DIRECTING CROSSOVER FOR REDUCTION OF BLOAT IN GP

M. D. Terrio and M. I. Heywood
Faculty of Computer Science, Dalhousie University, Halifax NS, Canada
{mterrio@cs.dal.ca | mheywood@cs.dal.ca}

Abstract

A method is proposed to reduce the amount of inviable code (or bloat) produced in individuals while searching for a parsimonious solution under tree structured genetic programming. Known as directed crossover, this process involves the identification of highly fit nodes to use as crossover points during operator application. Three test problems, including medical data classification, are used to assess the performance of directed crossover when applied at various thresholds. Results, collected over 1260 independent runs, identify conditions under which directed crossover reduces code bloat.

Keywords: *Code bloat; directed crossover.*

1.Introduction

1.1 Tree Based Genetic Programming

Originally introduced in 1992 by Koza [1], Genetic Programming is a generic data driven machine learning technique, providing solutions in the form of a computer program that satisfies some predefined criterion of success for a set of inputs. The solution program is typically represented by the best one of all the individual chromosomes, which constitute the population of a GP run. Unlike Genetic Algorithms, where individuals are made up of fixed length binary strings, individuals in GP are variable in length and are often represented by tree like structures. Thus, when genetic operators such as crossover and selection are applied, the lengths of individual genomes are often changed.

One possible outcome is a continuous, uncontrollable increase in the length of individuals, known as “bloat” [2]. Because individual fitness calculation is the most computationally expensive process of a GP trial, the presence of large individuals naturally detracts from the efficiency of the GP method [3]. In addition, bloated individuals will then stagnate the overall cycle of evaluate-select-modify. Finally, as increases in resource allocation are often not proportional to performance increase [4], it is evident that the current GP process is less than optimal.

1.2 Introns and Inoperative Code

One of the chief contributors to code bloat are introns or nocs - segments of code that make no contribution to the functional completeness of the individual [5]. In explaining their existence, Soule refers to both a removal bias, created by the difference in importance between the subtrees added and removed during crossover, and the Destructive Hypothesis [4]. The basis of the Hypothesis states that code growth occurs to preserve vitally important sections of code from the potentially destructive effects of applying genetic operators [6]. In a similar, more general theory, Miller and McPhee state that a Replication Accuracy Force (RAF) is at work in GP, which causes evolution to favor programs which replicate with semantic accuracy [7]. The result of the force is a general flow towards deeper programs.

Regardless of its origin, it is a widely held view that the removal of redundant code and the subsequent generation of a parsimonious solution is one of the keys to improving GP. To accomplish this, methods such as parsimony pressure, code modification and operator modification have been suggested and tested with varying results [4,8,9]. In addition to these works, complexity based fitness measures have been incorporated into GP runs [3,10] to overcome the fact that simple Koza style GP incorporates fitness functions which fail to consider parsimony.

In our work, we investigate a strategy for implementing chromosome crossover at points determined *a priori* to their selection, hereafter referred to as directed crossover. The objective being to alleviate some of the pitfalls of bloat while not hampering the generation of a parsimonious solution.

2.Method

2.1 Tree-Structured GP

GP utilizes Darwin’s notion of survival of the fittest to instigate a competition between individuals in the population. Search operators are then used to provide exploitation and exploration of the solution space. A search operator determines how individuals propagate between populations [1].

In this work, steady state tournament selection is employed, in which four individuals from the population are sampled, without bias for fitness. From here, they are sorted in order of descending fitness with the best two being singled out as the representative parents for potential operator application. Next, sequential/stochastic tests are made for applying crossover and mutation. In the case of applying crossover, a further test is made regarding crossover type - directed or standard. The decision to apply crossover or mutation is made against thresholds (c.f. classical GP) where as the type of crossover is determined as a percentage (e.g. 50:50, 40:60, etc.). Once again, this percentage is selected stochastically using a uniformly distributed probability function. However, for runs involving medical data, the final stages of this process are slightly altered.

Table 1: Functional and terminal set of problems used in directed crossover trials.

Problem Name	Functional Set	Terminal Set
Symbolic Regression	{+,-,*,%,sin,cos,exp,rlog}	X
Two Boxes	{+,-,*,%}	L_0,W_0, H_0, L_1,W_1, H_1
Medical Classification	{+,-,*,%,sin,cos,sqrt}	Problem Dependant

Specifically, we reinsert only one of the two chromosomes produced by crossover back into the population. The chromosome, to which the shorter subtree is adjoined during crossover, serves this purpose, while the other child is a duplication of the parent from which this subtree originates. This helps to further minimize the potential for extreme bloat caused when index points are identified at depths, whose difference is significant. In other words, the possibility of joining a large subtree to a tree already large in terms of node count is reduced.

2.2 Directed Crossover

As previously mentioned, the application of directed crossover is put into play with the objective of investigating shorter, more parsimonious solutions. The specifics of the process involve the identification of individual nodes whose contribution to the overall fitness of the individual is maximal. In order to do so, the fitness of each individual is measured and recorded at each individual node, regardless of whether it is a functional or terminal. The number of the node whose fitness value exceeds that of all others is tagged as being the "index" node for that chromosome. It was our hypothesis that using these index values as crossover points for the individuals would aid in reducing bloat by exchanging highly operative segments of code from one individual to another. Figure 1 portrays this idea.

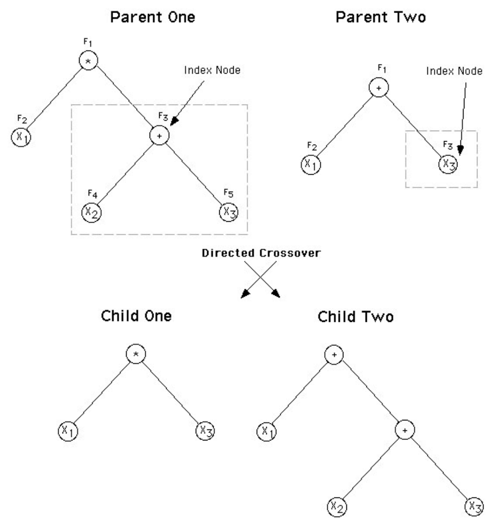

Fig. 1. The process of directed crossover.

The application of this process is conducted in two separate groups of tests. In the first, directed crossover always takes place at the "index" points of the chromosomes. We refer to this as "biased directed crossover". In the second, a roulette wheel is used to allow for the possibility of crossing over at nodes other than the index node. In this way, weight is given to nodes in relation to the percentage of their fitness, compared to that of the overall individual. In a sense, it represents a mix between a greedy and a purely stochastic incorporation of the crossover information. We refer to this type as "roulette-style directed crossover".

3.Testing Process

3.1 Parameters

In all cases a modification of [11] was used to implement our GP process. In each test, a population of 500 individuals was initialized using the ramped half and half technique [1]. The makeup of individuals created contained nodes of functionals and terminals (inputs) in relation to those listed in Table I. In keeping with the consistency of all test runs, the rate of crossover was set at 90%, while probability of mutation of individuals was set at 50%. For the majority of the runs conducted, trial runs continued until an optimal solution was found or until a limit of 50,000 tournaments was reached. For the remaining runs, involving medical data classification, this parameter was set at 15,000 as fitness convergence was found to occur quite rapidly. The only other parameter that was altered throughout was the rate of directed crossover (2.2).

3.2 Test Problems

The testing process involved applying the directed crossover process to three different types of test problems. The first two, symbolic regression and the two boxes problem, are commonly used in GP research. The third involved the generation of solutions to correctly classify the presence of Liver Disease and Breast Cancer in individuals, using benchmark medical data sets [12].

For symbolic regression, the goal follows that found in [13]. That is, we were attempting to find a solution to fit to the x and y values of 20 pairs of data points for the equation $y=x^4+x^3+x^2+x$ where x was uniformly sampled over the interval [-1,1]. Our fitness function incorporated the sum-squared error (SSE) of the output of the chromosome being evaluated [1,13]. Tournament selection and subsequent operator application continued to take place until an individual was evaluated where the number of “hits”, or data points mapped with less than 0.01 error, equaled 20. The sum squared error was also calculated at each individual node and recorded until all 20 data points had been tested. Once this was complete, the node of minimal SSE was marked as the “index” node for that individual. Work on the two boxes problem also involved minimizing the sum squared error. However, in searching for solutions to the difference between the volumes of two boxes ($L_1H_1W_1$-$L_2H_2W_2$), we used a success predicate of 10 hits while sampling from the set of integer values {1,...,10}. Identification of the index node was also done in a similar fashion.

In addition, performance under real world data sets was evaluated using the two benchmark medical classification data sets mentioned above. The first data set comprised of 699 patterns, constituting 9 columns of input and a single column of classification values related to the diagnosis of Breast Cancer. The second, related to Liver Disease, and contained 345 patterns of 6 inputs in addition to a single classification value. In both cases the classification value was a Boolean which pertained to the existence (1) and nonexistence (0) of the disease. Using this data, it was our overall goal to discover solutions which would produce classifications in resemblance to those given in the data. To do so, we set out by designating 75% of each data set as a training set, with the remaining 25% allocated for a test set. Fitness was calculated as the percentage of input patterns in the training set correctly classified by an individual. Because the probability of finding a solution that correctly classified 100% of the training set was next to none, our trials had no termination criteria with the exception of the 15,000-tournament limit previously mentioned. The last descendant chromosome generated after this time (which we deemed representative of the entire population) was then introduced to the test set in order to see how well it would perform with new data.

4.Results

The gathering of results involved repeating tests for different rates of both biased, and roulette-style directed crossover. Probabilistic intervals of 25% were used in hopes of determining a pattern in the ratio of directed vs. standard crossover.

For both the symbolic regression problem and the two boxes problem, 50 runs were conducted at biased directed crossover probability levels of 0 (no directed crossover), 25, 50, 75, and 100 percent. Similar parameters were also used for roulette-style directed crossover. Table 2 displays results derived from these tests.

The above process was then duplicated for medical classification on the Breast and Liver data sets. However, time constraints dictated that only 20 runs were made. Results tabulated from these runs are also found in Table 2

In measuring the effect of directed crossover on the given suite of problems, we identified a number of key statistics to incorporate into our comparative analysis. The two most pertinent were the average number of nodes found in chromosomes generated within an entire set of 50 runs, and the average length of the solutions evolved. In general, we correlate the node count as an indication of code bloat, while solution length acts as a measure of solution parsimony.

From the information portrayed in Table 2, it is obvious that directed crossover was detrimental when applied to the symbolic regression problem. This fact is especially evident when biased directed crossover was used. With minor exceptions, values for median computational effort, average solution length, and average number of nodes continuously increased as the threshold value grew. Furthermore, the number of solutions found as the threshold values increased, rapidly declined. While the application of roulette style directed crossover was not as harmful, results were still not considered promising.

In the two boxes problem, varying levels of success were met. When applied 75% of the time, runs involving biased directed crossover showed optimal results. At this level, the number of convergent solutions (18) far exceeded any of the corresponding values at other threshold levels. While the median computational effort value (2548) was also optimal, the values for average solution lengths

Table 2: Results of all runs conducted.

Parameter	**Rate of Biased Crossover**					**Rate of Roulette Crossover**			
	0.0	**.25**	**.5**	**.75**	**1.0**	**.25**	**.5**	**.75**	**1.0**
Symbolic Regression									
Number of convergent solutions	48	44	38	28	2	**47**	47	50	44
Average number of tournaments	5462	6668	12080	17431	13593	**4610**	7547	5324	6860
Median computational effort of all solutions (*10^4)	120	163	312	845	7,274	**108**	160	107	157
Average length of all solutions	15.1667	16.1818	18	18.3571	15.5	**14.6808**	15.2765	16.18	16.9772
Average number of nodes in chromosomes of all 50 runs	8.0743	9.7362	12.6599	13.8857	11.8025	**7.95847**	9.19127	8.221289	10.4835
Two Boxes									
Number of convergent solutions	2	0	6	**18**	9	1	2	0	2
Average number of tournaments	1435	No soln's	23635	**30730**	27295	13763	9577	No soln's	19591
Median computational effort of all solutions (*10^4)	9690	No soln's	6479	**2548**	5095	12549	6511	No soln's	12309
Average length of all solutions	15	No soln's	17	**26.7777**	20.7777	19	22	No soln's	24
Average number of nodes in chromosomes generated in all 50 runs	3.3983	3.3981	5.16154	**11.7577**	21.2237	3.782	3.266811	3.23599	3.4685
Breast Cancer Data									
Median classification accuracy of final solutions on test data	.9827	.9855	.9827	.9856	**.977**	.977	.9885	.9741	.9885
Median classification accuracy of final solutions on training data	.9494	.9675	.9666	,9666	**.9456**	.9627	.9637	.9465	.9599
Average length of all solutions	26.15	20.35	25.95	22.3	**15.5**	24.95	20.95	23.25	22.75
Average number of nodes in chromosomes of all 20 runs	16.09	14.93	19.16	17.59	**13.82**	17.52	16.42	18.70	16.75
Liver Disease Data									
Median classification accuracy of final solutions on test data	.6627	.6511	.6395	.6337	**.6511**	.6570	.6570	.6511	.6628
Median classification accuracy of final solutions on training data	.7200	.7374	.7297	.7162	**.7123**	.6969	.7085	.7142	.6757
Average length of all solutions	24.85	32.65	24.5	24	**12.9**	26.66	26.45	32.75	37.85
Average number of nodes in chromosomes generated in all 20 runs	16.68	20.87	16.53	17.83	**13.12**	19.62	16.78	19.74	20.10

(26.77) and average number of chromosomes (11.7577) were anything but. As was the case in runs on symbolic regression, these values increased with increasing levels of biased directed crossover. In terms of results for the application of roulette style directed crossover, most figures were quite not significant, as the number of solutions found at each level was quite small. However, it was a positive sign to see that average solution length and average number of nodes remained constant throughout. Thus, it seems that success in the two box problem comes at the expense of increased bloat. While biased crossover led to far more solutions than any other method, the nature of these solutions were not parsimonious.

Results for the medical data classification problems painted a different picture. In viewing the data given in Table 2, high thresholds of biased directed crossover led to optimal results for both node count and average solution length. For breast data classification, the average number of nodes in all chromosomes produced when using biased directed crossover was 13.82. The corresponding figure for liver data classification was 13.12. Both values represent the minimum of all other values for node count amongst the runs completed at each threshold level. Code bloat was therefore minimized using our biased directed crossover method 100% of the time.

Further support for the benefits resulting from the application of directed crossover came from the average length of solutions found at each threshold level. Once again, optimal values were found when biased directed crossover was applied at a rate of 100%. (15.5 for breast and 12.9 for liver). While roulette style directed crossover did not contribute further to our case, we believe the results show sufficient evidence as to the benefits of directed crossover.

5. Conclusions

From the results, it is obvious to see that directed crossover was detrimental in both the symbolic regression and two boxes problem. On the other hand, the reduction of code bloat and solution length were clearly evident when directed crossover was used in the classification domain.

While the effectiveness of directed crossover was clearly shown, the results produced on the medical data sets may partly be accredited to the reduction in the tournament limit from 50,000 (as in regression and two boxes) to 15,000. In preliminary testing, we observed a tendency for solution fitness to converge somewhere in the range of seven to ten thousand tournaments. Additional fitness increases after this time were few and far between, thus provoking us to decrease the longevity, and therefore the computational effort required for the runs. However, this in itself may have aided the reduction of code bloat. It is our belief that the probability of the root node being identified as the index node (maximal fitness) for directed crossover increased once fitness plateaued. This, in turn, caused rapid increase in tree size as the subtrees being joined were themselves entire individuals. In a preventative measure, we modified the process to only reinsert the child node to which the smallest subtree was attached (section 2.1). However, in future work, we would like to explore different avenues.

More specifically, we hope to modify the process such that on reaching a fitness plateau, a different functional set is employed for the remaining tournaments. By applying different functional operators, such as ‘if” statements and ADF’s [14], the objective would be to not only emphasize further problem solving, but also code reuse.

Finally, rather than crossing over at nodes of maximal fitness, we plan to implement processes that take into account the changes in fitness from one node to the next.

6.Acknowledgements

Mr. Terrio would like to take this opportunity to thank Omid Banyasad for his assistance in the formatting of this paper. Additionally, the authors would both like to express their appreciation to NSERC for its ongoing support of our research endeavours.

7.References

[1] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[2] W. Banzhaf, P. Nordin, R. E. Keller, & Frank D. Francone, *Genetic Programming: An Introduction*. San Fransisco, CA: Morgan Kaufmann, 1998.

[3] B. Zhang, H. Muhlenbein, “Adaptive Fitness Functions for Dynamic Growing/Pruning of Program Trees,” *Advances in Genetic Programming*, vol. 2, pp.241-255, 1996.

[4] T. Soule, “*Code Growth in Genetic Programming,*” Ph. D. dissertation, University of Idaho, May 1998.

[5] Daida et al., “What makes a Problem GP-Hard? Analysis of a tunably Difficult Problem in Genetic Programming,” *Genetic Programming and Evolvable Machines*, vol. 2, pp. 165-191, 2001.

[6] P. Nordin, W. Banzhaf, “Complexity compression and evolution,” *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 310-317, 1995.

[7] N.F. McPhee, J.D. Miller, “Accurate Replication in Genetic Programming,” *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp.303-309, 1995.

[8] P. W. H. Smith, K. Harries, “Code Growth, Explicitly Defined Introns, and Alternative Selection Schemes,” *Evolutionary Computation*, vol. 6, no. 4, pp 339-360, 1999.

[9] P.W.H. Smith, K.Harries, “Exploring Alternative Operators and Search Strategies in Genetic Programming,” *Genetic Programming 1997: Proceedings of the Second annual Conference*, pp. 147-155, 1997.

[10] H. Iba, H. de Garis, T. Sato, “Genetic Programming Using a Minimum Description Length Principle,” *Advances in Genetic Programming*, vol. 1, pp. 265- 284, 1994.

[11] A. Singleton, GPQUICK: A Simple Genetic Programming System in C++. *Available at:ftp://ftp.krl.caltech.edu/pub/EC/GP/src/gpquick-1.2.tar.gz*,

[12] Universal Problem Solver Inc., Machine Learning Data Sets. *http://www.upso.net/td1_frames.html*.

[13] K. Chellapilla, “Evolving Computer Programs Without Subtree Crossover,” in *IEEE Transactions On Evolutionary Computation*, vol. 1, no. 3, pp. 209-216, September 1997.

[14] J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable SubPrograms*. Cambridge MA: MIT Press, 1994.