Genetic Programming with One-Point Crossover and Point Mutation

Riccardo Poli and W.B. Langdon
School of Computer Science
The University of Birmingham (UK)
E-mail: {R.Poli,W.B.Langdon}@cs.bham.ac.uk

Abstract

In recent theoretical and experimental work on schemata in genetic programming we have proposed a new simpler form of crossover in which *the same* crossover point is selected in both parent programs. We call this operator *one-point crossover* because of its similarity with the corresponding operator in genetic algorithms. One point crossover presents very interesting properties from the theory point of view. In this paper we describe this form of crossover as well as a new variant called *strict one-point crossover* highlighting their useful theoretical and practical features. We also present experimental evidence which shows that one-point crossover compares favourably with standard crossover.

1 Introduction

Genetic Programming (GP) has been applied successfully to a large number of difficult problems like automatic design, pattern recognition, robotic control, synthesis of neural architectures, symbolic regression, image analysis, natural language processing, etc. [6, 7, 5, 8, 1, 14, 16, 15, 13]. However, only a relatively small number of theoretical results are available which try and explain why and how it works (see [10, pages 517–519] for a list of references).

Holland's schema theorem (see [4] and [3]) is often used to explain why genetic algorithms (GAs) work. For binary GAs, a schema is a string of symbols taken from the alphabet {0,1,#}. The character # is a "don't care" symbol, so that a schema can represent several bit strings. One way of creating a theory for GP is to define a concept of schema for parse trees and to extend the GA schema theorem.

Unfortunately, until very recently the efforts in this direction have given limited results. In the last few years alternative definitions of schema have been proposed [6, 12, 20]. All these definitions are based on the idea that a schema is composed of one or more trees or fragments of trees and that each schema represents all the programs in which such trees or tree fragments are present. These notions of schema have led to some theoretical results which, however, have a limited explanatory power. There is a simple explanation for this.

A schema is a subspace of the space of possible solutions, ideally represented using some concise notation (rather than enumerating all the solutions it contains). A population of strings or programs samples many sub-spaces in parallel. A schema theorem is an attempt to explain which subspaces will be sampled at the next generation. So, the crucial feature for schemata to be useful in explaining how GP searches is that their definition must make the effects of selection, crossover and mutation comprehensible and relatively easy calculate. The problem with the definitions of schema for GP mentioned above is not that they are not clear or concise, it is that they make the effects on schemata of the genetic operators used in GP too difficult to evaluate mathematically.

In recent work [18] we have reconsidered all this and proposed a new definition of schema for GP which is very close to the original concept of schema in GAs. We define a *schema* as a tree composed of functions from the set $\mathcal{F} \cup \{=\}$ and terminals from the set $\mathcal{T} \cup \{=\}$, where $\mathcal{F}$ and $\mathcal{T}$ are the function set and the terminal set used in a GP run. The symbol $=$ is a "don't care" symbol which stands for a *single* terminal or function. Therefore, a schema H represents programs having the same shape as H and the same labels for the non-$=$ nodes. For example, the schema `(AND (= x y) =)` represents the programs `(AND (OR x y) z)`, `(AND (AND x y) x)`, etc. but not `(OR (AND x y) z)` or `(AND (AND x y) (OR x z))`.

While this definition is simpler than others, it is still very difficult to model mathematically the effects on schemata of standard crossover. This prompted us to find a more natural form of crossover for GP which was mathematically in tune with our definition of schema. The similarity between our GP schemata and the original GA schemata, suggested to us a new form of crossover for GP, which we called *one-point crossover*, in which the same crossover point is selected in both parents (see Section 2). This is very similar to one-point crossover for bit strings where a common crossover point is selected in both parents and the offspring are produced by swapping the bits on the right or the left of the crossover point. We also chose a simple form of mutation, *point mutation*, in which a function in the tree is substituted with another function with the same arity or a terminal is substituted with another terminal [11].

With these genetic operators we were able to derive very naturally a schema theory [18] which has a considerable explanatory power (see Appendix A). Indeed, the predictions of the theory have been later corroborated by an experimental study [17] on the creation, propagation and disruption of GP schemata in small populations using the XOR problem. In this paper we experimentally study the performance of our genetic operators and compare them to standard GP on larger parity problems.

The paper is organised as follows. In Section 2 we describe one-point crossover as well as a new variant called *strict one-point crossover* and we discuss their properties. In Section 3, we present experimental evidence which shows that both forms of one-point crossover compare favourably with standard crossover on the even-3, 4 and 5 parity problems. Finally, we draw some conclusions and we give indications of future work in Section 4. Appendix A summarises our schema theorem for GP.

2 One-Point Crossover

One-point crossover works by selecting a common crossover point in (copies of) the parent programs and then swapping the corresponding subtrees like standard crossover. If the parents had always the same size and shape, this operation could be performed in a single stage, by selecting any link as the crossover point. However, in order to account for the possible structural diversity of the two parents, one-point crossover requires two phases:

(a) first the two parent trees are traversed to identify the parts with the same shape, i.e. with the same arity in the nodes encountered traversing the trees from the root node, then

(b) a random crossover point is selected with a uniform probability among the links belonging to the common parts identified in step (a).

Figure 1 illustrates the behaviour of one point crossover. It is worth noting how the offspring produced inherit the common structure (emphasised with thick lines) of the upper part of the parents. (One-point crossover has some similarity to the strong context preserving crossover operator proposed in [2] but context preserving crossover is less restrictive than one-point crossover as to which links can be selected as crossover points.)

One-point crossover has a very important property: it makes the calculations necessary to model the disruption of GP schemata feasible. This means that it is possible to study in detail its effects on different kinds of schemata and to obtain a schema theorem. This tells us how the GP search proceeds by predicting which areas of the search space have a high probability of being sampled by the programs in a generation given the programs in the previous one. Appendix A summarises our GP schema theorem. More on it can be found in [18] and [17]. Here we want only to recall the most important predicted and observed effect of one-point crossover: unlike standard crossover, in the absence of mutation, one-point crossover makes the population converge quite quickly like a standard GA (in some cases with help from genetic drift). The reason for this is probably that until a large-enough proportion of the population has exactly the same structure in the upper parts of the tree, the probability of selecting a crossover point in the lower parts will be very small. This effectively means that until a common upper structure is found, one-point crossover is actually searching a much smaller space of (approximately) fixed-size structures.[1] Therefore, GP behaves like a GA searching for a partial solution (i.e. a good upper part) in a relatively small search space. This means that the algorithm converges and a common upper part is quickly found, which cannot later be modified unless mutation is present. At that point the search concentrates on slightly lower levels in the tree with a similar behaviour, until level after level the entire population has completely converged. So, one-point crossover transforms a large search in the original space containing programs with different sizes and shapes into a sequence of smaller quick searches in space containing structures of fixed size and shape.

[1] Obviously the lower parts of the trees moved around by crossover influence the fitness of the fixed-size upper parts, but they are not modified by crossover at this stage.

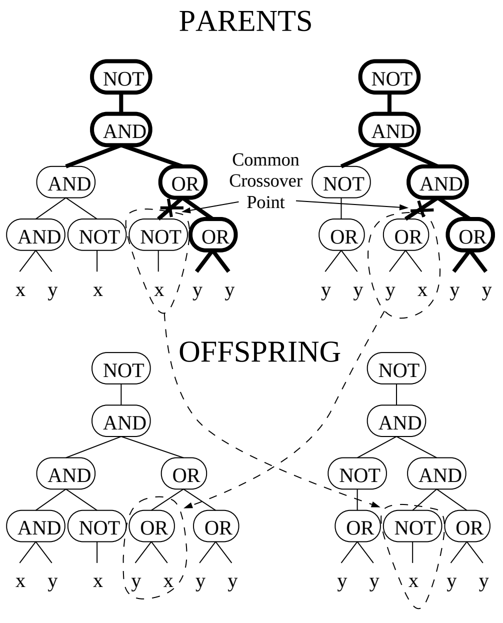

Figure 1: One-point crossover (potential crossover locations are shown in bold).

An important consequence of the convergence property of GP with one-point crossover is that that like in GAs mutation becomes a very important operator to prevent premature convergence and to maintain diversity.

In addition to the theory-related properties mentioned above, one-point crossover offers another very important property from the practice point of view: it does not increase the depth of the offspring beyond that of their parents, and therefore beyond the maximum depth of the initial random population. This is can be very useful to avoid the typical undesirable growth of program size (bloating) observed in GP runs (see for example [9]), which slows down the search for solutions and, in some cases, can lead to overfitting. One-point crossover does this without the need of any extra machinery (e.g. parsimony terms in the fitness function). Similarly, one-point crossover will not produce offspring whose depth is smaller than that of the shallowest branch in their parents and therefore than the smallest of the individuals in the initial population. This means that the search performed by GP with one-point crossover and point mutation is limited to a subspace of programs defined by the initial population. Therefore, the initialisation method and parameters chosen for the creation of the initial population can modify significantly the behaviour of the algorithm. For example, if one uses the "full" initialisation method [6] which produces balanced trees with a fixed depth, then the search will be limited to programs with a fixed size and shape. If on the contrary the "ramped half-and-half" initialisation method is used [6], which produces trees of variable shape and size with depths ranging from 0 to the prefixed maximum initial tree depth D, then the entire space of programs with maximum depth D will be searched (at least if the population is big enough).[2]

An interesting variant of one-point crossover, which we call *strict one-point crossover*, behaves exactly like one-point crossover except that the crossover point can be located only in the parts of the two trees which have exactly the same structure (i.e. the same functions in the nodes encountered traversing the trees from the root node). The links eligible as crossover points in strict one-point crossover are a subset of those eligible in standard one-point crossover. Figure 2 illustrates the behaviour of strict one-point crossover. In this case the offspring produced inherit both the structure and the nodes (emphasised with thick lines) of the upper part of the parents.

Strict one-point crossover has the same properties as one-point crossover but it more energetically forces the population to converge. This can be understood considering that until a large-enough proportion of the population has exactly the same nodes in the upper parts of the tree, the search will not be able to proceed to lower levels. Strict one-point crossover transforms the original search into a sequence of quick searches in spaces of structures of fixed size and shape which are even smaller than those used by one-point crossover.

[2] The "ramped half-and-half" initialisation method used in [6] enforced a minimum depth of 2. In our work we use a minimum depth of 0.

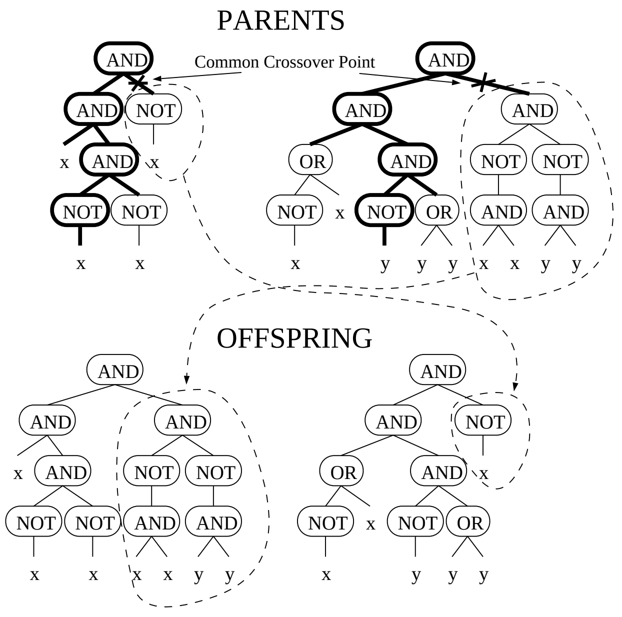

Figure 2: Strict one-point crossover (potential crossover locations are shown in bold).

In the case of strict one-point crossover the search seems to proceed very similarly to the search performed by a GA with Dynamic Parameter Encoding (DPE) [19], a technique for overcoming the precision/speed dilemma when encoding real-valued parameters with binary strings. In DPE the resolution of the encoding of one parameter is increased at run time when the most significant bit of such parameter has (nearly) converged in the whole population. A difference is that in GP with strict one-point crossover the search zooms into the subtrees of a converged node automatically, without the need for maintaining global convergence statistics.

The convergence property of the two forms of one-point crossover has been observed in real runs with the XOR problem [17]. Figure 3 shows the diversity in the population (averaged over 10 independent runs) as a function of the generation number for standard crossover and for the two types of one-point crossover in the absence of mutation (the experimental conditions are as in [17]). It is quite clear that standard crossover does not lead to convergence, while one-point crossover does.

3 Experimental Results

The behaviour of the two forms of one-point crossover introduced in the previous section has been studied and compared to standard crossover in over 3,000 runs on the even-n parity problems with n=3, 4 and 5, which have been extensively studied in the GP literature [6, 7].

An even-n parity problem consists of finding a combination of functions from the set $\mathcal{F}$=$\{$`OR, AND, NOR, NAND`$\}$ and terminals from the set $\mathcal{T}$=$\{$`x1, x2, x3, ..., xn`$\}$ which returns `true` if an even number of the n inputs `xi` is `true` and `false` otherwise. The fitness function for this class of problems is simply the number of entries of the truth table of the even-n parity function correctly represented by each program.

Given the importance of the initial population and the expected need for mutation to maintain diversity when using one-point crossover, we decided to test the performance of GP with different initial depths and different point mutation probabilities. In these experiments we used a crossover probability of 0.7, tournament selection with tournament size 7, no depth or size limit (for standard crossover only), and the "ramped-half-and-half" initialisation method. The population size was 1,000, the maximum number of generations was 50. In the tests we used the following mutation probabilities per node: p_m=0, 1/256, 1/128, 1/64, 1/32, 1/16. For the even-3 parity problem we used initial depths $D = 4$ and $D = 6$, while for the even-4 parity problem we used $D = 6$ and $D = 8$. For each combination of parameters we tried standard crossover, one-point crossover, strict one-point crossover

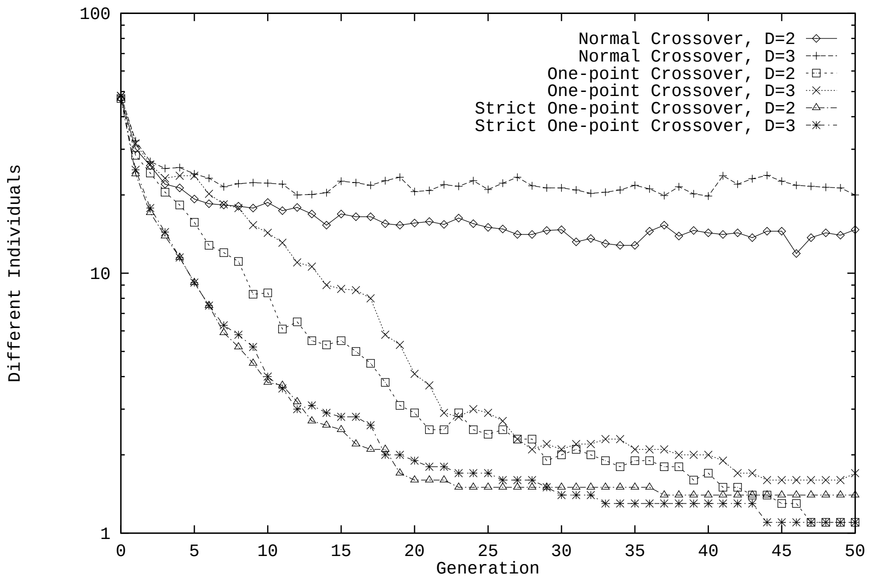

Figure 3: Plot of the number of different programs in a population of 50 individuals vs. generation number for the XOR problem. The data (averaged over 10 runs with different random seeds) for standard crossover, one-point crossover and strict one-point crossover with maximum initial depth D=2 and D =3 are show.

and mutation only (when applicable). We repeated 20 runs using different random seeds for each combination of parameters and operators.

To assess the performance of the various operators we used the computational effort E used in the GP literature (E is the minimum number of fitness evaluations necessary to get a correct program, in multiple runs, with probability 99%). We also measured the average size of the solutions found. On the even-n parity problems Koza [6, 7] obtained the results shown in Table 1 (we report them for an easier comparison with our experiments). Table 2 describes the results of our experiments.[3]

The experiments show that the maximum size of the initial population, a parameter largely ignored by the GP literature, has a considerable effect on the computational effort required to solve the problem. This is particularly true for the experiments with the two forms of one-point crossover and with point mutation only, as these operators cannot expand the search space determined by the initial population. For example, the effort for one-point crossover to find solutions to the even-3 parity problem becomes nearly 13 times smaller if the maximum depth is increased from 4 to 6.

As expected from our schema theory and the previous experiments with the XOR problem, the experiments suggest that both forms of one-point crossover suffer from premature convergence in the absence of mutation. For example, no solutions where obtained to the even-4 parity problem in the 80 runs using either form of one-point crossover.

The situation changes considerably if point mutation is present. Indeed, if the maximum depth of the initial population is appropriate and the right amount of point mutation is present one-point crossover can do up to 10 times better than standard GP without ADFs on the even-3 parity problem and up to 3 times better on the even-4 parity problem. This happens because point mutation can counteract the excessive convergence tendency of one-point crossover. As shown by Table 3, this positive effect cannot be obtained by simply reducing the selective pressure using tournament selection with tournament size 2.

Interestingly, point mutation improves performance considerably also when standard crossover is used. For example, the even-4 parity problem becomes 5 times easier with mutation rates as small as 1 node out of 256. Given that standard crossover is very disruptive and does not allow the convergence of the population, it is arguable

[3]Koza enforces a program-depth limit D=16 (considering the root node as depth 0), while we do not, uses a crossover probability of 90% rather than 70%, and selects crossover points with non-uniform probability.

that point mutation in this case helps settling into the narrow minima of the fitness function which need to be reached with small changes. These are very unlikely produced by standard crossover alone.

Point mutation performs very well on these problems even in the absence of crossover, and in some cases it outperforms GP with crossover (although these results need to be corroborated with larger numbers of runs).

In all cases there seem to be an optimal mutation probability somewhere between 1/128 and 1/32 which is problem and depth dependent. By computing the product of the average size of the solutions obtained with the best mutation rate and the mutation rate for each combination of D and n, it is possible to infer that the ideal mutation probability p_m is very close to 2 divided by the size of the tree. We checked this hypothesis on the even-3, 4 and 5 parity problems using a mutation scheme in which exactly two random nodes are mutated in each individual, i.e. in which the mutation probability is variable. The results of these experiments (averages of 20 runs) are shown in Table 4.

These results seem to suggest that a variable mutation probability is in general very beneficial, in particular for strict one-point crossover. Indeed, with variable mutation probability strict one-point crossover outperforms all other settings tried in our experiments and requires a computational effort up to 10 times smaller than for standard GP without ADFs. For the even-4 parity problem the computational effort is even smaller than for standard GP *with* ADFs.

Given the considerable effect of the maximum initial depth in all the combinations and settings of the operators used in our experiments, we decided to check the effects of the initialisation method, too. Table 5 shows the results obtained using the "full" initialisation method on the even-3 parity problem. Despite the fact that all trees have exactly the same shape (all the functions in the function set have the same arity) and that the search is much more constrained, the effects of starting from a population of larger individuals are striking: in nearly all cases the results are significantly better than those in Table 1. In particular for D=6, nearly any choice of operators and mutation rates leads to speed-ups of 10 to 15 times with respect to standard GP, the best results being obtained with variable mutation rates.

4 Conclusions

In this paper we have described two forms of crossover, one-point crossover and the new strict one-point crossover, which, thanks to constraints on the selection of crossover points, transmit to the offspring many of the common features of their parents.

These forms of crossover have several interesting properties. From the theory point of view they allow the derivation of a more explanatory schema theorem in which the effects of crossover on schemata are mathematically modelled. From the practice point of view, one-point crossover eliminates the bloating problem directly and naturally and force the population to converge like in standard GAs.

In the paper we have presented the first experimental evidence which shows that one-point crossover compares favourably with standard crossover as long as the initial population has the correct depth and premature convergence is prevented by using point mutation. Interestingly, in our study point mutation seemed a very beneficial operator also when used with standard crossover, in particular when the mutation probability was size-dependent. Surprisingly, point mutation did very well on the even-n parity problems even in the absence of crossover. Future research will be necessary to confirm these results for other classes of problems.

Acknowledgements

The authors wish to thank the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group for useful discussions and comments. This research is partially supported by a grant under the British Council-MURST/CRUI agreement and by a contract with the Defence Research Agency in Malvern.

Even-3, 4 and 5 Parity Problem (Koza's Results)

Population Size	Even-3	Even-4	Even-5
4,000, no ADFs	80,000 (N/A)	1,276,000 (N/A)	7,840,000 (N/A)
4,000, with ADFs	N/A	80,000 (N/A)	152,000 (N/A)
16,000, no ADFs	96,000 (45)	384,000 (113)	6,528,000 (300)
16,000, with ADFs	64,000 (48)	176,000 (60)	464,000 (157)

Table 1: Computational effort E and average solution size (in parenthesis) for standard GP with and without ADFs reported by Koza in [6, 7].

Even-3 Parity Problem

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
4	0	52,000 (45)	308,000 (24)	No Solution	N/A
4	1/256	39,000 (63)	264,000 (27)	810,000 (31)	810,000 (31)
4	1/128	48,000 (38)	110,000 (27)	1,320,000 (31)	396,000 (31)
4	1/64	32,000 (46)	170,000 (26)	315,000 (29)	399,000 (29)
4	1/32	31,000 (42)	128,000 (27)	105,000 (30)	147,000 (31)
4	1/16	54,000 (54)	96,000 (26)	133,000 (28)	86,000 (30)
6	0	24,000 (86)	24,000 (64)	270,000 (88)	N/A
6	1/256	16,000 (88)	27,000 (77)	42,000 (77)	54,000 (92)
6	1/128	15,000 (101)	18,000 (72)	28,000 (75)	44,000 (86)
6	1/64	8,000 (98)	10,000 (81)	8,000 (87)	25,000 (79)
6	1/32	16,000 (100)	14,000 (77)	9,000 (87)	21,000 (92)
6	1/16	19,000 (82)	42,000 (67)	26,000 (67)	28,000 (80)

Even-4 Parity Problem

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
6	0	1,276,000 (148)	No Solution	No Solution	N/A
6	1/256	238,000 (215)	638,000 (109)	725,000 (119)	880,000 (111)
6	1/128	216,000 (168)	507,000 (98)	357,000 (112)	220,000 (122)
6	1/64	195,000 (154)	224,000 (114)	136,000 (105)	198,000 (116)
6	1/32	611,000 (193)	598,000 (91)	360,000 (111)	510,000 (83)
6	1/16	No Solution	No Solution	No Solution	No Solution
8	0	812,000 (271)	No Solution	No Solution	N/A
8	1/256	126,000 (396)	189,000 (296)	196,000 (313)	319,000 (370)
8	1/128	120,000 (382)	140,000 (296)	129,000 (302)	144,000 (359)
8	1/64	170,000 (377)	144,000 (287)	154,000 (275)	105,000 (210)
8	1/32	1,131,000 (188)	329,000 (112)	500,000 (127)	385,000 (151)
8	1/16	No Solution	No Solution	No Solution	No Solution

Table 2: Computational effort E and average solution size (in parenthesis) as a function of the genetic operators used, the mutation probability and the maximum depth of the initial programs.

Even-4 Parity Problem (Tournament Size = 2)

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
6	0	697,000 (181)	No Solution	No Solution	N/A
6	1/128	No Solution	4,320,000 (105)	2,024,000 (87)	No Solution
8	0	1,392,000 (439)	3,330,000 (199)	No Solution	N/A
8	1/128	1,363,000 (361)	3,780,000 (115)	1,421,000 (265)	No Solution

Table 3: Computational effort E and average solution size as a function of the genetic operators used, with and without point mutation, for two different maximum depths of the initial programs when the tournament size is reduced to 2.

Even-3 Parity Problem (Variable Mutation Probability)

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
4	2/Size	48,000 (61)	44,000 (28)	51,000 (30)	64,000 (31)
6	2/Size	12,000 (105)	11,000 (92)	8,000 (101)	8,000 (106)

Even-4 Parity Problem (Variable Mutation Probability)

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
6	2/Size	156,000 (283)	210,000 (121)	99,000 (119)	144,000 (127)
8	2/Size	108,000 (518)	168,000 (334)	78,000 (353)	196,000 (328)

Even-5 Parity Problem (Variable Mutation Probability)

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
8	2/Size	Not tested	Not tested	1,232,000 (489)	Not tested
9	2/Size	Not tested	Not tested	730,000 (660)	Not tested

Table 4: Computational effort E and average solution size (in parenthesis) as a function of the genetic operators used and the maximum depth of the initial programs in the presence of variable point-mutation probability.

Even-3 Parity Problem (“Full” Initialisation)

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
4	0	28,000 (45)	80,000 (31)	220,000 (31)	N/A
4	1/256	19,000 (42)	70,000 (31)	78,000 (31)	572,000 (31)
4	1/128	16,000 (41)	48,000 (31)	121,000 (31)	147,000 (31)
4	1/64	14,000 (43)	63,000 (31)	36,000 (31)	84,000 (31)
4	1/32	18,000 (40)	48,000 (31)	22,000 (31)	31,000 (31)
4	1/16	33,000 (52)	39,000 (31)	30,000 (31)	26,000 (31)
4	2/Size	34,000 (66)	40,000 (31)	54,000 (31)	42,000 (31)
6	0	7,000 (129)	18,000 (127)	30,000 (127)	N/A
6	1/256	10,000 (131)	6,000 (127)	7,000 (127)	12,000 (127)
6	1/128	8,000 (134)	6,000 (127)	6,000 (127)	8,000 (127)
6	1/64	6,000 (133)	6,000 (127)	5,000 (127)	7,000 (127)
6	1/32	8,000 (127)	6,000 (127)	6,000 (127)	5,000 (127)
6	1/16	10,000 (128)	8,000 (127)	9,000 (127)	7,000 (127)
6	2/Size	5,000 (140)	7,000 (127)	5,000 (127)	5,000 (127)

Table 5: Computational effort E and average solution size as a function of the genetic operators used and the maximum depth of the initial programs when the “full” initialisation method is used.

A Schema Theorem for Genetic Programming with One-point Crossover and Point Mutation

In order to understand the importance of one-point crossover from the theory point of view it is necessary to introduce some additional definitions (see [18] and [17] for a more details on our schema theory). The number of non-$=$ symbols in a schema H is called the *order* $\mathcal{O}(H)$ of the schema, while the total number of nodes in the schema is called the *length* $N(H)$ of the schema. The number of links in the minimum subtree including all the non-$=$ symbols within a schema H is called the *defining length* $\mathcal{L}(H)$ of the schema. For example, the schema `(AND (= y =) x)` has order 3 and defining length 3.

Our GP schema theorem provides the following lower bound for the expected number of individuals sampling a schema H at generation $t+1$ for GP with one-point crossover and point mutation:

$$\begin{aligned} E[m(H,t+1)] \quad \geq \quad & m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{\mathcal{O}(H)} \times \\ & \left\{1 - p_c\left[p_{\text{diff}}(t)\left(1-\frac{m(G(H),t)f(G(H),t)}{M\bar{f}(t)}\right) + \right.\right. \\ & \left.\left. \frac{\mathcal{L}(H)}{(N(H)-1)}\frac{m(G(H),t)f(G(H),t)-m(H,t)f(H,t)}{M\bar{f}(t)}\right]\right\} \end{aligned}$$

where $m(H,t)$ is the number instances of the schema H in the population at generation t, $f(H,t)$ is the mean fitness of the instances of H, $\bar{f}(t)$ is the mean fitness of the programs in the population, p_c is the crossover probability, $E[\cdot]$ is the expected-value operator, p_m is the mutation probability (per node), $G(H)$ is the zero-th order schema with the same structure of H where all the defining nodes in H have been replaced with "don't care" symbols, M is the number of individuals in the population, $p_{\text{diff}}(t)$ is the conditional probability that H is disrupted by crossover when the second parent has a different shape (i.e. does not sample $G(H)$). The zero-order schemata $G(H)$'s represent different groups of programs all with the same shape and size. For this reason we call them *hyperspaces* of programs. We denote non-zero-order schemata with the term *hyperplanes*, as they can be seen as sub-spaces of the spaces of programs identified by different $G(H)$'s.

Our schema theorem is more complicated than the corresponding version for GAs [3, 4, 21]. This is due to the fact that in GP the trees undergoing optimisation have variable size and shape. This is accounted for by the presence of the terms $m(G(H),t)$ and $f(G(H),t)$, which summarise the characteristics of the programs belonging to the same hyperspace in which H is a hyperplane. However, both the theoretical analysis presented in [18] and the experimental work in [17] suggest that after a first phase in which GP really behaves differently from a standard GA, the number of hyperspaces is considerably reduced and GP behaves like a GA, i.e. the GP schema theorem asymptotically tends to the GA schema theorem.

References

[1] Peter J. Angeline and K. E. Kinnear, Jr., editors. *Advances in Genetic Programming 2*. MIT Press, Cambridge, MA, USA, 1996.

[2] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[4] John Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, second edition, 1992.

[5] K. E. Kinnear, Jr., editor. *Advances in Genetic Programming*. MIT Press, 1994.

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

[7] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Pres, Cambridge, Massachusetts, 1994.

[8] John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[9] W. B. Langdon and R. Poli. Fitness causes bloat. Technical Report CSRP-97-09, University of Birmingham, School of Computer Science, Birmingham, B15 2TT, UK, 24 February 1997.

[10] William B. Langdon. A bibliography for genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter B, pages 507–532. MIT Press, Cambridge, MA, USA, 1996.

[11] Ben McKay, Mark J. Willis, and Geoffrey W. Barton. Using a tree structured genetic algorithm to perform symbolic regression. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 487–492, Sheffield, UK, 12-14 September 1995. IEE.

[12] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[13] Riccardo Poli. Evolution of recursive transistion networks for natural language recognition with parallel distributed genetic programming. Technical Report CSRP-96-19, School of Computer Science, University of Birmingham, B15 2TT, UK, December 1996. Presented at AISB-97 workshop on Evolutionary Computation.

[14] Riccardo Poli. Genetic programming for image analysis. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 363–368, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[15] Riccardo Poli. Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. In *3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97*, 1997.

[16] Riccardo Poli and Stefano Cagnoni. Evolution of psuedo-colouring algorithms for image enhancement with interactive genetic programming. Technical Report CSRP-97-5, School of Computer Science, The University of Birmingham, B15 2TT, UK, January 1997. To be presented at GP-97.

[17] Riccardo Poli and W. B. Langdon. An experimental analysis of schema creation, propagation and disruption in genetic programming. Technical Report CSRP-97-8, University of Birmingham, School of Computer Science, February 1997. To be presented at ICGA-97.

[18] Riccardo Poli and W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. Technical Report CSRP-97-3, School of Computer Science, The University of Birmingham, B15 2TT, UK, January 1997. To be presented at GP-97.

[19] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9(1):9–21, 1992.

[20] P. A. Whigham. A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 178–181, Perth, Australia, 29 November - 1 December 1995. IEEE Press.

[21] Darrel Whitley. A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University, August 1993.