Mutational Robustness and Structural Complexity in Grammatical Evolution

Michael O'Neill
Natural Computing Research & Applications Group
School of Business
University College Dublin
Dublin, Ireland
m.oneill@ucd.ie

Anthony Brabazon
Natural Computing Research & Applications Group
School of Business
University College Dublin
Dublin, Ireland
anthony.brabazon@ucd.ie

***Abstract*—A recent study in Artificial Life found that the need for mutational robustness can give rise to simpler structures in an evolving population. This begs the question, do we observe a similar phenomenon in Genetic Programming? Genetic Programming requires the search of structural space of solutions, usually requiring code growth to find fitter solutions. Typically Genetic Programming algorithms then suffer from code bloat, which is code growth in the absence of fitness gains. In this study we ask a simple question. Would the necessity for mutational robustness under selection pressure drive the evolution of less complex solution structures in Genetic Programming, with the potential to counteract code bloat?**

***Index Terms*—Mutational Robustness, Complexity, Program Synthesis, Genetic Programming, Grammatical Evolution**

I. Introduction

With some exceptions, biological complexity is considered to have increased over evolutionary time [1]. The causes of increased complexity are theorised to be either a function of selection pressure or due to the operation of the variation operators, with degeneracy (a form of biological redudancy) being a key feature [2]. In a recent study in the Artifical Life literature [3], in the presence of selection, the generation of increasing complexity is observed in an environment where functional simplicity has higher fitness. However, with increased rates of mutation the complexity of the agents is observed to reduce. That is, the necessity for mutational robustness can give rise to simpler structures.

GP individuals are open in their structural complexity to the limit of a maximum tree depth and/or tree node count, in that their size and architecture can grow and shrink and varies amongst individuals in a population. Most researchers in GP have observed the phenomena of code growth (often a necessity to find a solution), and the problem of code bloat (the increase in solution size without a corresponding fitness gain), with many studies developing theory to explain the origin of code bloat and strategies to prevent excessive bloat, which can hamper effective evolutionary search.

Mutation is often an under-exploited variation operator in GP, which has a strong traditional dependence upon subtree crossover. Inspired by [3] we ask the question, can structural complexity in the form of code bloat in GP, be counteracted by a simple strategy of increased rates of mutation? To state this another way, does the necessity for mutational robustness give rise to simpler structures in GP?

The following Section II provides some background to the key concepts examined in this study with a particular emphasis on the context of GP. The experimental setup and a presentation of the results are provided in Sections III and IV, before we set out our conclusions and point to opportunities for future research in Section V.

This research is supported by the Science Foundation Ireland under grants 13/IA/1850 and 13/RC/2094.

II. Background

It is considered that robustness (i.e., the ability for function to persist after some modification) promotes evolvability in nature [4], and while genetic robustness can present a paradox in terms of having low evolvability (i.e., low ability to generate heritable phenotypic variation, and sometimes referred to as adaptive innovations) due to the requirement for neutral or nearly-neutral change events, robustness of phenotypic structures can have a positive impact on evolvability [5].

In Evolutionary Computation mutational robustness has included a large focus on neutral evolution and how neutrality can protect against harmful mutation and improve diversity (e,g,. [6]–[10]). In GP ideas around evolvability have been discussed since the early days of the field [11], and there have been studies on robustness and evolvability under recombination in Linear GP [12]–[14]. In the related fields of Search-based Software Engineering and Genetic Improvement there have been studies of software mutational robustness [15], [16].

Recently in an Artifical Life study [3] researchers observed that in evolving populations mutational robustness can give rise to simpler phenotypic structures. This raises an alternative perspective on the role of mutation in evolutionary systems, where instead of being for the introduction of novelty, with the corresponding negative side of this with too high a rate of mutation, or mutation at the incorrect locus which inflicts phenotypic impairment, mutation can have a positive impact through the promotion of simpler structures. This perspective has not been investigated in the field of GP where search of structural space is a necessity. One feature of searching

structural space is the requirement for code growth. That is individuals in a population are variable in size and tend to grow in size over evolutionary time in the search for a candidate solution. A negative side to code growth, is code bloat, where individuals grow in size without the benefit of fitness gains.

Bloat at generation g has been defined [17] as

$$bloat(g) = \frac{(\overline{\delta_g} - \overline{\delta_0})/\overline{\delta_0}}{(\overline{f_g} - \overline{f_0})/\overline{f_0}} \quad (1)$$

where $\overline{\delta_g}$ is the average program length at generation g, and $\overline{f_g}$ is the average population fitness at generation g. Therefore $\overline{\delta_0}$ and $\overline{f_0}$ are the average program length and average population fitness at generation zero, so this definition of bloat compares the increase in program length relative to generation zero to the increase in average fitness relative to generation zero.

Bloat is a consistent problem for GP practitioners impacting on the efficiency of search, and consequently has been a significant area of research in the GP community including development of theoretical foundations and empirical investigations into the potential origins or causes of bloat and strategies to mitigate against bloat. In more recent times the development of the Crossover Bias Theory and the resulting bloat mitigating operator equalisation by Dignum [18], [19] has produced a relatively simple and practical approach to managing bloat within GP populations. The interested reader is referred to a recent survey on bloat in GP [20].

In the current context where we wish to examine if there is a relationship between mutational robustness and the complexity of evolving structures, it is interesting to consider if there might be a corresponding impact on bloat through its reduction. It is interesting to note that a study of Linear GP search operators, which promote the inclusion of introns, and by definition increase bloat, improves performance on the classification problems examined [21]. Indeed one can find many contradictory studies around redudancy and neutrality in EC [6].

TABLE I: Evolution Parameter Settings for PonyGE2

Parameter	**Value**
Population size	500
Generations	500
Initialisation	Position Independent Grow Tree Depth=10
Max Derivation Tree Depth	17
Max Genome Length	500
Selection	Tournament (size=5)
Replacement	Generational with Elitism (5 elites)
Mutation	codon int flip per ind (1 or 2 events per individual)
No Mutation Invalids	True
Crossover	variable one-point (probability=0.9)
No Crossover Invalids	True
Within Used	True
Invalid Selection	False
Cache	True Lookup Fitness=True Mutate Duplicates=True
Fitness Function	mse
Replications	30 independent runs

III. Experimental Setup

In this study we set out to address the question, does the necessity for mutational robustness give rise to simpler structures in GP? To test this we employ two experimental setups. The first which permits a single mutation event on the genome (labelled `mutn1`), and the second (`mutn2`) doubles this allowing two mutation events. The mutation rate in both setups is deliberately chosen to be conservatively low and well below the error threshold where selection pressure would be unable to preserve information in the population. The null hypothesis states that we will observe no difference in structural complexity of the phenotype when comparing the two experimental setups.

In traditional GP the genome and phenome are equivalent (i.e., they are both the tree), and the complexity of the tree structure has been previously defined as the number of nodes in the tree [22]. We employ a grammar-guided form of GP, Grammatical Evolution [23]–[25] in the form of PonyGE2 [26] as it embodies an evolutionary algorithm with a genotype-phenotype mapping appropriate to study an exploration of the evolution of complexity of both genotypic and phenotypic structures. To this end we measure the complexity of genotypes in terms of the number of integer codons in a genome, and phenotypic complexity as the number of nodes in the derivation tree and the depth of the derivation tree of each individual.

The elephant in the room [27] of open issues in the field of GP [28] is it's lack of application to automatic programming. To this end we adopt 12 problems from the Helmuth and Spector Program Synthesis Benchmark problem set [29]. From these benchmark problems are drawn two easy (Number IO, Smallest), two medium (Grade, Last Index of Zero) and eight hard (Sum of Squares, Compare String Lengths, Scrabble Score, Checksum, Double Letters, Mirror Image, Pig Latin and Super Anagrams) problems in terms of levels of difficulty [30]. Greater potential for code growth and bloat is expected as the problem difficulty level increases. For each problem there is a set of training data, with fitness being calculated as the mean square error across the training set. Therefore, our objective is to minimise error, and lower fitness values are better.

Evolutionary parameters are detailed in Table I. With the Cache settings employed (`Cache=True`, `Lookup Fitness=True` and `Mutate Duplicates=True`) duplicate individuals are not permitted in the population. If a duplicate is detected it is mutated until it is unique. Linear genome mutation and crossover operators are adopted which prevent the generation of invalid individuals, and their application is restricted to occur in the expressed region of the genome (`Within Used=True`). A derivation-tree based initialisation (`Position Independent Grow`) is used to control variety of tree structures generated up to the derivation-tree initialisation depth maximum of 10. The grammars and code are available off-the-shelf from https://github.com/PonyGE/PonyGE2.

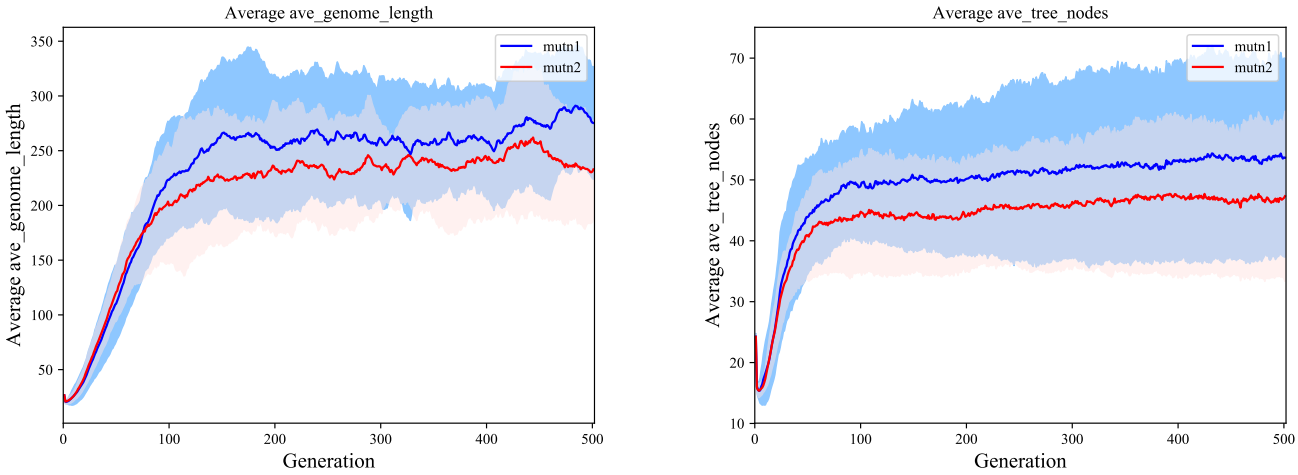


Fig. 1: Number IO (L to R: Genome Length, Tree Nodes)

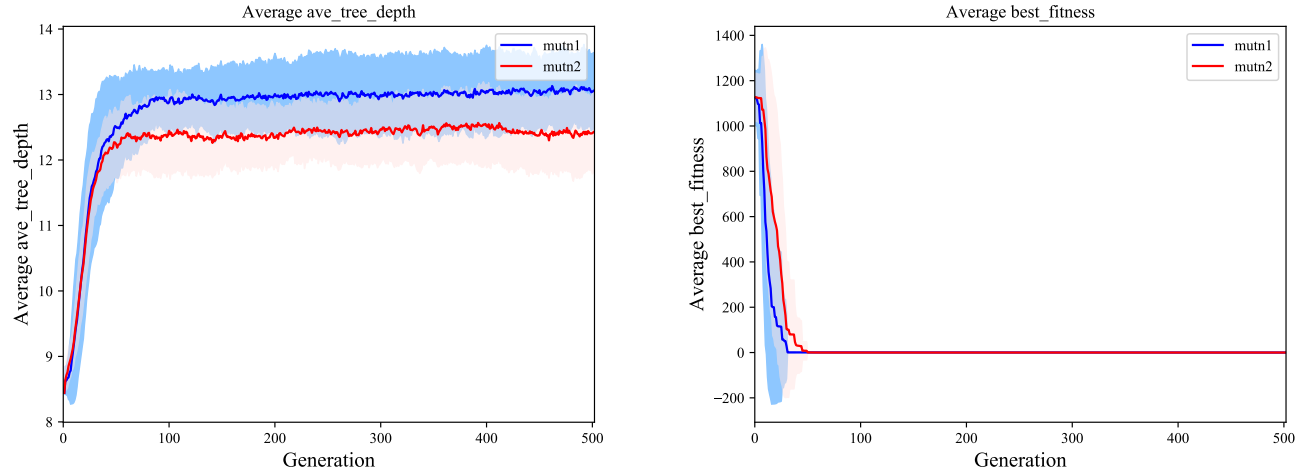


Fig. 2: Number IO (L to R: Tree Depth, Best Fitness)

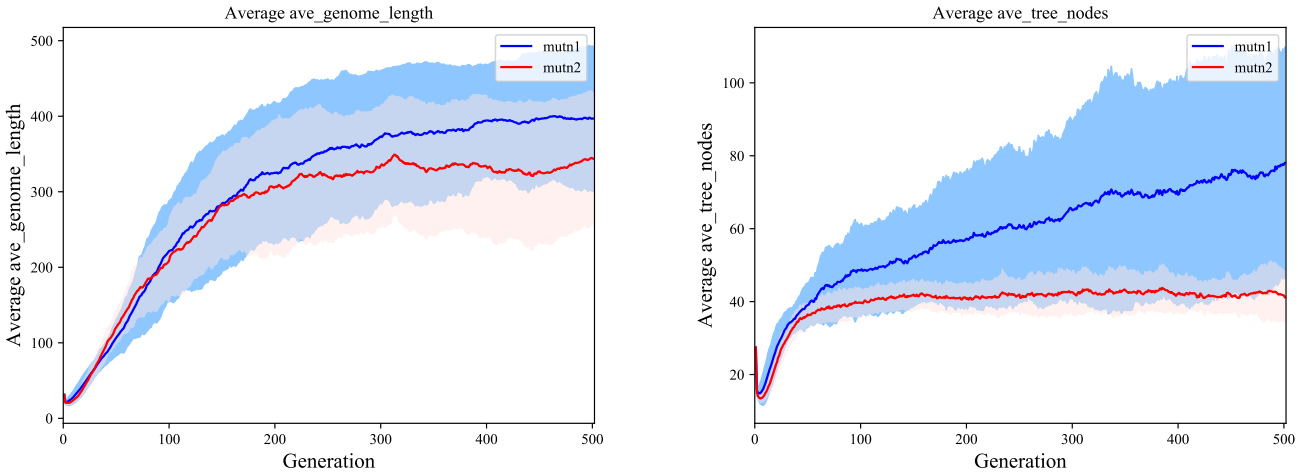


Fig. 3: Grade (L to R: Genome Length, Tree Nodes)

IV. Experimental Results

Figures 1, 2, 3, 4, 5, and 6 show genotypic complexity (number of codons in a genome), phenotypic complexity (nodes in a derivation tree and tree depth) and the evolution of best fitness over time on a sample of the easy, medium and hard difficulty program synthesis benchmarks with error bars. The lower mutation setup is labelled `mutn1`, and the higher mutation rate setup is labelled `mutn2` in each case.

It is observed that in the presence of the higher rate of mutation phenotypic complexity is lower on all of the 8 harder problem instances (the standard deviations of the low and high mutation rate setups are not overlapping on the average derivation-tree node count or on the average derivation tree depth). However, on the easy and medium difficulty instances this relationship is not observed with the exception of the average tree depth at the end of runs on the two medium difficulty problems. While there is some reduction in genome complexity, the average genome length standard deviations are overlapping on the two setups across all problem instances. Similarly, the difference in fitness attained at the end of the run on all problem instances is not significant.

In a control setup where there is no mutation or crossover, evolution is purely driven by selection and replacement, we observe mean derivation tree node counts remaining relatively static around 15, 17 and 18 for the easy, medium and hard sample problems, which corresponds to values observed in the earlier generations in the right-hand side of Figures 1, 3 and 5. In the same control setup tree-depth remains static around 8, 9 and 10 for the easy, medium and hard sample problem instances respectively. For comparison see the left-hand side of Figures 2, 4 and 6 where these correspond closely with the mean initialised depths. In the control setup we also observe mean genome lengths to remain static around 20 codons across the sample problems corresponding again to earlier generations as illustrated in the left-hand of Figures 1, 3 and 5. These observations of node count and tree depths on the control setup confirms that code growth and bloat is not occurring purely as a function of run-time/generations.

For the both the high (`mutn2`) and low (`mutn1`) mutation rate setups it is also be observed that the genome complexity is much higher than the resulting phenotype complexity measures, indicating that there are large portions of the genome which are not expressed in the generation of the phenotypes. This is confirmed examining the ratio of the expressed genome to the genome length in Figures 7, 8 and 9 where towards the end of the run less than one fifth of the genome is used across all problem instances.

Note that the maximum derivation-tree depth permitted in the population is 17, and the maximum genome length is 500 codons. While on average we do not observe a depth of 17 being reached on any of the problem instances, this is in effect a primitive mechanism for code growth prevention, with the possibility that some crossover events have been prevented from occurring due to their violating this depth limit. Similarly the maximum genome length is a control over genotypic complexity and this limit is approached on a number of the problem instances. It would be interesting to observe what occurs if these primitive structural limits are relaxed to a larger number. Will we observe an increased difference between the phenotypic structures under the different rates of mutation?

Do we observe an impact of the rate of mutation on bloat? We undertake a post-hoc analysis of the average derivation-tree node count and average population fitness and calculate bloat using a modified equation to the one adopted in [17]. We modify Equation 1 as this definition assumes lengths (derivation-tree node counts) increase over time, and it also assumes that the lowest length is at generation zero. As these do not always hold true in GE (we tend to observe an initial reduction in lengths in early generations before growth occurs), instead of using average tree node counts at generation zero as a reference point, we find the generation at which the lowest average tree node count occurs and use this $\overline{\delta_{min}}$ in place of $\overline{\delta_0}$. Our new bloat equation becomes:

$$bloat(g) = \frac{(\overline{\delta_g} - \overline{\delta_{min}})/\overline{\delta_{min}}}{(\overline{f_g} - \overline{f_0})/\overline{f_0}} \quad (2)$$

Figures 10, 11 and 12 illustrate on the sample problems what we observe on the majority of problems (8 out of the 12 examined) that the higher rate of mutation setup (`mutn2`) results in reduced bloat.

V. Conclusions & Future Work

In addressing the question, does the necessity for mutational robustness give rise to simpler structures in GP?, in our experiments we observe a reduction in structural complexity of phenotypes on harder problem instances under higher rates of mutation without a significant loss of fitness. We also observe a reduction in code bloat on 8 out of the 12 problems examined. These results suggest an important role for mutation in GP algorithms as a potential mechanism to facilitate control of code bloat, but perhaps more importantly to facilitate the generation of simpler structures. Future work will include extending the problem set, and testing different flavours of GP such as PushGP, CGP, traditional Koza-style GP etc to see if we observe similar behaviour under increased rates of mutation.

References

[1] C. Adami, "What is complexity?" *BioEssays*, vol. 24, no. 12, pp. 1085–1094, 2002.

[2] J. M. Whitacre, "Degeneracy: a link between evolvability, robustness and complexity in biological systems," *Theoretical Biology and Medical Modelling*, vol. 7, no. 1, p. 6, Feb 2010. [Online]. Available: https://doi.org/10.1186/1742-4682-7-6

[3] V. Liard, D. Parsons, J. Rouzaud-Cornabas, and G. Beslon, "The complexity ratchet: Stronger than selection, weaker than robustness," *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, pp. 250–257, 2018. [Online]. Available: https://www.mitpressjournals.org/doi/abs/10.1162/isal_a_00051

[4] O. C. Lenski RE, Barrick JE, "Balancing robustness and evolvability," *PLoS Biol*, vol. 4, p. 428, 2006.

[5] A. Wagner, "Robustness and evolvability: a paradox resolved," *Proceedings of the Royal Society B: Biological Sciences*, vol. 275, no. 1630, pp. 91–100, 2008.

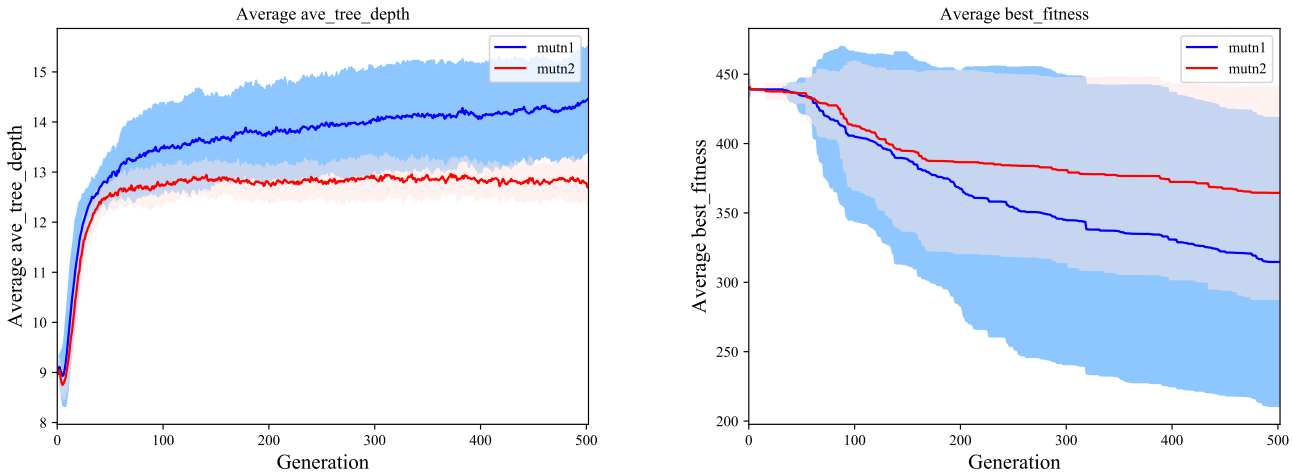


Fig. 4: Grade (L to R: Tree Depth, Best Fitness)

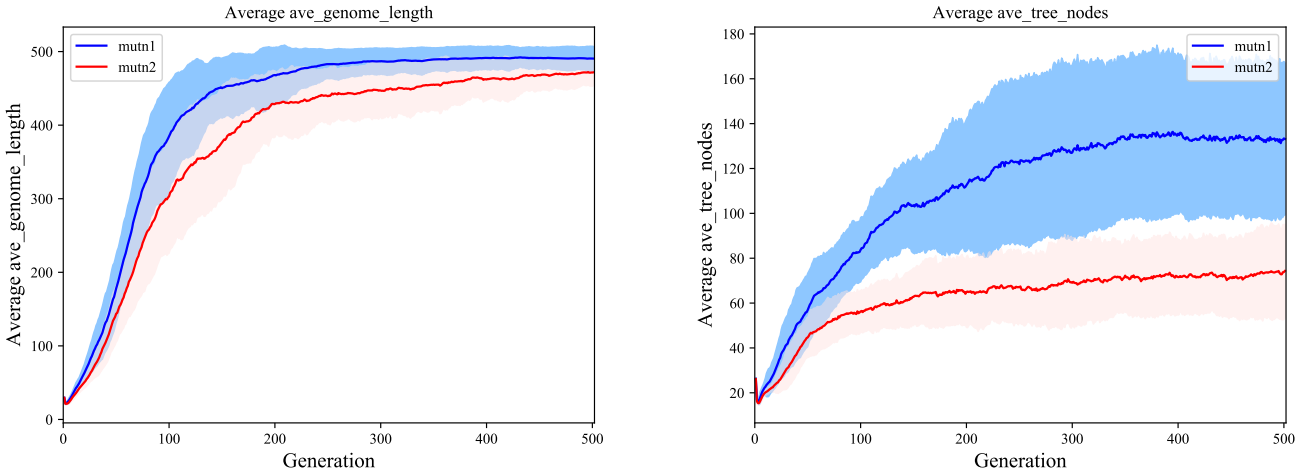


Fig. 5: Sum of Squares (L to R: Genome Length, Tree Nodes)

[6] E. Galvan-Lopez, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon, "Neutrality in evolutionary algorithms... what do we know?" *Evolving Systems*, vol. 2, no. 3, pp. 145–163, Sep. 2011.

[7] T. Hu, J. Payne, J. Moore, and W. Banzhaf, "Robustness, evolvability, and accessibility in linear genetic programming," in *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, ser. LNCS, S. Silva, J. A. Foster, M. Nicolau, M. Giacobini, and P. Machado, Eds., vol. 6621. Turin, Italy: Springer Verlag, 27-29 Apr. 2011, pp. 13–24.

[8] W. Banzhaf, "Genotype-phenotype-mapping and neutral variation – a case study in genetic programming," in *Parallel Problem Solving from Nature III*, ser. LNCS, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., vol. 866. Jerusalem: Springer-Verlag, 9-14 Oct. 1994, pp. 322–332. [Online]. Available: ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/papers/ppsn94.ps.gz

[9] A. Wagner, "Robustness, evolvability, and neutrality." *FEBS Letters*, vol. 579, pp. 1772–8, 2005.

[10] M. Ebner, M. Shackleton, and R. Shipman, "How neutral networks influence evolvability," *Complexity*, vol. 7, no. 2, pp. 19–33, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cplx.10021

[11] L. Altenberg, "The evolution of evolvability in genetic programming," in *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. MIT Press, 1994, ch. 3, pp. 47–74. [Online]. Available: http://dynamics.org/~altenber/PAPERS/EEGP/

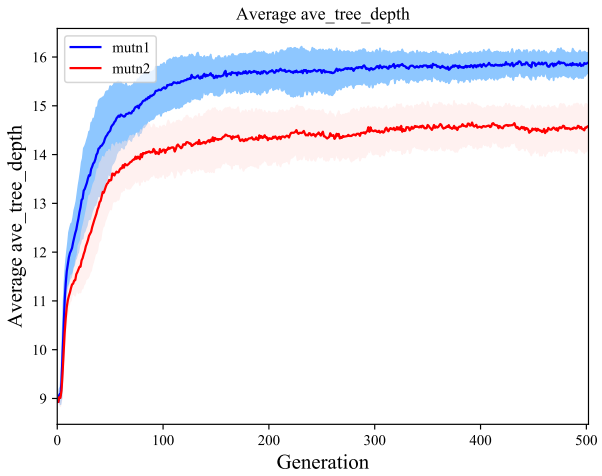
[12] T. Hu, W. Banzhaf, and J. H. Moore, "Robustness and evolvability of recombination in linear genetic programming," in *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, ser. LNCS, K. Krawiec, A. Moraglio, T. Hu, A. S. Uyar, and B. Hu, Eds., vol. 7831. Vienna, Austria: Springer Verlag, 3-5 Apr. 2013, pp. 97–108.

[13] ——, "The effects of recombination on phenotypic exploration and robustness in evolution," *Artificial Life*, vol. 20, no. 4, pp. 457–470, Oct. 2014, ten thousandth GP entry in the genetic programming bibliography.

[14] T. Hu, "Evolvability and rate of evolution in evolutionary computation," Ph.D. dissertation, Department of Computer Science, Memorial University of Newfoundland, ST. John's, Newfoundland, Canada, May 2010. [Online]. Available: http://www.mun.ca/computerscience/graduate/thesis_TingHU.pdf

[15] E. Schulte, Z. P. Fry, E. Fast, W. Weimer, and S. Forrest, "Software mutational robustness," *Genetic Programming and Evolvable Machines*, vol. 15, no. 3, pp. 281–312, Sep. 2014. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.416.675

[16] J. Petke, "New operators for non-functional genetic improvement," in *GI-2017*, J. Petke, D. R. White, W. B. Langdon, and W. Weimer, Eds. Berlin: ACM, 15-19 Jul. 2017, pp. 1541–1542. [Online]. Available: http://geneticimprovementofsoftware.com/



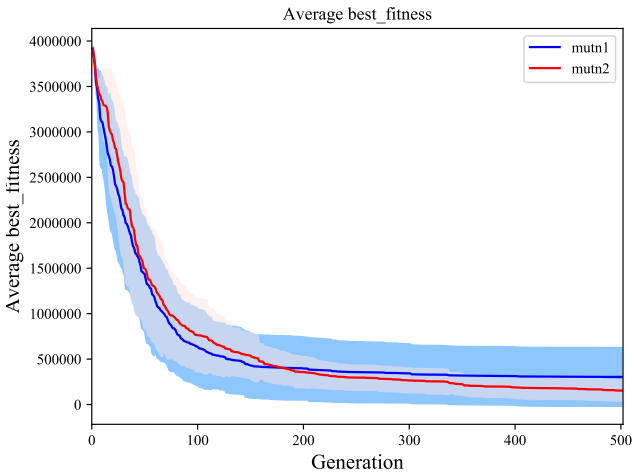


Fig. 6: Sum of Squares (L to R: Tree Depth, Best Fitness)

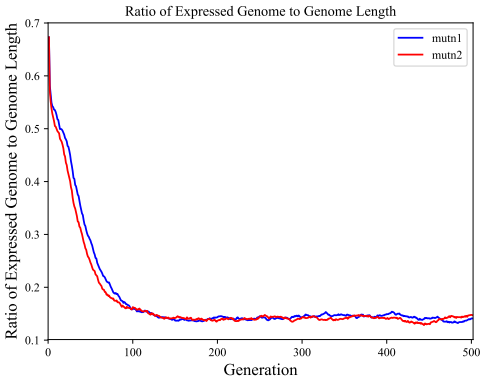


Fig. 7: Ratio of Expressed Genome to Genome Length on Number IO

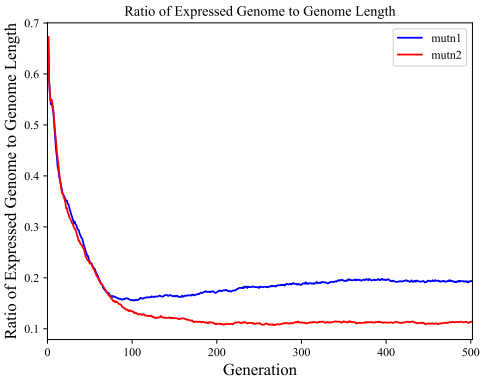


Fig. 9: Ratio of Expressed Genome to Genome Length on Sum of Squares

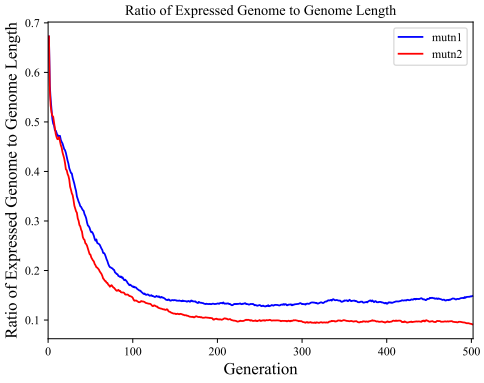


Fig. 8: Ratio of Expressed Genome to Genome Length on Grade

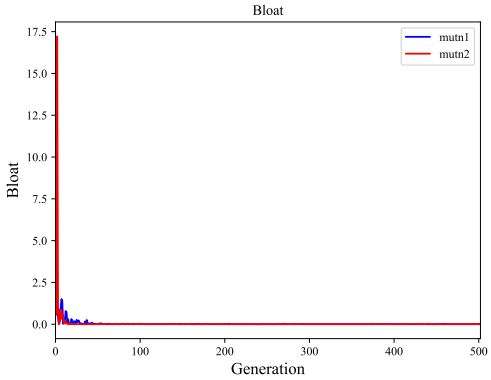


Fig. 10: Bloat on Number IO

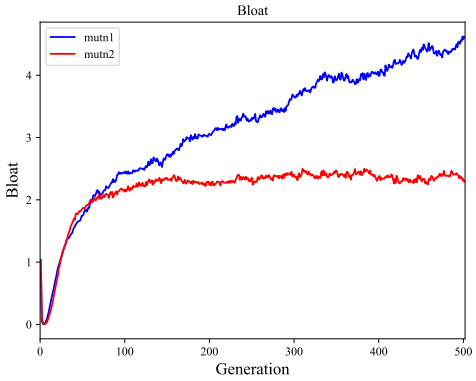


Fig. 11: Bloat on Grade

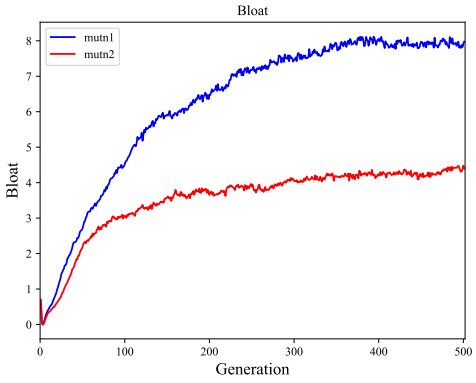


Fig. 12: Bloat on Sum of Squares

wp-content/uploads/2017/05/petke2017_operators.pdf

[17] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, J. Branke, M. Pelikan, E. Alba, D. V. Arnold, J. Bongard, A. Brabazon, J. Branke, M. V. Butz, J. Clune, M. Cohen, K. Deb, A. P. Engelbrecht, N. Krasnogor, J. F. Miller, M. O'Neill, K. Sastry, D. Thierens, J. van Hemert, L. Vanneschi, and C. Witt, Eds. Portland, Oregon, USA: ACM, 7-11 Jul. 2010, pp. 877–884.

[18] S. Dignum and R. Poli, "Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, D. Thierens, H.-G. Beyer, J. Bongard, J. Branke, J. A. Clark, D. Cliff, C. B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K. O. Stanley, T. Stutzle, R. A. Watson, and I. Wegener, Eds., vol. 2. London: ACM Press, 7-11 Jul. 2007, pp. 1588–1595. [Online]. Available: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1588.pdf

[19] ——, "Operator equalisation and bloat free GP," in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, ser. Lecture Notes in Computer Science, M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, and E. Tarantino, Eds., vol. 4971. Naples: Springer, 26-28 Mar. 2008, pp. 110–121.

[20] S. Silva, S. Dignum, and L. Vanneschi, "Operator equalisation for bloat free genetic programming and a survey of bloat control methods," *Genetic Programming and Evolvable Machines*, vol. 13, no. 2, pp. 197–238, Jun. 2012.

[21] M. Brameier and W. Banzhaf, "Neutral variations cause bloat in linear gp," in *Genetic Programming, Proceedings of EuroGP'2003*, ser. LNCS, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610. Essex: Springer-Verlag, 14-16 Apr. 2003, pp. 286–296. [Online]. Available: http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2610&spage=286

[22] E. J. Vladislavleva, G. F. Smits, and D. den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 333–349, Apr. 2009.

[23] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, ser. Genetic programming. Kluwer Academic Publishers, 2003, vol. 4. [Online]. Available: http://www.wkap.nl/prod/b/1-4020-7444-1

[24] I. Dempsey, M. O'Neill, and A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*, ser. Studies in Computational Intelligence. Springer, Apr. 2009, vol. 194. [Online]. Available: http://www.springer.com/engineering/book/978-3-642-00313-4

[25] C. Ryan, M. O'Neill, and J. J. Collins, Eds., *Handbook of Grammatical Evolution*. Springer, Sep. 2018.

[26] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, and M. O'Neill, "PonyGE2: Grammatical evolution in python," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17. Berlin, Germany: ACM, 15-19 Jul. 2017, pp. 1194–1201. [Online]. Available: http://doi.acm.org/10.1145/3067695.3082469

[27] M. O'Neill and D. Fagan, "The elephant in the room: Towards the application of genetic programming to automatic programming." in *Genetic Programming Theory and Practice XVI.*, ser. Genetic and Evolutionary Computation., S. L. Banzhaf W., Spector L., Ed. Springer, 2019, pp. 179–192.

[28] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf, "Open issues in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3/4, pp. 339–363, Sep. 2010, tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[29] T. Helmuth and L. Spector, "General program synthesis benchmark suite," in *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, S. Silva, A. I. Esparcia-Alcazar, M. Lopez-Ibanez, S. Mostaghim, J. Timmis, C. Zarges, L. Correia, T. Soule, M. Giacobini, R. Urbanowicz, Y. Akimoto, T. Glasmachers, F. Fernandez de Vega, A. Hoover, P. Larranaga, M. Soto, C. Cotta, F. B. Pereira, J. Handl, J. Koutnik, A. Gaspar-Cunha, H. Trautmann, J.-B. Mouret, S. Risi, E. Costa, O. Schuetze, K. Krawiec, A. Moraglio, J. F. Miller, P. Widera, S. Cagnoni, J. Merelo, E. Hart, L. Trujillo, M. Kessentini, G. Ochoa, F. Chicano, and C. Doerr, Eds. Madrid, Spain: ACM, 11-15 Jul. 2015, pp. 1039–1046. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754769

[30] S. Forstenlechner, "Program synthesis with grammars and semantics in genetic programming," Ph.D. dissertation, University College Dublin, 2019.