CONTROLLING THE PROBLEM OF BLOATING USING STEPWISE CROSSOVER AND DOUBLE MUTATION TECHNIQUE

Arpit Bhardwaj[#1] Aditi Sakalle[*2] Harshita Chouhan[#3] Harshit Bhardwaj[*4]

[#]Computer Science, Information Technology RGPV, RGPV
Khandwa India, Khandwa India
arpitgsits17@gmail.com

harshichouhan18nov@gmail.com

*Electrical and Electronics, Computer Science RGPV, RGPV
Khandwa India, Khandwa India
aditi.sakalle@gmail.com

harshit.cs@gmail.com

ABSTRACT

During the evolution of solutions using genetic programming (GP) there is generally an increase in average tree size without a corresponding increase in fitness—a phenomenon commonly referred to as bloat. The conception of "bloat" in Genetic Programming is a well naturalized phenomenon characterized by variable-length genomes gradually maturating in size during evolution. "In a very real sense, bloating makes genetic programming a race against time, to find the best solution possible before bloat puts an effective stop to the search." In this paper we are proposing a Stepwise crossover and double mutation operation in order to reduce the bloat. In this especial crossover operation we are using local elitism replacement in combination with depth limit and size of the trees to reduce the problem of bloat substantially without compromising the performance. The use of local elitism in crossover and mutation increases the accuracy of the operation and also reduces the problem of bloat and further improves the performance. To shew our approach we have designed a Multiclass Classifier using GP by taking few benchmark datasets.

KEYWORDS

Bloat, Stepwise Crossover, Double mutation, Elitism, Fitnes.

1. INTRODUCTION

Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. GP exploits Darwin's concept of evolution and survival of the fittest to develop computer programs. It is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task. It is one of the most common forms of variable-length evolutionary computation. The goal of GP is usually to find a fitting individual for a given application. It is introduced to evolve computer programs automatically. Multiclass or multi-label classification is the special case within statistical classification of assigning one of

DOI : 10.5121/acij.2011.2606

several class labels to an input object. An approach takes an integrated view of all classes when the GP evolves. A Multitree representation of chromosomes is used. GP is a method of solving problems using computers through an analogue of natural selection. A way to have computers automatically solves problems, without having to define or even know the form or structure of the solution ahead of time. It is a domain independent, stochastic method with an important ability to represent programs of arbitrary size and shape. During the evolution of solutions using Genetic Programming there is generally an increase in average tree size without a corresponding increase in fitness—a phenomenon commonly referred to as bloat [1].

Code bloat is the uncontrolled growth of program size that may occur in GP when relying on a variable length representation. This has been identified as a key problem in GP for which there have been several empirical studies. However, very few theoretical studies addressed this issue directly. Although code bloat is not clearly understood, it is yet possible to distinguish at least two kinds of code bloat. We first define structural bloat as the code bloat that necessarily takes place when no optimal solution can be approximated by a set of programs with bounded length. In such a situation, optimal solutions of increasing accuracy will also exhibit an increasing complexity (larger programs), as larger and larger code will be generated in order to better approximate the target function. Another form of bloat is the functional bloat, which takes place when program length keeps on rowing even though an optimal solution (of known complexity) does lie in the search space. Three main methods for controlling bloat are commonly proposed: set an upper bound to the complexity of individuals in the population, introduce an explicit fitness penalty (parsimony measure) that biases against larger individuals [4], and apply genetic operators designed to target redundant code or that bias against offspring size increases [6]. Recently, there have been a number of theoretical advances in understanding bloat [3], [7], which have confirmed that selection pressure is one fundamental driver in bloat creation.

In our paper we have proposed a special type of tournament known as Triple tournament. In Triple tournament first we randomly selects 9 individual from the population and from those selected individual we select 5 on the basis of size and from those 5 individual we select three on the basis if depth and from those 3 we select the 2 individual on the basis of fitness.

In this we also propose a stepwise crossover technique and a double mutation operation that reduces the destructive nature of conventional genetic operations. Mutation affects an individual in the population. It can replace a whole node in the selected individual, or it can replace just the node's information. Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as help to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability. Double Mutation operates on a single individual from the population. Mutation operations often end up decreasing the fitness of an individual because the new material is untested. They tend to be used less than crossover; however, they are still important to a successful GP run. In double Mutation technique we generate the two individual from the selected parent and from the two generated individual we reject one individual on the basis of fitness and the remaining individual is now compared with the parent if the fitness of the parent is better than the selected individual than with a probability of 0.5 we transfer the parent to the next generation. Stepwise crossover enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. An individual which has high fitness rate and low size and depth must be inherited.

The paper is structured as follows: Section II describes the code bloat in Genetic Programming, Section III describes the previous work to control the bloat, Section IV

describes about the theoretical concept of StepWise Crossover and Double Mutation technique, its algorithm and the experimental results.

2. BLOATING IN GENETIC PROGRAMMING

Bloating presents a serious problem in scaling GP to larger and more difficult problems. First, code bloating consumes computing resources, hampering the ability of GP to breed and discover better solutions. Second, bloated candidate solutions are often more larger than necessary. Just like in a wide variety of arbitrary-length representations, including neural networks, finite state automata, and rule sets, the difficulty in GP has highlighted roadblock faced: the candidate individuals being considered tend to grow larger and larger, without corresponding increases in fitness. This phenomenon called code bloating or code growth. Bloat is both distracting, since the reasons for it have not yet been clearly understood, and it obstructs the search, because it forces the GP algorithm into stagnation.

While bloat is well-defined and can be identified, there are currently no consensual explanations on why it occurs. Indeed, three popular theories can be found in the literature to explain it [9]:

– The introns theory states that bloat acts as a protective mechanism in order to avoid the destructive effects of operators once relevant solutions have been found. Introns are pieces of code that have no influence on the fitness: either sub-programs that are never executed, or subprograms which have no effect;

– The fitness causes bloat theory relies on the assumption that there is a greater probability to find a bigger program with the same behavior (i.e. semantically equivalent) than to find a shorter one. Thus, once a good solution is found, programs naturally tend to grow because of fitness pressure. This theory states that code bloat is operator-independent and may happen for any variable length representation based algorithm. As a consequence, code bloat is not to be limited to population-based stochastic algorithm (such as GP), but may be extended too many Algorithms using variable length representation.

– The removal bias theory states that removing longer subprograms is more dangerous to do than removing shorter ones (because of possible destructive consequence), so there is a natural bias that benefit to the preservation of longer programs. While it is now considered that each of these theories somewhat capture part of the problem there has not been any definitive global explanation of the bloat phenomenon. At the same time, no definitive practical solution has been proposed that would avoid the drawbacks of bloat (i.e. increasing evaluation time of large trees) while maintaining the good performances of GP on difficult problems. Some common solutions rely either on specific operators e.g. size-fair crossover or different fair mutation on some parsimony-based penalization of the fitness or on abrupt limitation of the program size such as the one originally used by Koza. Also, some multi-objective approaches have been proposed. Some other more particular solutions have been proposed but are not widely used yet.

3. PREVIOUS WORK DONE IN THE FIELD

In this section we present the work done by the various authors previously to control the problem of bloat.

Whigham [1], has presented an implicit model of bloat control based on a spatially-structured population with local elitism; referred to as SS+E. Regular spatial structures (such as a ring or torus) maintain diversity and slow bloat by effectively reducing the population size. In addition, elitism reduces the growth of introns, especially once the population has largely converged and cannot easily find fitness improvements. Previous panmictic models with elitism found that this resulted in crossover largely becoming a copying operator, resulting in convergence to non optimal solutions. Most bloat control methods tradeoff controlling size and fitness, however SS+E appears to balance this tradeoff without compromising overall fitness.

Langdon & Poli [2] has describe the way to control bloat Using a fix size or depth limit (LGP) in which the bloat is controlled by applying the limit to the allowed individual size or depth simply. Individuals exceeding the limits are removed from the population. Because individual size or depth is calculated easily during evaluation, this approach only requires relatively little additional computation.

Stringer [5], has handled bloat by explicitly setting an upper bound on the depth of evolved trees or by incorporating a parsimony pressure that adjusts the fitness of individuals by a tradeoff between performance and size.

Bleuler, Brack, Thiele, and Zitzler proposed a nonparametric method, Double Tournament [8], this method is similar to a multi objective approach to bloat, however the objectives of fitness and size are treated separately. Hence, there are two tournaments: one based on parsimony, which produces an initial set of winners, and a subsequent tournament that selects a subset of those winners based on fitness.

Sara Silva and Ernesto Costa[11] present two important variations on a recently successful bloat control technique, Dynamic Maximum Tree Depth, intended at further improving the results and extending the idea to non *tree-based GP*. Dynamic Maximum Tree Depth introduces a dynamic limit on the depth of the trees allowed into the population, initially set with a low value but increased whenever needed to accommodate a new best individual that would otherwise break the limit. The first variation to this idea is the Heavy Dynamic Limit that, unlike the original one, may fall again to a lower value after it has been raised, in case the new best individual allows it. The second variation is the Dynamic Size Limit, where size is the number of nodes, instead and regardless of depth. The variations were tested in two problems, Symbolic Regression and Parity, and the results show that the heavy limit performs generally better than the original technique, but the dynamic limit on size fails in the Parity problem. The possible reasons for success and failure are discussed.

In RHC the original population, termed the resident population, is copied to produce a second population, and named the visitor population. Each member of the resident population is randomly paired with a member of the visitor population and genetic operators are applied to produce a child. If the child is at least as fit as the resident parent then the resident parent is replaced by the child. RHC was compared with standard GP for the symbolic regression and Sante Fe ant problem, and showed that RHC produced more accurate, robust and simpler solutions.

Soule and Foster [10], introduced the concept of removal bias, arguing that neutral branches of code (i.e., introns) are likely to be small, however their replacement with crossover does not have this restriction. Hence, the children produced from neutral crossover events are likely on average to increase in size. In this paper they describes that the initial population are likely to be small but introns grows on increasing in size after crossover operation and the size of the individual can be very large so to restrict the size of the individual two forms of nondestructive crossover (NDC) were presented: a child would replace a parent if it was at least as fit as the parent, or in the strict version the child had to exceed the parent's fitness. These methods were tested with a maze navigation problem and a parity problem, with both examples showing a reduction in bloat and an improvement in convergence to fit solutions. However, since crossover is often destructive, strict elitism can reduce the effectiveness of crossover as a search mechanism, especially once the population has begun to converge.

4. PROPOSED WORK

To demonstrate our approach we have designed a Multiclass Classifier using Step Wise Crossover and double Mutation technique.

4.1. INITIALIZATION

In genetic programming first we have to generate initial population for performing operations in it. The terminal and function sets are important components for generating initial population. The terminal and function sets are the alphabet of the programs to be made. The terminal set consists of the variables and constants of the programs. The functions are several mathematical functions, such as addition, subtraction, division, multiplication and other more complex functions.

Each of the trees for each individual is initialized randomly using the function set F which consists of arithmetic functions and the terminal set T containing feature variables and constants. The function set F and terminal set T used here are as follows:
F = {+, -, *, %} and
T = {feature variables, R}
Where R contains randomly generated constants in [0.0 to 10.0]. We have initialized trees using the ramped half-and-half method

4.2. FITNESS MEASURE

The most difficult and most important concept of genetic programming is the fitness function. The fitness function determines how well a program is able to solve the problem. GP is guided by the fitness function to search for the most efficient computer program to solve a given problem. A simple measure of fitness has been adopted for the pattern classification problem.

Fitness = n/N

n= Samples classified correctly
N= Number of samples used for training during evolution

4.3. SELECTION

We select the parent for crossover on the basis of triple tournament selection method. For mutation the parent is selected randomly and for reproduction we use roulette wheel selection method. In double mutation we select one best child on the basis of fitness and reject another one. In stepwise crossover we select best child and remove remaining child step by step.

4.4. ALGORITHM FOR TRIPLE TOURNAMENT

1. First we randomly select 9 individual from the population.
2. Now on the basis of depth we select 5 out of 9 individual the individual with smaller depth are selected.
3. Now on the basis of size we eliminate 2 individual from 5 and the individual with smaller size are selected.
4. Now finally from these 3 individual we select two individual with higher fitness.

4.5. DOUBLE MUTATION

Double Mutation is a special technique in which we randomly selected an individual from the population and from this randomly selected individual we generate the two

children. From this two generated children one is rejected on the basis of fitness the children with the lower fitness is rejected now we compare the selected individual with the parent and compare its fitness with the parent if the fitness of parent is better than the child than with a probability of 0.5 parent is transferred to the next generation otherwise children is transferred to the next generation. By applying this double mutation technique we are sure that the generated individual does not reduce the fitness and also provide the diversity among the individuals.

4.6. ALGORITHM FOR DOUBLE MUTATION METHOD

1. First we generate the two children from the randomly generated individual.
2. Reject the one individual on the basis of fitness the individual with greater fitness are select.
3. Now we apply the elitism we compare the selected individual with parent and if the fitness of parent is better than the child than with a probability of 0.5 we transfer the parent to the next generation otherwise we transfer the child to the next generation.

4.7. STEPWISE CROSSOVER

The two individual which are selected from the triple tournament performs the Stepwise Crossover. The two individual will generate the six children. From these six generated children two are rejected on the basis of depth and size, children with the smaller depth and size are selected. Now from these 4 selected children eliminate two on the basis of fitness, children with the greater fitness are chosen. Now apply the elitism on the generated children and compare the fitness of the children with the parent if the fitness of the parent is greater than the children than with the probability of 0.5 parent is transferred to the next generation otherwise children is transferred to the next generation.

4.8. ALGORITHM FOR STEPWISE CROSSOVER

1. Randomly select individual from the population for triple tournament selection.
2. Select the best two individuals of the triple tournament for the crossover operation.
3. Randomly select the subtree or a node from a parent and place it at three different positions in another parent and generate the three children and the same process is repeated with another parent also, so the total number of generated children are six.
4. Now select the best 4 children from these 6 child on the basis of depth and size the children with the smaller depth and size are chosen.
5. Finally the two children are chosen on the basis of fitness the individual with the higher fitness are selected.
6. Now we apply the elitism on the selected individual if the fitness of the parent is better than the child than we transfer the parent with a probability of 0.5 to the next generation otherwise we transfer the child to the next generation.

4.9. GP Algorithm with Step Wise Crossover and Double Mutation

1. GP begins with a randomly generated population of solutions of size N.
2. A fitness value is assigned to each solution of the population.
3. A genetic operator is selected probabilistically.

4. If it the reproduction operator, then an individual is selected (we use fitness proportion based selection) from the current population and it is copied into the new population. Reproduction replicates the principle of natural selection and survival of the fittest.
5. If it is the crossover operator, then we apply the Step Wise Crossover.
6. If the selected operator is mutation, we apply a double mutation.
7. Continue 3. Until the new population gets solutions.
8. This completes one generation.
9. Step 3 to 7are repeated till a desired solution is achieved. Otherwise, terminate the GP operation after a predefined number of generations.

5. EXPERIMENTAL RESULTS

We have designed a MultiClass Classifier to demonstrate our results. We have used Java 6.0 as a front end tool and Oracle 10g as a back end tool to develop our project. We have used 4 real data sets for training and validating our methodology. These are IRIS, WBC, BUPA, WINE, ABALONE, SPOKEN ARABIC DIGIT, HILL-VALLEY. Table I gives a brief description about all the data sets used.

a. Data Sets

1) IRIS Data: This is the well-known Anderson's Iris dataset. It contains a set of 150 measurements in four dimensions taken on Iris flowers of three different species or classes.The classes are Iris Setosa, Iris Versicolour and Iris Virginica. The four features are sepal length, sepal width, petal length, and petal width. The data set contains 50 instances of each of the three classes.
2) Wisconsin Breast Cancer (WBC): This data set has 683 data points distributed in four classes. Each data point is represented by ten attributes. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
3) BUPA Liver Disorders (BUPA): It consists of 345 data points in six dimensions distributed into two classes on liver disorders. The first 5 variables are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption.
4) WINE: This dataset contains 178 instances and 13 attributes. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
5) ABALONE**:** This dataset predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem. It has 4177 instances and 8 features.
6) SPOKEN ARABIC DIGIT: This dataset has 8800 classes and 13 attributes. This dataset contains time series of mel-frequency cepstrum coefficients (MFCCs) corresponding to spoken Arabic digits.Includes data from 44 male and 44 female native Arabic speakers. Dataset from 8800(10 digits x 10 repetitions x 88 speakers) time series of 13 Frequency Cepstral Coefficients (MFCCs) had taken from 44 males and 44 females Arabic native speakers between the ages 18 and 40 to represent ten spoken Arabic digit.
7) HILL-VALLEY: This dataset contains 606 instances and 101 attributes. Each record represents 100 points on a two-dimensional graph. When plotted in order (from 1 through 100) as the Y co-ordinate.

Table 1. Datasets

Name of Data Set	**No of classes**	**No of Features**
IRIS	3	4
WBC	2	9
BUPA	2	6
WINE	3	13
ABALONE	4177	8
SPOKEN ARABIC DIGIT	8800	13
HILL-VALLEY	606	101

b. PARAMETERS

Table II describes the common parameters used for all the data sets.

Table 2. Common Parameters for all datasets

Parameters	**Values**
Probability of Crossover Operation	75%
Probability of Reproduction Operation	10%
Probability of Mutation Operation	15%
Population Size	50-500
Minimum Tree Depth	2
Maximum Tree Depth	6
Number of Generations	10-60

c. RESULTS

To demonstrate our approach we have designed a MultiClass Classifier using genetic programming. The training and testing of the classifier generated is done using real data sets. We have compared the outcome of our results with the conventional crossover and mutation method shown in Table III and found that our method outperforms the conventional crossover and mutation method and improves the accuracy of the classifier with a fair amount and it also helps to reduce the problem of code bloating. The overall size of the generated classifier is almost reduce to half with the help of proposed methods as compared with other methods to generate the classifier due to which the problem of bloat is reduced.

Table 3. Comparison of Conventional Crossover and Mutation Method with Stepwise Crossover and Double Mutation Method

NAME OF DATASETS	**Conventional Crossover and Mutation Method**		**Stepwise crossover and Double Mutation Method**	
	Training Accuracy	Generalization Accuracy	Training Accuracy	Generalization Accuracy
IRIS	83.89%	80.46%	93%	91.25%
WBC	80.24%	78.36%	87.89%	86.11%
BUPA	82.14%	80.58%	85.4%	83.11%
WINE	86.47%	84.55%	90.78%	88.47%
ABALONE	76.24%	72.45%	79.14%	76.48%
SPOKEN ARABIC DIGIT	62.89%	60.14%	69.47%	65.79%
HILL-VALLEY	85.15%	81.82%	91.74%	89.45%

6. CONCLUSION

In this paper, we have proposed a Triple Tournament technique to select the parent for the crossover operation which helps in improving the overall fitness and applying the limit of size and depth on the individual. We are also using a unique double Mutation technique in which we are generating two children and applying the fitness limit, elitism on the selected individual. We also proposed a Stepwise crossover technique in which we generate the six child from two parent and step wise eliminate the four child on the basis of size, depth and fitness and apply the elitism on the two remaining children and compare with parent. To demonstrate our approach we have designed a MultiClass Classifier and presented the results on different datasets and found that the overall size of the generated classifier is very small than the size of the classifier generated with conventional method. To verify our approach, we have compared our crossover method with the conventional crossover and mutation method and obtained impeccable results.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] Peter A. Whigham, Member IEEE, and Grant Dick, Member IEEE, "Implicitly Controlling Bloat in Genetic Programming," IEEE Transactions on Evolutionary Computation, Vol. 14, No. 2, APRIL 2010.

[2] R. Poli, "Genetic Programming for image analysis," in Proc. 1st Int. Conf. Genetic Programming, Stanford, CA, July 1996,pp. 363– 368.

[3] D.Agnelli, A. Bollini, and L. Lombardi, "Image classification: an evolutionary approach," Pattern Recognit. Lett., vol.23, pp. 303–309,2002.

[4]. H. Stringer and A. Wu, "Bloat is unnatural: An analysis of changes invariable chromosome length absent selection pressure," University of Central Florida, Tech. Rep. CS-TR-04-01, 2004.

[5] H. Stringer and A. Wu, "Winnowing wheat from chaff: The chunking GA," in Proc. Genet. Evol. Comput. (GECCO ' 04) Part II, vol. 3103. Seattle, WA: Springer-Verlag, Jun. 26–30, 2004, pp. 198-209.

[6] C. Skinner, P. J. Riddle, and C. Triggs,"Mathematics prevents bloat," in Proc. 2005 IEEE Congr. Evol. Comput., vol. 1. Edinburgh, U.K.: IEEE Press, Sep.2–5, 2008, pp. 390–395.

[7] S. Bleuler, M. Brack,L. Thiele, and E.Zitzler, "Multiobjective Genetic Programming : Reducing bloat using SPEA2," in Proc. 2001 Congr .Evol. Comput. (CEC '01), Piscataway, NJ: IEEE Press, 2007, pp.536–543.

[8] Zhang B.T., and M¨uhlenbein H., "Balancing accuracy and parsimony in genetic programming," Evolutionary Computation, vol. 3, no. 1, pp. 17-38, 1995.

[9] Soule T., Foster J. A., and Dickinson J., "Code growth in geneticProgramming,", Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press, pp. 215-223, 1996.

[10] W. Banzhaf, P. Nordin, R. E.Keller, and F. D. Francone, Genetic Programming: An Introduction. San Mateo, CA: Morgan Kaufmann, 1998.

[11] Sara Silva and Ernesto Costa, Dynamic Limits for Bloat Control Variations on Size and Depth K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 666–677, 2004.