A Statistical Learning Theory Approach of Bloat

Sylvain Gelly, Olivier Teytaud, Nicolas Bredeche, Marc Schoenauer
Equipe TAO - INRIA Futurs
LRI, Bat. 490,
University Paris-Sud
91405 Orsay Cedex. France
{*firstname.name*}*@lri.fr*

ABSTRACT

Code bloat, the excessive increase of code size, is an important issue in Genetic Programming (GP). This paper proposes a theoretical analysis of code bloat in the framework of symbolic regression in GP, from the viewpoint of Statistical Learning Theory, a well grounded mathematical toolbox for Machine Learning. Two kinds of bloat must be distinguished in that context, depending whether the target function lies in the search space or not. Then, important mathematical results are proved using classical results from Statistical Learning. Namely, the Vapnik-Chervonenkis dimension of programs is computed, and further results from Statistical Learning allow to prove that a parsimonious fitness ensures Universal Consistency (the solution minimizing the empirical error does converge to the best possible error when the number of examples goes to infinity). However, it is proved that the standard method consisting in choosing a maximal program size depending on the number of examples might still result in programs of infinitely increasing size with their accuracy; a more complicated modification of the fitness is proposed that theoretically avoids unnecessary bloat while nevertheless preserving the Universal Consistency.

(full paper available at `http://www.lri.fr/~teytaud/longBloat.pdf`, [3])

Categories & Subject Descriptors

G.1.6 Global Optimization

General Terms

Algorithms, Reliability, Theory.

Keywords

Code Bloat, Code Growth, Genetic Programming, Statistical Learning Theory.

1. CODE BLOAT IN GP

There exists several theories that intend to explain code bloat : the *introns* theory ([12, 11, 2]), the *fitness causes bloat* theory ([8],[6]), the *removal bias* theory ([15]). While it is now considered that each of these theories somewhat captures part of the problem [1], 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 (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 [7], or different Fair Mutation [9]), on some parsimony-based penalization of the fitness [16] or on abrupt limitation of the program size such as the one originally used by Koza [5]. Some other more particular solutions have been proposed but are not widely used yet [13, 14, 10].

2. OVERVIEW OF RESULTS

The detailed results are available in [4]. We shall investigate the Universal Consistency of Genetic Programming algorithm, and study in detail structural and functional bloat that might take place when searching program spaces using GP.
A formal and detailed definition of the program space that will be assumed for GP is given in Lemma 1 ([4]), and two types of results will then be derived:

- *Universal Consistency* results, i.e. does the probability of misclassification of the solution given by GP converges to the optimal probability of misclassification when the number of examples goes to infinity?
- Bloat-related results, first regarding structural bloat, that will be proved to be incompatible with accuracy, and second with respect to functional bloat, for which the consequences of introducing various types of fitness penalization and/or bound on the complexity of the programs on the behavior of the complexity of the solution will be thoroughly studied.

Let us now state precisely, yet informally, our main results:

- First, we will precisely define the set of programs under examination, and prove that such a search space fulfills the conditions of the standard theorems of Statistical Learning Theory.
- Applying those theorems will immediately lead to a first Universal Consistency result for GP, provided that some penalization for complexity is added to the fitness (Theorem 3)
- The first bloat-related result, Proposition 4, unsurprisingly proves that if the optimal function does not belong to the search space, then converging to the optimal error implies that the complexity of the empirical optimal solution goes to infinity (unavoidable structural bloat).

GECCO'05, June 25–29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

- Theorem 5 is also a negative result about bloat, as it proves that even if the optimal function belongs to the search space, minimizing the LSE alone might lead to (structural) bloat (i.e. the complexity of the empirical solutions goes to infinity with the sample size).

- But the last two theorems (5' and 6) are the best positive results one could expect considering the previous findings: it is possible to carefully adjust the parsimony pressure so as to obtain both Universal Consistency and bounds on the complexity of the empirical solution (i.e. no bloat).

Note that, though all proofs in [4] will be stated and proved in the context of classification (i.e. find a function from $\mathbb{R}^d$ into $\{0,1\}$), their generalization to regression (i.e. find a function from $\mathbb{R}^d$ into $\mathbb{R}$) is straightforward.

3. CONCLUSION

In this paper, we have proposed a theoretical study of an important issue in Genetic Programming known as code bloat. We have shown that GP trees used in symbolic regression (involving the four arithmetic operations, the exponential function, and ephemeral constants, as well as test and jump instructions) could be studied using some classical tools from Statistical Learning Theory. This has lead to two kinds of original outcomes: some results about Universal Consistency of GP, i.e. some guarantee that if GP converges to some (empirical) function, this function will be close to the optimal one if sufficiently many examples are used; and results about the bloat, both the unavoidable structural bloat in case the target ideal function is not included in the search space, and the functional bloat, for which we proved that it can – theoretically – be avoided by simultaneously bounding the length of the programs with some *ad hoc* bound and using some parsimony pressure in the fitness function. Some negative results have been obtained, too, such as the fact though structural bloat was know to be unavoidable, functional bloat might indeed happen even when the target function does lie in the search space, but no parsimony pressure is used.

Interestingly, all those results (both positive and negative) about bloat are also valid in different contexts, such as for instance that of Neural Networks (the number of neurons replaces the complexity of GP programs). Moreover, results presented here are not limited to the scope of regression problems, but may be applied to variable length representation algorithms in different contexts such as control or identification tasks.

Finally, going back to the debate about the causes of bloat in practice, it is clear that our results can only partly explain the actual cause of bloat in a real GP run – and tends to give arguments to the "fitness causes bloat" explanation [8]. It might be possible to study the impact of size-preserving mechanisms (e.g. specific variation operators, like size-fair crossover [7] or fair mutations [9]) as somehow contributing to the regularization term in our final result ensuring both Universal Consistency and no-bloat.

Acknowledgements

This work was supported in part by the PASCAL Network of Excellence.

4. REFERENCES

[1] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[2] T. Blickle and L. Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms Workshop at KI-94*, pages 33–38. Max-Planck-Institut für Informatik, 1994.

[3] S. Gelly, O. Teytaud, N. Bredeche, and M. Schoenauer. Apprentissage statistique et programmation gntique: la croissance du code est-elle invitable? In *Actes de la conference CAp*. PUG, 2005.

[4] S. Gelly, O. Teytaud, N. Bredeche, and M. Schoenauer. A statistical learning theory approach of bloat. *www.lri.fr/ teytaud/longBloat.pdf*, 2005.

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

[6] W. B. Langdon. The evolution of size in variable length representations. In *ICEC'98*, pages 633–638. IEEE Press, 1998.

[7] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming And Evolvable Machines*, 1(1/2):95–119, 2000.

[8] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In J. Koza, editor, *Late Breaking Papers at GP'97*, pages 132–140. Stanford Bookstore, 1997.

[9] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming III*, pages 163–190. MIT Press, 1999.

[10] S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. L. et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann Publishers, 2002.

[11] N. F. McPhee and J. D. Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

[12] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[13] A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. C. et al., editor, *Artificial Evolution VI*. Springer Verlag, 2001.

[14] S. Silva and J. Almeida. Dynamic maximum tree depth : A simple technique for avoiding bloat in tree-based gp. In E. C.-P. et al., editor, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1776–1787. Springer-Verlag, 2003.

[15] T. Soule. Exons and code growth in genetic programming. In J. A. F. et al., editor, *EuroGP 2002*, volume 2278 of *LNCS*, pages 142–151. Springer-Verlag, 2002.

[16] T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, 1998.