

Smooth Uniform Crossover with Smooth Point Mutation in Genetic Programming: A Preliminary Study

J. Page, R. Poli and W. B. Langdon
School of Computer Science,
University of Birmingham,
Birmingham B15 2TT, UK
{J.Page,R.Poli,W.B.Langdon}@cs.bham.ac.uk

Abstract. In this paper we examine the behaviour of the uniform crossover and point mutation GP operators [12] on the even- n -parity problem for $n = 3, 4, 6$ and present a novel representation of function nodes, designed to allow the search operators to make smaller movements around the solution space. Using this representation, performance on the even-6-parity problem is improved by three orders of magnitude relative to the estimate given for standard GP in [5].

1 Introduction

Although a mutation operator is defined, the canonical form of Genetic Programming (GP) [4] relies almost exclusively on the crossover operator for exploring the solution space. GP crossover selects a random subtree from one parent program and splices it to a random location in another, affording GP the ability to search a space of arbitrary-sized programs. Its insensitivity to position in cutting and splicing (aside of satisfying constraints on tree depth), combined with the fact that multiple instances of function nodes are generally distributed throughout each tree, make it extremely unlikely that a function will be eliminated from the population altogether. In contrast to genetic algorithms (GAs), GP does not generally require a mutation operator to reintroduce information eliminated by selection, and many GP runs use crossover alone.

More recently, however, crossover has come under a certain amount of criticism. Since the standard form is applied to tree-based representations, most randomly selected crossover points are located towards the leaves, an effect that becomes increasingly pronounced as tree size increases [8] [12]. Furthermore, trees tend to increase rapidly in size as a GP run progresses – a phenomenon referred to in the literature as bloat (e.g. [4], [1]). Soule and Foster [16] argue that this bias towards selecting lower branches as crossover points further exacerbates the increase in program size.

The tendency of crossover to exchange subtrees located towards the leaves led Poli to argue [9] and prove [12] that GP crossover is essentially a local search operator, sampling at best only points in the immediate vicinity of the solution space occupied by the parent donating the root node. In contrast, most GA

crossover operators combine genetic material from each parent in roughly equal quantities. Early in a GA run, when genetic diversity is high, two randomly selected parents will probably be quite different from each other, with the result that the offspring may little resemble either. As the population begins to converge, an increasing number of locations on the bitstring will be identical in both parents and the offspring produced are therefore genotypically quite similar. We may therefore say that GA crossover moves from being a global to a local search operator as the run progresses. This is a desirable property since global search early in the run allows an efficient exploration of the search space and an identification of promising areas, whilst local search later in the run affords the fine-tuning of imperfect solutions. GP crossover, however, rapidly becomes a local search operator such that individuals behave essentially as local hill-climbers and the population thus become susceptible to convergence on local optima.

These observations led Poli and Langdon to develop two GA-inspired crossover operators for GP: one-point crossover [10] [11] [13] and uniform crossover [12]. Poli and Langdon have presented theoretical results indicating that, given a sufficiently diverse initial population, both operators proceed from a global to local exploration of the search space as the run progresses [12].

In this paper, we build on the work reported in [12] by introducing a novel representation of function nodes that allows GP uniform crossover (GPUX) to perform more finely-grained, directed movements around the solution space. We refer to GPUX operating on this representation as *smooth uniform crossover* (GPSUX), and the representation itself as *sub-symbolic node representation*. The remainder of the paper is organised as follows. The next section describes uniform crossover, and a mutation operator - point mutation, in detail. We then describe the sub-symbolic node representation and the manner in which uniform crossover acts upon it. We compare GPUX, GPSUX and an implementation of standard GP on the even-3-parity, even-4-parity and even-6-parity induction problems and show that GPSUX performs extremely favourably. We conclude with a discussion of these results and suggest some future work.

2 Uniform Crossover and Point Mutation

GP Uniform crossover (GPUX)[12], as the name suggests, is a GP operator inspired by the GA operator of the same name [17]. GA uniform crossover (GAUX) constructs offspring on a bitwise basis, copying each allele from each parent with a 50% probability. Thus the information at each gene location is equally likely to have come from either parent and on average each parent donates 50% of its genetic material. The whole operation, of course, relies on the fact that all the chromosomes in the population are of the same structure (i.e. linear bit strings) and the same length. No such assumption can be made in GP since the parent trees will almost always contain unequal numbers of nodes and be structurally dissimilar.

GP uniform crossover [12] begins with the observation that many parse trees are at least partially structurally similar. This means that if we start at the root

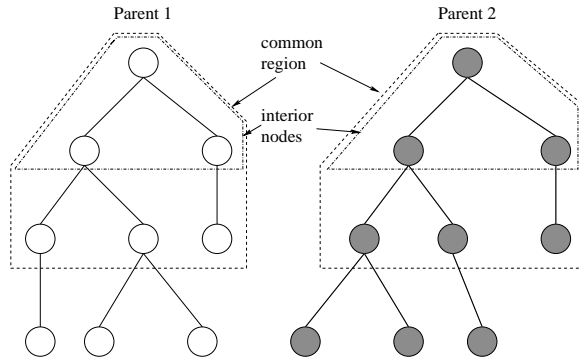


Fig. 1. Two parental parse trees prior to uniform crossover

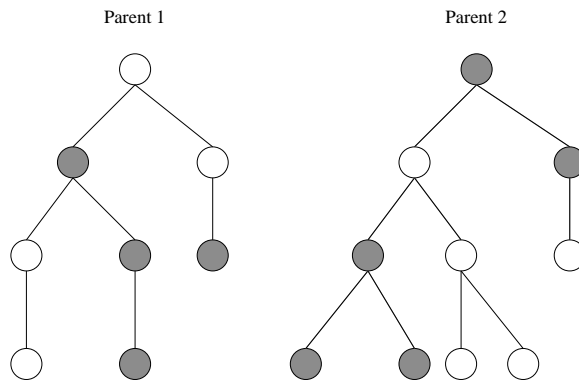


Fig. 2. Offspring trees after uniform crossover

node and work our way down each tree, we can frequently go some way before finding function nodes of differing arity at the same locations. Furthermore we can swap every node up to this point with its counterpart in the other tree without altering the structure of either. Working down from the root node, we can define two regions of a pair of trees as follows. Any node in one tree having a corresponding node at the same location in the other is said to be located within the *common region*. Those pairs of nodes within the common region that have the same arity are referred to as *interior*. The interior nodes and common region of two trees is illustrated in figure 1. Note that the common region necessarily subsumes the interior nodes. The uniform crossover algorithm is then as follows. Once the interior nodes have been identified, the parent trees are both copied. Interior nodes are selected for crossover with some probability p_c . Crossover involves exchanging the selected nodes between the trees, with those nodes not selected for crossover remaining unaffected. Non-interior nodes within the common region can also be crossed, but in this case the nodes and their subtrees are swapped. Nodes outside the common region are not considered.

As in GA uniform crossover, the value of p_c is generally set to 0.5, resulting in an exchange of 50% of the nodes. The result of uniform crossover applied to the trees in figure 1 is shown in figure 2.

GPUX, like GAUX, is a homologous operator, that is it preserves the position of genetic material in the genotype. As a result of sampling error, this can in both cases lead to the phenomenon of lexical convergence whereby a sub-optimal gene becomes fixed at a given location. When this happens, crossover cannot introduce the optimal gene and for this reason it is generally desirable to include a mutation operator to maintain diversity in the population. The operator we use here – GP point mutation (GPPM)[7] – is also inspired by its GA counterpart (GAPM). GPPM substitutes a single function node with a randomly selected replacement of the same arity. As in GAPM, the number of mutations performed on an individual is a function of the program size and a user-defined mutation rate parameter. Since in GP program lengths vary, this means that larger programs undergoing mutation will, on average, be perturbed to a greater degree than smaller ones. Since such perturbations are generally detrimental to the fitness of a highly-evolved program, this will generate an emergent parsimony pressure [10].

3 Sub-Symbolic Node Representation and Smooth Operators

Whilst a single point mutation is the smallest syntactical operation that can be applied to a parse tree under the standard representation, it may nevertheless result in a significant change in behaviour. For example, consider the following subtree: (AND T_1 T_2) where T_1 and T_2 are Boolean input terminals. If the AND node is replaced with NAND (and bear in mind that with the standard Boolean function set of {AND, OR, NAND, NOR} the probability of this is $\frac{1}{3}$, assuming the mutation operator replaces the original with a different node), the value returned by the subtree will be altered in all the fitness cases. Controlling this means addressing the mechanism used to replace the node. Our solution is simple. We begin by noting that a Boolean function of arity n can be represented as a truth table (bit-string) of length 2^n , specifying its return value on each of the 2^n possible inputs. Thus AND may be represented as 1000, OR as 1110. We refer to this representation as *sub-symbolic* because the representation, and hence the behaviour, of a function node can be modified slightly during the course of a GP run. For example, flipping a single bit will alter the behaviour of the node for just one of its possible input combinations.

We can define a point mutation operator which works in exactly this manner – a single randomly-selected bit is flipped in a single randomly-selected node. Since GPUX is homologous, we can extend it to use a GA crossover operator on the nodes at reproduction (in the experiments reported here we use GAUX). The crossover operation is illustrated in figure 3. When a pair of interior nodes are selected for crossover, GA uniform crossover is applied to their binary representations. In other words, the bits specifying each node’s function are swapped

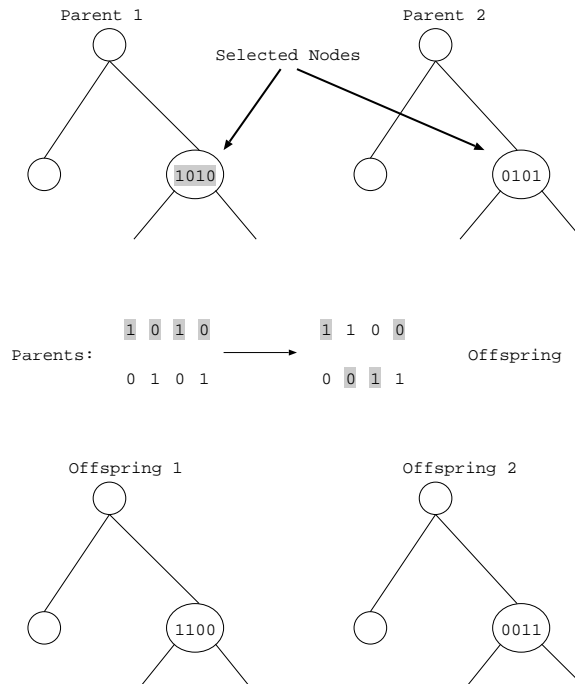


Fig. 3. Uniform crossover on the sub-symbolic representation. In this example, the two central bits of the selected nodes are exchanged to generate the offspring

with probability 0.5. Clearly such an operator interpolates the behaviour of the parents' corresponding nodes, rather than exchanging nodes in their entirety. The sub-symbolic node representation allows GP to move around the solution space in a smoother, more controlled manner and hence we refer to these versions of the operators as *smooth uniform crossover* (GPSUX) and *smooth point mutation* GPSPM.

One feature of the sub-symbolic representation of Boolean function nodes is that it necessarily incorporates all 2^n nodes of arity n into its function set. This is in contrast to the reduced function set normally used for Boolean classification problems. We further discuss the effects of this in the next section.

4 Experimental Set-up

In this paper we demonstrate an implementation of GPSUX and GPSPM on the even-3-, even-4- and even-6-parity problems. Our function set was restricted to diadic Boolean functions. Unconstrained, the GPSUX and GPSPM operators can potentially generate any of the 16 arity 2 Boolean functions and for this reason we included them all in the standard representation function set. Such a set will undoubtedly contain a number of extraneous functions (always-on and

always-off being the most obvious examples), although what effect, if any, they have on performance is poorly understood. Rosca [15] notes that increasing the size of the function set from 4 to 8 increases the fitness diversity of randomly generated trees on the even-5-parity problem, but that this effect is slightly reduced when the size is further increased to 16 functions. Koza [4] examined the effects of extraneous functions on a number of problems including the 6-multiplexer and found performance using set sizes of less than 6 to be superior to that using larger sets, with the complete set {NAND} performing best of all. However, the choice of additional function is likely to be of crucial importance. For example, the addition of the EQ (even-2-parity) function to the standard set of {AND, OR, NAND, NOR} dramatically improves standard GP’s performance on the even-6-parity problem [6].

The other parameters of our runs are summarised in Table 1. Pilot studies suggested that the sub-symbolic representation was quite capable of solving the problem with a population of just 50, but we used 200 too to give standard GP a fighting chance. Note that every individual in the population was subjected to mutation (although an elitist strategy was used in the event of a decrease in fitness of the best of generation individual) and that mutation probability in the case of point mutation therefore refers to the probability that a single node will be mutated. In the case of standard mutation, a single random subtree was spliced to a random node of each tree with a probability of 0.1. At 0.3, the crossover rate is significantly lower than that used in most standard GP runs and is based on the results of a number of pilot studies using the sub-symbolic representation, and on evidence from the literature suggesting that low crossover and high mutation rates are beneficial when small populations are used in GP [3]. It was used in all the conditions to ensure validity of comparison. Finally, the initial tree depth value of 7 counts the root node as node 0 such that, with ramped half-and-half population initialisation, the largest individual in the initial population contained 511 nodes.

Population sizes	50 and 200
Crossover probability	0.3
Mutation probability	0.1
Tournament size	7
Max. initial tree depth	7
Initialisation method	ramped
Max. tree depth	17
Maximum generations	80
Number of runs per condition	50

Table 1. Experimental Parameters for even- n -parity problems

5 Results

We compared performance of standard GP without ADFs, GP using the GPUX and GPPM operators and GP using the GPSUX and GPSPM operators on the even- n -parity problem for $n = 3, 4, 6$. We would have liked to have studied GPUX and GPPM in isolation, but the lexical convergence associated with GPUX means that some mutation is necessary to maintain population diversity. However, we did examine the GPPM and GPSPM operators in isolation for the standard and fine-grain representations. Tables 2, 3 and 4 show the minimum effort (fitness evaluations) required to find a solution with 99% probability [4], the average complexity (number of nodes) of the correct solutions, and the percentage of runs on which a correct solution was found for each of the three problems.

Operators	Population size	Fitness evaluations	Complexity	Success Rate
Standard GP	50	5,550	742	32%
Standard GP	200	2,400	156	76%
GPUX, GPPM	50	850	67	92%
GPUX, GPPM	200	2,000	66	100%
GPPM only	50	1,050	72	98%
GPPM only	200	1,400	59	100%
GPSUX, GPSPM	50	900	72	94%
GPSUX, GPSPM	200	600	43	100%
GPSPM only	50	600	56	100%
GPSPM only	200	600	48	100%

Table 2. Minimum effort, average complexity of correct solutions and percentage of runs on which a correct solution was found for the even-3-parity problem

Given the small population sizes, the unusual parameter settings and the test function, it is unsurprising that standard GP failed to find a solution on the even-6-parity problem – its poor performance on this problem is well known [5] [2]. Koza reports that the effort required for GP *with* ADFs to solve the even-6-parity problem is 1,344,000 [5]. Gathercole and Ross [3] did not estimate the effort required for their limited-error fitness approach to solving the task, but did report that it required 2500 generations to find a solution with a population size of 400 (i.e. 1,000,000 individuals evaluated). Standard GP’s performance on the even-3-parity and even-4-parity problems, however, compares favourably with other results reported using the standard function set of {OR, AND, NOR, NAND}. Koza [5] reports efforts of 96,000 and 384,000 fitness evaluations for the even-3-parity and even-4-parity problems respectively. We believe this to be due to the facilitating effect engendered by the inclusion of the EQ and XOR functions [6]. The other conditions fared considerably better and compare favourably with

Operators	Population size	Fitness evaluations	Complexity	Success Rate
Standard GP	50	11,250	428	12%
Standard GP	200	19,000	568	68%
GPUX, GPPM	50	4,200	84	88%
GPUX, GPPM	200	6,000	49	98%
GPPM only	50	4,200	68	80%
GPPM only	200	6,000	49	98%
GPSUX, GPSPM	50	2,250	82	92%
GPSUX, GPSPM	200	2,200	56	100%
GPSPM only	50	2,250	76	98%
GPSPM only	200	8,400	49	98%

Table 3. Minimum effort, average complexity of correct solutions and percentage of runs on which a correct solution was found for the even-4-parity problem

Operators	Population size	Fitness evaluations	Complexity	Success Rate
Standard GP	50	No solution found	N/A	0%
Standard GP	200	No solution found	N/A	0%
GPUX, GPPM	50	34,850	38	36%
GPUX, GPPM	200	127,600	40	60%
GPPM only	50	35,550	43	44%
GPPM only	200	71,400	51	82%
GPSUX, GPSPM	50	17,000	49	62%
GPSUX, GPSPM	200	19,200	53	80%
GPSPM only	50	16,200	59	67%
GPSPM only	200	18,400	42	82%

Table 4. Minimum effort, average complexity of correct solutions and percentage of runs on which a correct solution was found for the even-6-parity problem

other algorithms reported in the literature, particularly on the even-6-parity problem.

The performance of GPUX with GPPM seems to be 3 to 5 times better than that of standard GP. We believe this to be due to the ability of the GPUX operator to perform global search [12].

The complexity of the solutions is also surprisingly low. Since GPUX, GPPM, GPSUX and GPSPM cannot increase tree depth beyond that of the largest tree created at initialisation, the bloating observed in the standard GP runs cannot occur. Nevertheless, the average solution sizes are all well short of the maximum possible (511 nodes in this case), confirming that mutating on a per node basis generates a counter-pressure against increased program size, since parsimonious solutions are still found in the absence of the uniform crossover operator.

Clearly, the sub-symbolic node representation with GPSUX and GPSPM fur-

ther enhances the performance of GP, typically by a factor of 2 on the even-4 and even-6-parity problem. We believe that this is because this allows GP to make smoother movements around the solution space, although further analytical work is necessary to confirm this.

The GPPM and GPSPM operators also performed well on their own. In many conditions mutation alone performed the same as or better than when it was used with crossover. However, pilot studies on higher-order versions of the parity problem suggested that better performance could be achieved using uniform crossover as well, indicating that the performance of mutation alone is mainly a result of the relative simplicity of the lower-order parity problems (for a discussion of the work on higher-order parity problems see [14]).

6 Conclusions

In this paper we have demonstrated the application of two alternative search operators for GP – point mutation and uniform crossover. We have also described a simple representation for function nodes and shown that the combined approach produces a considerable improvement in performance on the even- n -parity problem, particularly for $n = 6$. This improvement appears to be due to a combination of two factors: the global search property of the GPUS operator and the finer granularity of the sub-symbolic representation which allows GP to make smaller, directed movements around the solution space. Of these, the former has been established theoretically [12]. Further work is required to confirm the latter.

Of course, we have only demonstrated the success of the sub-symbolic node representation on a single problem. Furthermore, certain regularities associated with the parity problems may have contributed to the encouraging results reported here. However, we are confident that the approach can be generalised to other problems and other representations. If our assumptions concerning the benefits that can be accrued from increasing the granularity of the search space and allowing the operators to make use of this increase are correct, there is good reason to expect that the approach will also succeed when applied to other problems. Future work will address this.

References

1. Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).
2. Chris Gathercole and Peter Ross. The MAX problem for genetic programming - highlighting an adverse interaction between the crossover operator and a restriction on tree depth. Technical report, Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN, UK, 1995.

3. Chris Gathercole and Peter Ross. Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
4. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
5. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.
6. W. B. Langdon and R. Poli. Boolean functions fitness spaces. In R. Poli, P. Nordin, W. B. Langdon, and T. Fogarty, editors, *Proceedings of the Second European Workshop on Genetic Programming – EuroGP'99*, Goteborg, May 1999. Springer-Verlag.
7. Ben McKay, Mark J. Willis, and Geoffrey W. Barton. Using a tree structured genetic algorithm to perform symbolic regression. In A. M. S. Zalzal, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALEZIA*, volume 414, pages 487–492, Sheffield, UK, 12-14 September 1995. IEE.
8. 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, page 573, Perth, Australia, 29 November - 1 December 1995. IEEE Press.
9. Riccardo Poli. Is crossover a local search operator? Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97, 20 July 1997.
10. Riccardo Poli and W. B. Langdon. Genetic programming with one-point crossover. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 180–189. Springer-Verlag London, 23-27 June 1997.
11. Riccardo Poli and W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 278–285, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
12. 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.
13. Riccardo Poli and William B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.
14. Riccardo Poli, Jonathan Page, and William B. Langdon. Solving even-12, -13, -15, -17, -20 and -22 boolean parity problems using sub-machine code GP with smooth uniform crossover, smooth point mutation and demes. Technical Report CSRP-99-2, University of Birmingham, School of Computer Science, January 1999.
15. Justinian P. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, University of Rochester, Rochester, NY 14627, February 1997.

16. Terence Soule and James A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pages 781–186, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
17. G. Syswerda. Uniform crossover in genetic algorithms. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.