Genetic Programming Crossover: Does it Cross Over?

Colin G. Johnson

Computing Laboratory
University of Kent
Canterbury, Kent, CT2 7NF
England
C.G.Johnson@kent.ac.uk

Abstract. One justification for the use of crossover operators in Genetic Programming is that the crossover of program syntax gives rise to the crossover of information at the semantic level. In particular, a fitness-increasing crossover is presumed to act by combining fitness-contributing components of both parents. In this paper we investigate a particular interpretation of this hypothesis via an experimental study of 70 GP runs, in which we categorise each crossover event by its fitness properties and the information that contributes most strongly to those fitness properties. Some tentative evidence in support of the above hypothesis is extracted from this categorisation.

1 Introduction

Descriptions of genetic programming (GP) typically include a discussion of why the crossover and mutation operators are used (see e.g. [1, 12]). Such discussion goes beyond saying that such operators are used because they are analogies with biological processes: it gives a justification for why such operators are considered to be effective in carrying out search.

Typically the explanation given for the crossover operator is something akin to this. A crossover operator takes components of two successful parent solutions and produces a new child solution which combines features of the two parents. If "bad" features are combined, then the child will be selected out; however if "good" features are combined, then the child will have higher fitness than either of its parents and will have a high chance of being chosen by a selection algorithm

There is lots of experimental work which shows that GP is effective on particular problems (see e.g. the overview in [1], and that various versions of these operators work better or worse in ways which are broadly consistent with the above hypothesis. There is also theoretical work which provides some arguments that GP works as hypothesised (see [6] for an overview). However there seems to be little work which actually looks in detail at the results of operators in GP runs.

The aim of this paper is therefore to do this. We will define more carefully what is meant by "combining features", and then check experimentally whether

successful crossovers do result from the combination of features. In particular, the paper focuses on the results of crossover on the results of executing the program, rather than just on the program code.

The remainder of the paper is structured as follows. The next section reviews existing work in this area. A formalization of the ideas above is given, and an experiment proposed. The results of runs of this experiment on seven different test problems is given, and the results discussed. Finally, some weaknesses in the definitions are explored and suggestions for future work given.

2 Background

This section reviews some relevant background on crossover. At the core of the issues of interest is the distinction between the effect of crossover on syntax and semantics. GP defines a biologically-inspired syntax for crossover, and we expect that some notion of crossover in the semantics of the solution. It can be noted that biology does the same; operators, after all, act directly on genotypes not on phenotypes.

When the representation is simple, there is no conflict between the syntax and the semantics. Consider, for example, a GA for timetables which uses a grid of times and days as its representation for a particular room. Define (uniform) crossover by taking each grid square in turn and taking at random the value occupied by the first or the second parent; and define mutation by choosing a grid square and choosing another value. Clearly crossover and mutation are doing basically what is described in the above loose descriptions. There is no complex syntax→semantics map in which this could be "lost".

By contrast, in GP (and other more complicated representations, e.g. those with epistatic interactions), this syntactic crossover gives no guarantee of consequent semantic crossover. Two parents can both have regions of fitness, and there can be no way to carry out a crossover between them so that the child contains both regions. By contrast, in a simple GA representation, the fitness of a particular part of the chromosome makes a particular contribution to aggregate fitness, regardless of other parts. A sub-program that creates high fitness for certain fitness cases in one program can fail to do so when placed in a different context.

A number of authors have explored this connection, theoretically or experimentally. Koza [5] discusses informal analogies with the *schema theory* for genetic algorithms. This has been formalised further by Poli and Langdon [11, 6]. In particular, in [11], they carry out experiments to show the effect of different crossover operators in terms of the amount of (genotypic) material exchanged.

A number of papers have explored variants on crossover. These are interesting from the point of view of this paper not in terms of the results as such, but from the evaluation methodology. For example, O'Reilly and Oppacher [9] explore the idea of using hill-climbing to choose between a number of crossover-based moves; Ryan [13] explores the idea of *disassortative mating*, where the two parents are chosen using different criteria; and, Beadle and Johnson [2] explore the idea of

preventing crossovers where the semantics of the child are the same as one of the parents. In all of these the influence of the variant of crossover is evaluated indirectly, via its effect on fitness performance. Whilst this is valuable in terms of evaluating the *quality* of the operator, and therefore whether it is sensible to use it, it helps less with *understanding* whether the operator works as designed.

Our experiments below are one version of an attempt to do this at the phenotypic level. Our methods are similar to those carried out by Nordin, Banzhaf and Francone [7, 8], in that they are statistical analyses of all the crossovers in GP runs. In those papers, the aim is to study the fitness change in crossover; their conclusions are that most crossovers have one of two outcomes: a large reduction in fitness, or zero fitness change. In our work we similarly analyse a large collection of crossovers, but the aim of the analysis is different.

3 Does Crossover Cross Over (in practice)?

The experimental results in the work discussed above are typically characterised by a population-level approach to analysis: make a modification to the operator, and see what the end result is. In the work below we take a different approach, by looking at each time an operator is used and comparing the parents with the child. Some informal analysis of this kind has been carried out by e.g. Koza [5]. However this work simply takes a small number of particular examples of parent/child triples and analyses them by hand.

The approach taken below is to gather a dataset consisting of certain measurements made concerning every crossover that happens during a GP run. This dataset is then analysed to examine hypotheses about why crossover works.

Informally stated, the hypothesis that we are examining is as follows:

Crossover hypothesis. Crossover works by combining features of the two parents in the child. More precisely, in a successful crossover features in the input-output behaviour of the program that contribute most strongly to the fitness in each parent will contribute to regions of high fitness in the child

Exactly what is meant by "features" is left vague, and a number of possible investigations are possible depending on the definition chosed. In the examples below we will formalise this in one particular away (defining features as subsets of the fitness cases), and then explore it in the context of symbolic regression.

4 Experimental Details

This experiment looks at the crossovers in a GP run, and studies whether crossover leads to the combination of the strongest features in the two parent programs. More formally, we look at whether the fitness cases that have the strongest fitness in the child program overlap with those in the parent programs.

4.1 Methods

This experiment consists of the analysis of crossovers in a number of GP runs. The GP system used was the TinyGP system ([12, 10]). No modification has been made to the GP code as such. However, a number of sections have been added to the program to extract information about crossovers, and to analyse that information. The modified version of the code can be downloaded from `http://www.cs.kent.ac.uk/people/staff/cgj/software.html`.

For a particular crossover, let $p1$ and $p2$ represent the parents and c the child. Define f_{p1} and f_{p2} respectively to be the fitnesses of the parents, and f_c to be the fitness of the child. A crossover will be termed *positive* if $(f_c > f_{p1})\&(f_c > f_{p2})$, *negative* if $(f_c < f_{p1})\&(f_c < f_{p2})$, and *neutral* otherwise. A crossover is said to be *syntactically-identical* if the program text of $p1$ and $p2$ are the same, and *fitness-case-identical* if $p1$ and $p2$ both produce the same values on all fitness cases. Clearly all syntactically-identical crossovers are also fitness-case-identical.

The aim of the analysis is to investigate whether the strongest fitness contributors in each parent both contribute to fitness in the child. This is measured by considering the contribution to fitness of the fitness-cases in the parent and the child; a crossover will be regarded as crossing over fitness if the strongest contributors to fitness in both parents contribute towards the fittest parts of the child. An idea of this is given in figure 1. This is formalised as follows. Let the number of fitness cases be n. Take a proportion b (this will be 20% in this experiment) and form a set of those fitness cases that rank in the top $b \times n$, for each of the parents and the child: call these sets s_{p1}, s_{p2} and s_c.

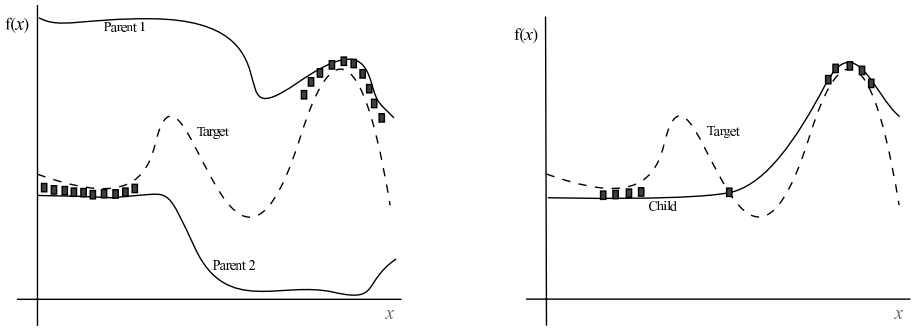


Fig. 1. An example of a good crossover: the best parts of the parents are also found in the best parts of the child.

Finally, calculate the amount of intersection between the sets representing the fittest regions for the two parents and children. Let $n_1 = |s_{p1} \cap s_c|$ and $n_2 = |s_{p2} \cap s_c|$. Define a lower threshold ℓ and an upper threshold u; in this experiment $\ell = b \times n \times 0.1$ and $u = b \times n \times 0.4$ (these were chosen following some initial informal experimentation; an future investigation will examine changes to these parameter values).

Now define a 0-*crosser* to be a crossover where $(n_1 \leq \ell)\&(n_2 \leq \ell)$, i.e. where regions that contributed most strongly to the fitness of both parents do not contribute strongly to the fitness of the child. Similarly, a 1-crosser is defined to be a crossover where $((n_1 \leq \ell)\&(n_2 \geq u))|((n_1 \geq u)\&(n_2 \leq \ell))$, i.e. one of the fitness-contributing regions (i.e. subsets of fitness cases) from the parent is important in creating the fitness of the child, and one is not. Finally, a 2-crosser is a crossover where $(n_1 > u)\&(n_2 > u)$, i.e. both of the regions that caused the parents to be fit also cause the child to be fit. Note that this is not an exhaustive partition of the space of all possible crossovers; we ignore the middle-ground where there is a small amount of intersection between both. The results below will distinguish between those 2-crossers where $s_{p1} = s_{p2} = s_c$ and those where there are differences. This is to check that genuine information exchange is going on, and that we are not just counting as 2-crossers only those crossovers which are effectively identical at the relevant sample points. These are referred to as *2-crossers(S)* in the case where the values at all the sample points are identical, and *2-crossers(D)* in the case where at least one is different.

We carry out these experiments on a set of one-dimensional symbolic regression problems. These are mostly the one-dimensional problems used by Keijzer [3], which in turn are taken from [4, 14, 15]. These functions are given in Table 1.

Ref. Number	**Function**	**Range**	**Number of Steps**
1.	$f(x) = 0.3x \sin 2\pi x$	$[-2, 2]$	101
2.	$f(x) = x^3 \exp -x \cos x \sin x$ $(\sin x^2 \cos x - 1)$	$[0, 10]$	101
3.	$f(x) = \sum_{i=1}^{x} 1/i$	$[1, 50]$	50
4.	$f(x) = \log x$	$[1, 100]$	100
5.	$f(x) = \sqrt{x}$	$[0, 100]$	101
6.	$f(x) = \text{arcsinh}(x)$	$[0, 100]$	101
7.	$f(x) = \sin x$	$[0, 6.2]$	63

Table 1. The functions used for symbolic regression in the experiments

For each function, 10 runs of a GP system with parameters set as in Table 2 were carried out. These are the default values in the TinyGP system. Despite this being an experiment purely concerned with crossover, we have retained mutation in the system because the aim of the experiment is to study crossover in the context of a "normal" GP run. Data from each crossover was stored and subsequently analysed in order to obtain the results presented in the following section.

4.2 Results and Discussion

Three sets of results are given. The first of these are in tables 3, 4, 5 and 6, which gives the results for a number of measures from across the run. The numbers presented are means from the ten runs. The numbers in the *proportions* tables represent the proportion of negative crossings which are 0-crossers, 1-crossers, et cetera, then the proportion of neutral crossings which are 0-crossers,

Parameter	Value
Population Size	100000
Crossover Probability	0.9
Probability of Mutation per Node	0.05
Number of Generations	100× population size
Selection type	Size-2 tournament selection
Replacement Strategy	steady-state
Problem Type	1-variable symbolic regression
Function Set	$+, -, \times, \div$ (protected)
Terminal Set	x, constants (generated randomly in $[-5, 5]$

Table 2. Parameters used for the GP algorithm in the experiments.

1-crossers, et cetera. Note that these columns don't add up to 1.0, as there are some crossovers that don't fit any of the 4 definitions.

Figure 2 shows the number of 0-, 1- and 2-crossers (of both types) over time. These two sets of plots are generated by grouping the crossovers into blocks of 100, and counting the number of crossovers at each point in each block. These numbers are then averaged across the ten runs at each point on the x-axis. Typically an early spike of 1-crossers is followed by a spike of 0-crossers, followed in some cases by a large number of 2-crossers(D)s. It is difficult to draw any clear conclusions from the patterns of behaviour in these graphs.

The most interesting results are in the tables of proportions. Our initial hypothesis was that crossover works by bringing together the fitness-contributing regions of each of the parents. If this were to be the case (as opposed to crossover simply being a mutation operator), we would expect the proportion of crossovers in the fitness-positive 2-crosser(D) category to be larger than those in the corresponding negative section. This is confirmed in each experiment, as is the other piece of supporting evidence, i.e. that the proportion of 0-crossers in the negative column is larger than in the proportion in the positive column.

However, it is noticeable that in each experiment, there are many crossovers where the best fitness cases are brought together in the child (2-crossers) and yet there is an overall decline in fitness.

5 Conclusions and Future Work

We have formalised the idea that a fitness-increasing crossover will combine the best features of the two parents, and shown some tentative support that this is what occurs when fitness-increasing crossovers happen. However, these experiments are rather limited in scope, and there are a number of further experiments and refinements which will be carried out in future work.

One important limitation of the definition that we have given is that a crossover of the type depicted in figure 1 can occur, but it not recognised because one of the regions in the child is not in its best fitness-producing region. Two approaches to these would be (1) to look at improvements in the strongest

Function 1 $f(x) = 0.3x \sin 2\pi x$

Total # of Crossovers:	989711
# of Syntactically-Identical Crossovers:	97111
# of Fitness-Case-Identical Crossovers:	173643
Total # of positive crossovers	22532
Total # of neutral crossovers	257685
Total # of negative crossovers	709502

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	37615	1466	684
1-Crossers	5696	6637	1786
2-Crossers(S)	122797	26824	1398
2-Crossers(D)	426317	202962	16845

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.053	0.006	0.033
1-Crossers	0.008	0.026	0.085
2-Crossers(S)	0.171	0.103	0.066
2-Crossers(D)	0.602	0.789	0.733

Function 2 $f(x) = x^3 \exp -x \cos x \sin x(\sin x^2 \cos x - 1)$

Total # of Crossovers:	989963
# of Syntactically-Identical Crossovers:	103871
# of Fitness-Case-Identical Crossovers:	129642
Total # of positive crossovers	23820
Total # of neutral crossovers	221973
Total # of negative crossovers	744170

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	42913	1598	697
1-Crossers	7046	6975	1940
2-Crossers(S)	101285	15768	1630
2-Crossers(D)	433855	170407	17022

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.058	0.007	0.032
1-Crossers	0.009	0.031	0.087
2-Crossers(S)	0.135	0.072	0.072
2-Crossers(D)	0.583	0.768	0.700

Table 3. Various measures from the crossovers (functions 1 and 2).

Function 3 $f(x) = \sum_{i=1}^{x} 1/i$

Total # of Crossovers:	990308
# of Syntactically-Identical Crossovers:	132284
# of Fitness-Case-Identical Crossovers:	148843
Total # of positive crossovers	24543
Total # of neutral crossovers	226276
Total # of negative crossovers	739489

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	297101	14592	2171
1-Crossers	109637	80765	7365
2-Crossers(S)	30797	20490	2373
2-Crossers(D)	80522	63577	8385

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.400	0.066	0.094
1-Crossers	0.149	0.360	0.299
2-Crossers(S)	0.042	0.092	0.100
2-Crossers(D)	0.110	0.275	0.337

Function 4 $f(x) = \log x$

Total # of Crossovers:	989741
# of Syntactically-Identical Crossovers:	118612
# of Fitness-Case-Identical Crossovers:	139199
Total # of positive crossovers	24339
Total # of neutral crossovers	210944
Total # of negative crossovers	754457

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	302442	15531	2594
1-Crossers	125091	86635	6962
2-Crossers(S)	23559	16535	2766
2-Crossers(D)	54276	43396	6987

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.400	0.075	0.115
1-Crossers	0.166	0.411	0.285
2-Crossers(S)	0.031	0.079	0.110
2-Crossers(D)	0.072	0.204	0.282

Table 4. Various measures from the crossovers (functions 3 and 4).

Function 5 $f(x) = \sqrt{x}$

Total # of Crossovers:	989948
# of Syntactically-Identical Crossovers:	113295
# of Fitness-Case-Identical Crossovers:	127623
Total # of positive crossovers	27818
Total # of neutral crossovers	225301
Total # of negative crossovers	736827

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	277874	11488	2079
1-Crossers	110455	79601	7312
2-Crossers(S)	27205	17420	2444
2-Crossers(D)	47924	50063	8710

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.377	0.051	0.081
1-Crossers	0.150	0.356	0.262
2-Crossers(S)	0.365	0.078	0.093
2-Crossers(D)	0.065	0.220	0.306

Function 6 $f(x) = \text{arcsinh}(x)$

Total # of Crossovers:	990366
# of Syntactically-Identical Crossovers:	105236
# of Fitness-Case-Identical Crossovers:	116170
Total # of positive crossovers	28481
Total # of neutral crossovers	226120
Total # of negative crossovers	735765

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	220624	14599	2486
1-Crossers	101593	77143	7988
2-Crossers(S)	21078	15640	2353
2-Crossers(D)	59345	52766	9090

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.300	0.065	0.094
1-Crossers	0.139	0.339	0.270
2-Crossers(S)	0.029	0.071	0.089
2-Crossers(D)	0.081	0.232	0.317

Table 5. Various measures from the crossovers (functions 5 and 6).

Function 7 $f(x) = \sin x$

Total # of Crossovers:	990024
# of Syntactically-Identical Crossovers:	188128
# of Fitness-Case-Identical Crossovers:	201428
Total # of positive crossovers	15187
Total # of neutral crossovers	201163
Total # of negative crossovers	773674

Counts:

	Negative	**Neutral**	**Positive**
0-Crossers	125859	4270	742
1-Crossers	54842	29412	2306
2-Crossers(S)	49847	23109	871
2-Crossers(D)	88051	56990	5562

Proportions:

	Negative	**Neutral**	**Positive**
0-Crossers	0.161	0.022	0.051
1-Crossers	0.070	0.150	0.155
2-Crossers(S)	0.065	0.114	0.058
2-Crossers(D)	0.115	0.278	0.361

Table 6. Various measures from the crossovers (function 7).

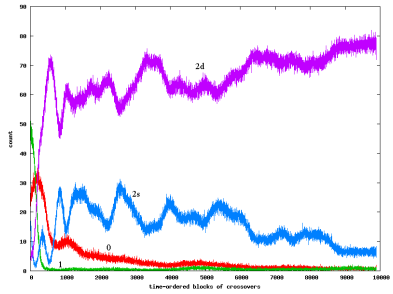
fitness-contributing regions in the parent or (2) replace the notion of a fixed proportion of strongest fitness cases with a (perhaps adaptive) notion of a fitness-improvement threshold. Another aspect to consider is the intrinsic easyness of some fitness cases in certain problems.

The experiments above have been carried out on a limited domain (they are all symbolic regression problems) and therefore it would be interesting to extend this analysis to other problem domains. Also, it would be interesting to analyse the different types of crossover at different times during the run (in more detail than the time-series plots given in this paper).

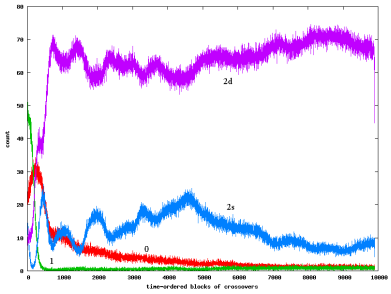
A larger-scale criticism of this approach is that it is focused on the fitness cases as individuals, and does not attempt to extract higher-scale features of the problem. An alternative approach, such as studying a formal semantics of the program [2], might be of use in approaching that question.

This paper has focused on this as an analysis approach. However, this could be turned around and used as a way of creating more crossovers of the type which appear to give positive results more frequently.

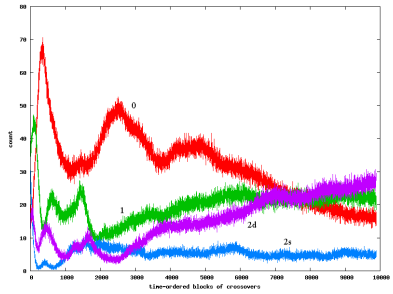
Finally, there are many other aspects of GP that that could be studied via this general idea of *instrumenting* the activities that occur during a GP run. The most obvious topic to tackle using this approach would be mutation.



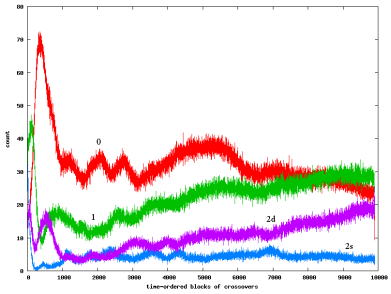
1.



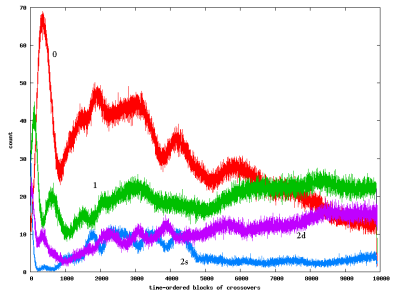
2.



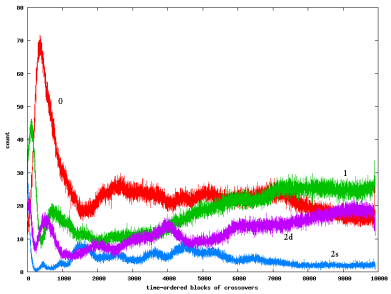
3.



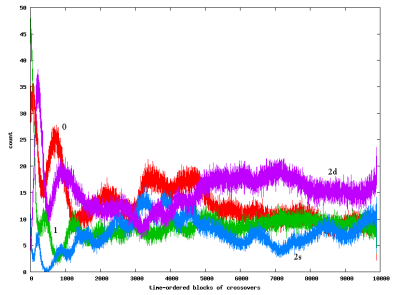
4.



5.



6.



7.

Fig. 2. The change over time in the evolutionary run of the proportion of each type of crossover for each of the 7 functions. Mean over 10 runs.

References

1. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1998.
2. Lawrence Beadle and Colin G. Johnson. Sematically driven crossover in genetic programming. In *Proceedings of the 2008 IEEE World Congress on Computational Intelligence*, pages 111–116. IEEE Press, 2008.
3. Maarten Keijzer. Improving symbolic regression with interval arithmetic. In Conor Ryan et al., editor, *Proceedings of the 6th European Conference on Genetic Programming*, pages 70–82, 2003.
4. Maarten Keijzer and Vladan Babovic. Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 76–90, Edinburgh, 15-16 April 2000. Springer-Verlag.
5. John R. Koza. *Genetic Programming : On the Programming of Computers by means of Natural Selection*. Series in Complex Adaptive Systems. MIT Press, 1992.
6. William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer, 2001.
7. Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.
8. Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 9 July 1995.
9. Una-May O'Reilly and Franz Oppacher. Hybridized crossover-based search techniques for program discovery. In *Proceedings of the 1995 World Conference on Evolutionary Computation*, volume 2, pages 573–578, Perth, Australia, 29 November - 1 December 1995. IEEE Press.
10. Riccardo Poli. TinyGP software. `http://cswww.essex.ac.uk/staff/rpoli/TinyGP/`; visited November 2008.
11. Riccardo Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
12. 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).
13. Conor Ryan. Pygmies and civil servants. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 243–263. MIT Press, 1994.
14. Rafal Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.
15. L. Sanchez. Interval-valued GA-P algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):64–72, April 2000.