Code Bloat Problem in Genetic Programming

Anuradha Purohit*, Narendra S. Choudhari, ArunaTiwari****

* Computer Technology and Applications Deptt., S.G.S.I.T.S. Indore
** Computer Engineering Deptt., IIT Indore

***Abstract*-** The concept of "bloat" in Genetic Programming is a well-established phenomenon characterized by variable-length genomes gradually increasing in size during evolution [1]. Bloat hampers the efficiency and ability of genetic programming for solving problems. A range of explanations have been proposed for the problem of bloat, including destructive crossover and mutation operators, selection pressure and individual representation. Different methods to avoid bloat and to control bloat have been proposed by researchers. This paper proposes a theoretical analysis of code bloating problem and the discussion on the work already done by various authors to handle bloat in genetic programming.

***Index Terms*-** Bloat, Double Tournament, Elitism, Genetic Programming, Spatial Structure.

I. INTRODUCTION

During the evolution of solutions using Genetic Programming (GP) there is generally an increase in average tree size and depth without a corresponding increase in fitness—a phenomenon commonly referred to as bloat [1]. Bloat is a well-known phenomenon in Genetic Programming. An individual program in GP could be of any size. Such flexibility in representation provides more freedom in searching solutions, but at the same time it causes the bloat problem, individuals growing unnecessarily large. Apparently big individuals are computationally more expensive to evaluate during evolution. If they are final solutions, then their execution time in applications would increase accordingly. That would not be desirable for situations where speed is a requirement such as in real-time systems. Furthermore bloat makes these evolved programs even more difficult to comprehend [17].

Code bloating presents a serious problem in scaling GP to larger and more difficult problems. First, bloat consumes computing resources, making the search process slower and slower, and eventually forcing it to stop when all available resources have been exhausted. Second, bloated candidate solutions are often more difficult to modify in meaningful ways, hampering the ability of GP to breed and discover better solutions. Third, bloating can slow the grade-assessment process. 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 [29].

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 [10]; and apply genetic operators designed to target redundant code or the bias against offspring size increases [12]. Many authors have shown a number of theoretical advances in understanding bloat [3], [9]–[12]. Poli [9] reduced bloat by a stochastic approach to setting the fitness of above average-sized individuals to zero. Stringer and Wu [10], [11] showed that a shrinking effect on genome length occurred for a chunking GA once the population had essentially converged and selection had become random. Skinner et al. [12] provided a theoretical argument for this observed tendency of variable length genomes to shrink when selection is not considered (i.e., under the process of genetic drift). The paper implied that the presence of a lower absorbing boundary (a genome size that once reached cannot be reduced further), combined with no upper bound, results in a reduction of the average size of a population under drift. Although this theoretical model contributes to an understanding of individual size dynamics, the concept has not at present formed the basis for new bloat control methods. Related to these results other research [13] has shown that for large, discrete programs, fitness convergence of the population is possible and has been used to explain sub quadratic growth of program size.

In this paper, we have analyzed and presented the problem of code bloat in GP, its types and variants and the effective measures taken by various authors to prevent or control bloat. The rest of the paper is organized as follows: section II contains the description of the code bloat problem, section III discuss the work done by various authors to prevent bloat and section IV presents different methods present in the literature to avoid code bloating in GP.

II. CODE BLOAT IN GENETIC PROGRAMMING

"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". While bloat is well-defined and can be identified, there are currently no consensual explanations on why it occurs. Authors have presented different explanations regarding bloat. Three popular theories can be found in the literature to explain it [36]:

– The introns theory states that bloat acts as a protective mechanism in order to avoid the destructive effects of operator's 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 sub-programs 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 to many algorithms using variable length representation.

– The removal bias theory states that removing longer sub-programs 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 largetrees) 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.

Authors have distinguished bloat into two main types: structural bloat and functional bloat [32]:

A. Structural Bloat

The structural bloat is defined 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.

B. Functional Bloat

Another form of bloat is the functional bloat, which takes place when program length keeps on growing even though an optimal solution (of known complexity) does lie in the search space. Most of the works cited earlier are in fact concerned with functional bloat which is the most surprising, and the most disappointing kind of bloat. There are various levels of functional bloat: cases where the length of programs found by GP runs to infinity as the number of test cases run to infinity whereas a bounded-length solution exists, and also cases where large programs are found with high probability by GP where as a small program is optimal.

III. Previous Work Done

Liu, Cai, Ying, and Le in [29] stated that GP using a size or depth limit (LGP) is a common approach to battle bloat, but LGP is not ideal in size control and searching efficiency. In their paper, besides extended the concept of bloated individual in LGP, and the concept of Candidate Crossover Points Set is presented. A new variants of LGP, named RLGP, which adds some restrictions in genetic operations (crossover, swap, and mutation), is proposed. RLGP introduces Candidate Crossover Points Set (CCPS) into crossover operations. Finally, in even 3, 4, and 5-parity problem, strongly positive results are reported regarding both size control and searching efficiency.

Thomas Helmuth, Lee Spector and Brian Martin [30], introduced a new node selection method that selects nodes based on a tournament, from which the largest participating sub-tree is selected. Size-based tournaments differentiate between internal nodes of different sizes, whereas Koza 90/10 treats all internal nodes equally. This method of size-based tournaments improves performance on three standard test problems with no increases in code bloat as compared to unbiased and Koza 90/10 selection methods.

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 appear to balance this tradeoff without compromising overall fitness.

Langdon and Poli [2] have described a 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 [4], 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 [15] proposed a nonparametric method, Double Tournament, 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 [31] presented 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.

Soule and Foster [32], 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 described 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.

A review and comparison of the most common methods is given in Luke and Panait [18], while discussion on the causes of bloat maybe found in Soule and Heckendorn and the Field Guide to Genetic Programming [25].These previous methods generally take into account individual size to control bloat. However, a number of researchers have also considered methods that do not explicitly consider the size of individuals and therefore bloat reduction results as a side effect.

Over the years a range of methods have been introduced to manage bloat [1]-[33] There are two major approaches to dealing with GP tree bloat. First, by improving breeding, selection, and tree-generation, GP can be made to search more efficiently to find better individuals before bloat sets in. Second, various techniques can help GP put off bloat as long as possible, lengthening the search interval [4]. Following things are considered under these two approaches while handling bloat:

- treating fitness and size as a multiobjective optimization;
- using disassortative mating based on two species (one selected on fitness, the other on fitness and size);
- explicitly reducing the fitness of above average-sized individuals (referred to as the Tarpeian method);
- eliminating programs where the parent and child fitness are similar using a modified tournament selection operator that uses either fitness, depth or an ordered combination of both for selection;
- placing a form of resource constraint on the population so that larger individuals are discouraged;
- using a waiting room for individual entry into a population, with time proportional to size;
- biasing selection for removal from a population based on size;
- explicitly simplifying individuals after each generation;
- dynamically extending an initially low maximum tree depth only when a child is produced that is fitter than the best individual and larger then this size limit;
- viewing size as a resource constraint, that can only be extended by fitness improvements; and
- applying specific genetic operators to reduce the size of large individuals or maintain the size of children to parents.

IV. Methods to Avoid Bloat

Different methods have been proposed in the literature to avoid bloat. Some of them are described as follows:

A. Using a fix size or depth limit (LGP)

The most common way to avoid bloat is to limit the size (number of nodes) and depth (height of the tree) of the individual [2]. The limitations on size and depth can be considered during initialization of population by using suitable algorithm for tree generation. 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. This method efficiently restricted individual from bloating.

B. Parsimony Pressure

Parsimony Pressure is not only another popular bloat control technique in GP, but also has been used in a wide variety of arbitrary-length representations tended to get out of control. It is the second most common method for controlling bloat. This is done by adding a tree size penalty as an additional criterion in the individual's fitness assessment. Individual having larger tree size is allotted lower fitness. Many researchers till date have used parsimony pressure as a method to avoid bloat while using GP for problem solving. Such usage is divided into two broad categories: parametric parsimony pressure, where size is directly defined as a numerical factor in fitness, and pareto parsimony pressure, where size is considered as a separate objective in a pareto-optimization procedure.

C. Local Elitism Method

While performing crossover or mutation operation, new individual or child are generated. The child can only replace the parent in new generation if its fitness is better than or equal to the parent fitness otherwise it will be retained with some probability [34]. In this way better individuals can be taken to the new generation.

D. Using modified genetic operators

The genetic operators, crossover and mutation can be modified to avoid the problem of bloat. For example in FEDS crossover (Fitness, Elitism, Depth limit & Size), the concept of fitness, elitism, depth limit and tree size for generating the next generation individuals through crossover operation is proposed to handle the problem of code bloating in GP [35]. In [37], authors

have used point mutation operation to avoid increase in tree size during mutation.

V. CONCLUSION

In this paper, we have done theoretical study of an important issue in Genetic Programming known as code bloating. Bloating hampers the performance of program structures designed using GP. A lot of research has been done to find the actual cause of bloat and methods to avoid and reduce the problem of bloat. For this various strategies including modification in genetic operators (crossover and mutation) have been presented.

Our contribution in the paper can be summarized as follows:

1) We have presented the theoretical concept of code bloating in GP.
2) We have identified different types of bloat and their root causes.
3) We have discussed about different methods to avoid and control bloat in GP.

REFERENCES

[1] Peter A. Whigham, and Grant Dick, "Implicitly Controlling Bloat in Genetic Programming," IEEE Transaction on Evolutionary Computation, Vol. 14, No. 2, APRIL 2010, pp. 173-190.
[2] W. B. Langdon and R. Poli, "Fitness causes bloat: Mutation" in Proc. Theory Application Evolutionary Comput. (ET '97), London, U.K.: University College London, 1997, pp. 59–77.
[3] W. Banzhaf and W. Langdon, "Some considerations on the reason for bloat," Genetic, Programming Evolvable Mach.,vol. 3, no. 1, pp. 81–91, 2002.
[4] S. Luke, "Modification point depth and genome growth in genetic programming," Evol. Comput., vol. 11, no. 1, 2003, pp. 67–106.
[5] J. Koza, "Genetic Programming: On the Programming of Computers by Natural Selection", Cambridge, MA: MIT Press, 1992.
[6] B.-T. Zhang and H. Muhlenbein, "Evolving optimal neural networks using genetic algorithms with Occam's razor," Complex Syst., vol. 7, pp. 199–220, 1993.
[7] E. D. Jong, R. Watson, and J. Pollack, "Reducing bloat and promoting diversity using multiobjective methods," in Proc. Genet. Evol. Comput. Conf. (GECCO'01), San Francisco, CA: Morgan Kaufmann, 2001, pp.11–18.
[8] M. Terrio and M. I. Heywood, "Directing crossover for reduction of bloat in GP," in Proc. IEEE Can. Conf. Elect Comput. Eng. (IEEECCECE 2003), Piscataway, NJ: IEEE Press, May 12–15, 2002, pp.1111–1115.
[9] R. Poli, "A simple but theoretically- motivated method to control bloat in genetic programming," in Proc. Genet. Programming (EuroGP '03) vol. 2610. Essex: Springer-Verlag, Apr. 14–16, 2003, pp. 204–217.
[10] H. Stringer and A. Wu, "Bloat is unnatural: An analysis of changes invariable chromosome length absent selection pressure," Univ. Central Florida, Tech. Rep. CS-TR-04-01, 2004.
[11] 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, June26–30, 2004, pp. 198–209.
[12] 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, 2005, pp. 390–395.
[13] W. B. Langdon, "Quadratic bloat in genetic programming," in Proc. Genet. Evol. Comput. Conf. (GECCO '00), Las Vegas, NV: Morganm Kaufmann, Jul. 10–12, 2000, pp. 451–458.
[14] B.-T. Zhang and H. Muhlenbein, "Balancing accuracy and parsimony in genetic programming," Evol. Comput., vol. 3, no. 1, 1995, pp. 17–38.
[15] 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, 2001, pp.536–543.
[16] C. Ryan, "Pygmies and civil servants," Advances in Genetic Programming" MIT Press, 1994, ch. 11, pp. 243-263. Available:http://cognet.mit.edu/library/books/viewfiisbn=0262111888.
[17] M. J. Streeter, "The root causes of code growth in genetic programming," in Proc. Genet. Programming (EuroGP '03), vol. 2610. Essex: Springer-Verlag, Apr. 14–16, 2003, pp. 443–454.
[18] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," Evol. Comput., vol. 14, no. 3, 2006, pp. 309–344.
[19] N. Wagner and Z. Michalewicz, "Genetic programming with efficient population control for financial time series prediction," in Proc. Genet. Evol. Comput. Conf. Late Breaking Papers, 2001, pp. 458–462.
[20] P. Wong and M. Zhang, "Algebraic simplification of GP programs during evolution," in Proc. 8th Annu. Conf. Genet. Evol. Comput. (GECCO '06), Seattle, WA: ACM, pp. 927–934.
[21] S. Silva and J. Almeida, "Dynamic maximum tree depth," in Proc.Genet. Evol. Comput. (GECCO '03), vol. 2724. Chicago, IL: Springer-Verlag, 2003, pp. 1776–1787.
[22] S. Silva and E. Costa, "Resource-limited genetic programming: The dynamic approach," in Proc. 2005 Conf. Genet. Evol. Comput. (GECCO '05), New York: ACM, pp. 1673–1680.
[23] C. J. Kennedy and C. Giraud-Carrier, "A depth controlling strategy for strongly typed evolutionary programming," in Proc. Genet. Evol. Comput. Conf. vol. 1. Orlando, FL: Morgan Kaufmann, nos. 13–17 1999, pp. 879–885.
[24] S. Luke and L. Panait, "Fighting bloat with nonparametric parsimony pressure," in Proc. 7th Int. Conf. Parallel Problem Solving Nature (PPSNVII), vol. 2439, Granada, Spain: Springer, Sep. 7–11, 2002, pp. 411–421.
[25] R. Poli, W. B. Langdon, and N. F. McPhee, "A Field Guild to Genetic Programming" Lulu.com, 2008, pp. 102–103.
[26] F. Fernandez, L. Vanneschi, and M.Tomassini, "The effect of plagues in genetic programming: A study of variable-size populations," in Proc. Genet. Programming (EuroGP '03), vol. 2610. Essex: Springer-Verlag, 2003, pp. 317–326.
[27] D. Rochat, M. Tomassini, and L. Vanneschi, "Dynamic size populations in distributed genetic programming," in Proc. 8th Eur. Conf. Genet. Programming, vol. 3447. Berlin: Springer, 2005, pp. 50–61.
[28] F. Fernandez, G. G. Gil, J. A. Gomez, and J. L. Guisado, "Control of bloat in genetic programming by means of the island model," in Proc. Parallel Problem Solving Nature (PPSN VIII), vol. 3242. Birmingham, U.K.: Springer-Verlag, 18–22 Sep. 2004, pp. 263–271.
[29] Liangxu Liu, Haibin Cai, Mingyou Ying, and Jiajin Le, "RLGP: An Efficient Method to Avoid Code Bloating on Genetic Programming", Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, August 5 - 8, 2007, Harbin, China, pp. 2945-2950.
[30] Helmuth, T., L. Spector, and B. Martin, "Size-Based Tournaments for Node Selection", in GECCO'11, Workshops, Genetic and Evolutionary Computation Conference, 2011, ACM Press. pp. 799-802.
[31] Sara Silva and Ernesto Costa, "Dynamic Limits for Bloat Control Variations on Size and Depth", K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, 2004, pp. 666–677.
[32] Soule T., Foster J. A., and Dickinson J., "Code growth in Genetic Programming,", Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press, 1996, pp. 215-223.
[33] Sylvain Gelly, Olivier Teytaud, Nicolas Bredeche, Marc Schoenauer, "A Statistical Learning Theory Approach of Bloat", GECCO'05, June 25–29, 2005, Washington, DC, USA, ACM, pp. 1783-1784.
[34] Anuradha Purohit, Arpit Bhardwaj, Aruna Tiwari, Narendra S. Chaudhari, "Handling The Problem of Code Bloating To Enhance The Performance of Classifier Designed Using Genetic Programming", 5th Indian International Conference on Artificial Intelligence (IICAI-11), 14-16 December 2011, Tumkur, pp. 333-342.
[35] Anuradha Purohit, Arpit Bhardwaj, Aruna Tiwari, Narendra S. Chaudhari, "Removing Code Bloating in Crossover Operation in Genetic Programming", International Conference on Recent Trends in Information Technology (ICRTIT-11), June 03-05 2011, pp. 77.

[36] Nur Merve Amil, Nicolas Bredeche, and Christian Gagne, "A Statistical Learning Perspective of Genetic Programming", October 26, 2007, pp.1-51.
[37] Durga Prasad Muni, Nikhil R. Pal, and Jyotirmoy Das, "A Novel Approach to Design Classifiers Using Genetic Programming", IEEE Transactions on Evolutionary Computation, Vol. 8, No. 2, April 2004, pp. 183-196.

AUTHORS

First Author – Anuradha Purohit, M.E. Computer Engg., Assistant Professor, Computer Technology and Applications Deptt., S.G.S.I.T.S. Indore, M.P., India, anuradhapurohit@rediffmail.com.
Second Author – Narendra S. Chaudhari, Ph.D. Computer Engineering, Professor and Head, Computer Engineering Deptt., IIT Indore, M.P., India, nsc183@gmail.com.
Third Author – Aruna Tiwari, Ph.D. Computer Engineering, Assistant Professor, Computer Engineering Deptt., IIT Indore, M.P., India, aruna_tiwari@rediffmail.com.

Correspondence Author – Anuradha Purohit, anuradhapurohit@rediffmail.com, apurohit@sgsits.ac.in, 9826065208.