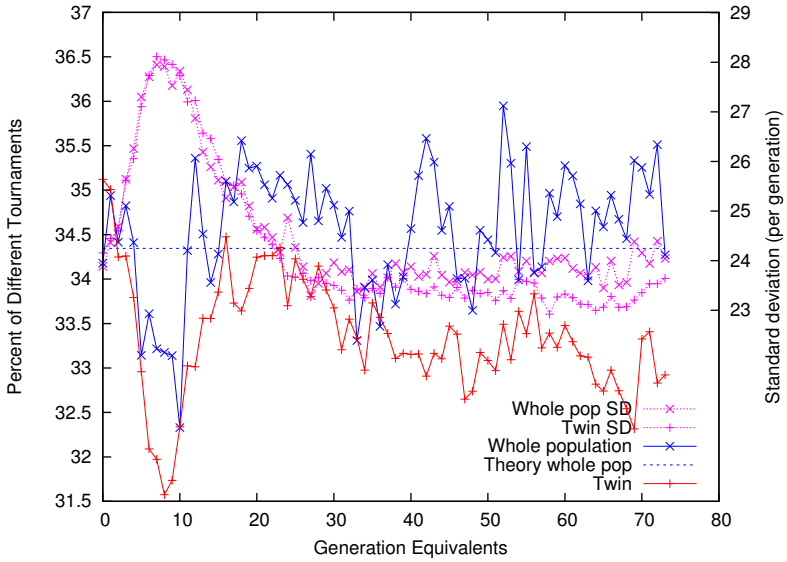

Fig. 6. Evolution of expected fraction of binary tournaments with different outcome in typical kin selection run with population of 10 000 (+). (Same run as Figure 4.) Expected fraction for whole population plotted × for comparison. Twin's fitness gives same outcome slightly more often than using the population.

approximated by a Gaussian distribution (The same holds when the functions are reversible [Langdon, 2003].) The distribution's mean is 2^{n-1} and its variance is 2^{n-2} (standard deviation $2^{n/2-1}$). (E.g. for the 6-mux the mean is 32 and the standard deviation is 4.) In a random 6-mux population there is a reasonable chance of drawing two programs with the same fitness. In higher order problems (i.e. letting n increase) the width of the distribution grows. When it is large compared to 1.0 there is essentially no chance two random programs will have the same fitness. Thus for large n we need not consider the chance of tournaments having to consider a draw where individuals have the same fitness.

Consider a program with fitness i having a twin with fitness j in a binary tournament with a program of fitness k. It will win if $i > k$ and lose if $i < k$. Draws $i = k$ are resolved randomly. We can calculate the likelihood that substituting it with its twin's fitness will not change the outcome of the tournament. The twin still wins if $i > k$ & $j > k$ and still loses if $i < k$ & $j < k$ and half draws will yield the same answer as before, i.e. $\frac{1}{2}(i = k \mid j = k)$. Assuming fitness are randomly distributed, we can calculate the probability of the same outcome as:

$$\sum_{i=0}^{N} 2^{-N}{C_i}^N \sum_{j=0}^{N} 2^{-N}{C_i}^N \sum_{j=0}^{N} 2^{-N}{C_i}^N \begin{pmatrix} \delta(i > k \;\&\; j > k) + \\ \delta(i < k \;\&\; j < k) + \\ 1/2\delta(i = k \mid j = k) \end{pmatrix} \quad (1)$$

Where $\delta(x)$ is 1 if x is true and 0 otherwise. If N is large we can ignore the draws (i.e. neglect the space occupied by $(i = k \mid j = k)$). $(i > k \;\&\; j > k)$ and $(i < k \;\&\; j < k)$ both partition the $(0..N)^3$ cube and allocate a quarter of it each. I.e. a half in total. Since the density function $2^{-3N}{C_i}^N{C_j}^N{C_k}^N$ is symmetric about the centre of the $(0..N)^3$ cube the total sum of probabilities will be a half. Thus in random populations of large Boolean problems kin selection which uses the twin's fitness will disrupt half of binary tournaments.

In the case of 6-mux, $N = 64$ and we evaluate Equation 1 numerically using the actual random tree's fitness distribution as 65.7%. (See horizontal line in Figure 6.) Notice the close agreement with estimates drawn from a real run in Figure 6 for the first generation. Some variation as the population moves away from its initial random distribution might be expected but the similarity in later generations suggests although the population's average fitness has changed its variation (standard deviation) remains similar (see dotted lines in Figure 6).

6 Conclusions

In twin genetic programming the combined size of the two children is always identical to that of their parents. Thus no genetic operation changes the average size of programs in the evolving population. Nonetheless a very small but sustained bias in fitness selection to kill smaller trees leads to cumulative increase in program size commonly known as bloat.

Twin genetic programming can be effective even when using elements of the fitness of the other twin, see Table 2. Evolution is adversely effected when

either ignoring the best fitness of the twins or exclusively using the fitness of the other twin (kin selection). However we have demonstrated kin selection can evolve solutions. Surprisingly it is able to do this although using the twin's fitness disrupts almost as many selection tournaments as choosing at random from the population (Figure 6) and yet evolution makes no progress at all with totally random selection (last row in Table 2). Section 5.3 presents a generic theoretical analysis of the impact of kin selection using binary tournaments for large programs in high order (large n) Boolean problems and applies numerical values for the special case of small trees and $n = 6$.

Acknowledgements

I am grateful for discussions with T.H. Westerdale.

Implementation

C code for twin GP is available via anonymous FTP and via `http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/tiny_gp_twin.c`

References

Blickle and Thiele, 1996. Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, Winter 1996.

Dignum and Poli, 2010. Stephen Dignum and Riccardo Poli. Sub-tree swapping crossover and arity histogram distributions. In Anna Isabel Esparcia-Alcazar et al., editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 38–49, Istanbul, 7-9 April 2010. Springer.

Koza, 1992. John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT press, 1992.

Langdon and Poli, 1997. W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry et al., editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 June 1997.

Langdon *et al.*, 1999. William B. Langdon, Terry Soule, Riccardo Poli, and James A. Foster. The evolution of size and shape. In Lee Spector et al., editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.

Langdon, 2003. W. B. Langdon. The distribution of reversible functions is Normal. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practice*, chapter 11, pages 173–187. Kluwer, 2003.

Langdon, 2009. W. B. Langdon. Scaling of program functionality. *Genetic Programming and Evolvable Machines*, 10(1):5–36, March 2009.

Poli *et al.*, 2008. Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

Syswerda, 1989. Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. David Schaffer, editor, *Proceedings of the third international conference on Genetic Algorithms*, pages 2–9, George Mason University, 4-7 June 1989. Morgan Kaufmann.