Cartesian Genetic Programming: Why No Bloat?

Andrew James Turner and Julian Francis Miller

Electronics Department
University of York
Heslington, York
YO10 5DD, UK
{andrew.turner,julian.miller}@york.ac.uk

Abstract. For many years now it has been known that Cartesian Genetic Programming (CGP) does not exhibit program bloat. Two possible explanations have been proposed in the literature: neutral genetic drift and length bias. This paper empirically disproves both of these and thus, reopens the question as to why CGP does not suffer from bloat. It has also been shown for CGP that using a very large number of nodes considerably increases the effectiveness of the search. This paper also proposes a new explanation as to why this may be the case.

1 Introduction

Bloat, the uncontrolled growth in program size, is a serious issue for Genetic Programming (GP) that has received much study [1] [2]. However, bloat does not appear in Cartesian Genetic Programming (CGP) [3]. In the literature there are two possible theories as to why CGP does not exhibit bloat; Neutral Genetic Drift (NGD) [3] and length bias [4]. This paper introduces both of these theories and then proceeds to empirically disprove them by removing the underlying assumptions each of them make. This leaves us with no explanation for the lack of bloat in CGP and opens the topic for further investigation.

The investigations also show that there is an evolutionary pressure to increase the program size when the current program size is insufficient[1] to solve a given task. Conversely we find empirically that there is no evolutionary pressure to decrease the program size if the current program size is much larger than required to solve a given task. It therefore appears that using large program sizes is not detrimental to CGP, in keeping with previous results [5] which show it is actually beneficial. A new hypothesis is presented as to why this is the case. When subject to a mutation operator, using a large number of nodes causes, on average, the fitness of an individual to vary by a lesser degree than when using a smaller number of nodes. Using a large number of nodes has smoothed out the fitness landscape making it easier to navigate. This accords with the desirability of synonymous redundancy in representations introduced by Goldberg and

[1] This is compatible with the length bias theory [4] as is discussed later.

M. Nicolau et al. (Eds.): EuroGP 2014, LNCS 8599, pp. 222–233, 2014.

Rothlauf who propose that genotype representations should have the property that mutational neighbours represent similar phenotypes [6].

The remainder of the paper is as follows: Section 2 describes CGP, Section 3 discusses bloat and past theoretical work surrounding bloat and CGP, Section 4 describes a series of experiments which empirically investigate the described theories with the results given in Section 5 and finally Sections 6 and 7 provide a discussion and closing conclusions.

2 Cartesian Genetic Programming

CGP [7] [8] is a form of GP which represents computational structures as directed, usually acyclic graphs indexed by their Cartesian coordinates. Each node may take its inputs from any previous node or program input. The program outputs are taken from the output of any internal node or program inputs. This structure leads to many of the nodes described by the CGP chromosome not contributing to the final operation of the phenotype, these inactive, or "junk", nodes have been shown to greatly aid the evolutionary search [5] [9] [10].

The nodes described by CGP chromosomes are arranged in a rectangular $r \times c$ grid of nodes, where r and c respectively denote the user-defined number of rows and columns. In CGP, nodes in the same column are not allowed to be connected together. It is important to note that any architecture (limited by the number of nodes) can be constructed by arranging the nodes in a $1 \times n$ format where the n represents the maximum number of nodes (columns). Using this representation the user does not need to specify the topology, which is then automatically evolved along with the program.

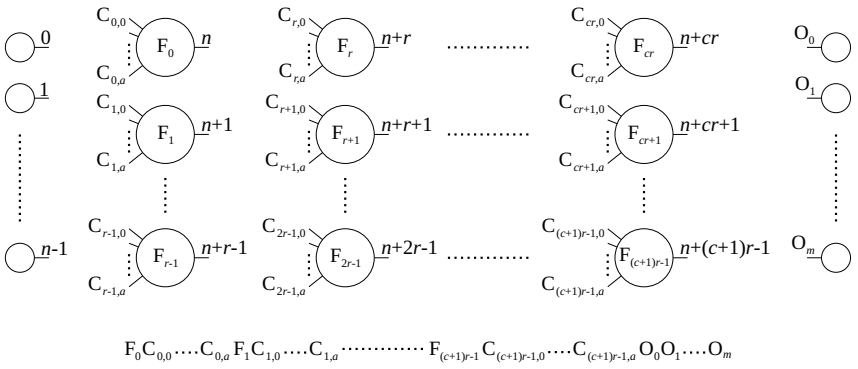


Fig. 1: Depiction of a Cartesian Genetic Programs structure with chromosome encoding below, taken from [8]

Figure 1 gives the general form of a CGP showing that a CGP chromosome can describe multiple input multiple output programs with a range of node

transfer functions and arities. In the chromosome string, also given in Figure 1, F_i denote the function operation at each node, C_i index where the node gathers its inputs and each O_i denote which nodes provide the outputs of the program. It should be noted that CGP is not limited to only one data type, it may be used for Boolean values, floats, images, audio files, videos etc. CGP generally uses the Evolutionary Strategy (ES) algorithm $(1 + \lambda)$-ES. In this algorithm each generation contains $1 + \lambda$ candidates and the fittest is chosen as the parent. The next generation is formed by this parent and λ offspring obtained through mutation of the parent. It is important to note that if no offspring are fitter than the parent, but at least one has the same fitness as the parent, then the offspring is chosen as the new parent. In CGP, the λ value is commonly set as four. The connection genes in the chromosomes are initialised with random values that obey the constraints imposed by the CGP structural parameters r and c. The function genes are randomly chosen from the allowed values in the function lookup table. The output genes O_i are randomly initialised to refer to any node or input in the graph. The standard mutation operator used in CGP works by randomly choosing valid alleles at a randomly chosen gene locations. The reason why both a simple operator and a simple evolutionary algorithm are so effective is related to the presence of non-coding genes. Simple mutations can connect or disconnect whole sub-programs. For a more detailed description of CGP see [8].

3 Bloat and CGP

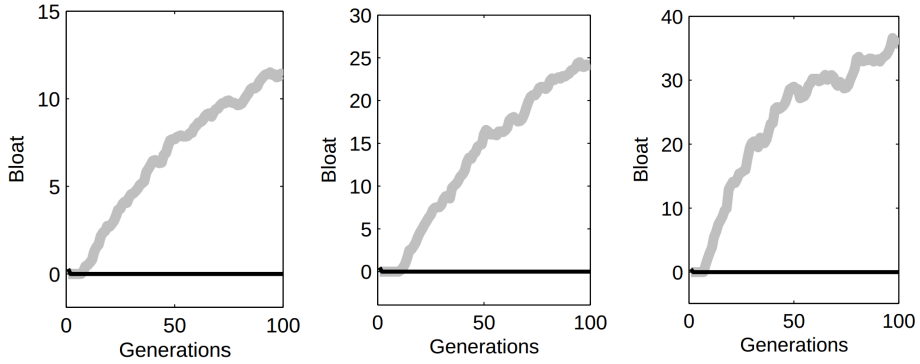
Bloat can be defined as "program growth without (significant) return in terms of fitness" [11], that is, if program length is increasing disproportionately to fitness improvements then bloat is said to be occurring. This definition has been formally stated as a metric which measures the amount of bloat on any given generation [12]. Here we use a variation on this bloat equation is given in Equation 1:

$$B(g) = \frac{N(g)}{D(g)}, \quad N(g) = \frac{\hat{A}(g) - \bar{A}(0)}{\bar{A}(0)}, \quad D(g) = \frac{\bar{F}(0) - \hat{F}(g)}{\bar{F}(0)} \tag{1}$$

Where $B(g)$ is the bloat at generation g, $\hat{A}(g)$ is the number of active nodes used by the fittest member of the population at generation g, $\bar{A}(0)$ is the average number of active nodes used by the population at generation 0, $\bar{F}(0)$ is the average fitness of the population at generation 0 and $\hat{F}(g)$ is the fitness of the fittest member of the population at generation g. Equation 1 holds when the target is to minimise the fitness to zero. When the fitness is to be maximised the fitness values can be amended by subtracting the current fitness from the target fitness; thus transforming the problem into a minimisation task. The equation effectively gives the ratio of increase in program size to improvement in fitness. If the program size is increasing disproportionately to fitness then the bloat value will also increase, thus indicating bloat.

The bloat equation given in [12] was adapted here to show the amount of bloat exhibited by the fittest member of the population; as opposed to the average bloat of the population. There are two reasons for this alteration: 1) CGP uses a $(1 + \lambda)$-ES without crossover, and so the only solution of interest is the current fittest. 2) The small population sizes typically used by CGP leads to very noisy average active nodes and fitness values which are hard to analyse graphically.

Figure 2 gives three examples of the unaltered bloat metric when used by the original authors; see [12] further details of their experiments. As can be seen in Figure 2, bloat is easily detected by a high continuous increase in the bloat metric.



(a) $f(x) = log(x)$ **(b)** %F **(c)** LD50

Fig. 2: The bloat metric comparing standard GP (light gray) and DynOpEq GP (black) on (a) symbolic regression and (b)(c) two real world classification tasks. Images taken from [12].

Although CGP uses fixed size genotypes, each genotype can encode phenotypes (programs) of different lengths. This is because many of the genes in the genotypes are typically inactive or "junk" and are therefore not decoded into the phenotype. If bloat occurred in CGP it would via a disproportionate increase in active nodes with respect to fitness. The following subsections introduce two theories found in the literature which have been proposed to explain why CGP does not bloat.

3.1 Neutral Genetic Drift

One of the many theories surrounding why GP in general suffers from bloat is the drift hypothesis [13]. The drift hypothesis goes as follows. When a population is trapped in a local optimum many of the parents children will have the same or very similar fitness. A method often used by GP is to replace parents with their children if their fitness is equal or very similar; with the aim to improve genetic

diversity and to escape local optima with future mutations. If adding or removing a small number of nodes does not lessen the fitness of the child then the child may be larger or smaller respectively. Additionally it has been shown that for a given chromosome size there exist more solutions with the same fitness which are larger than smaller [14]. Therefore there exists an evolutionary pressure to increase the size of the program when trapped in local optima.

It is argued in [3] that CGP does not suffer from bloat due to the inactive genes causing NGD [15]. Their argument is that when a population is trapped in a local optimum the majority of the mutations which do not cause a reduction in fitness will be mutations affecting inactive genes; as opposed to active genes. Mutating inactive genes cannot alter the program size, therefore CGP does not increase in length. However mutating inactive genes alone cannot help the population escape the local optima, but the activation of previously inactive genetic material can. This effect is strengthened when the inactive genetic material is continuously changing as it causes the possible phenotypes one mutation away to also continuously change; meaning that a wide area of the search space can be sampled generation to generation. The term given to this continuously changing inactive genetic material is NGD and it is this, coupled with non-coding genes, which is thought to be the cause of CGP not exhibiting bloat.

3.2 Length Bias

Length bias offers an alternative argument for why CGP does not suffer from bloat [4]. Length bias shows that nodes positioned closer to the chromosome inputs are much more likely to be active than those positioned nearer the outputs. This is because when CGP encodes feed-forward (acyclic) networks each node can only gather its inputs from previous nodes i.e. those closer to the inputs. This means that nodes closer to the inputs have a higher probability of being active; as the probability of any given node being an active node is directly proportional to the number of nodes which can connect to that node. This results in a higher concentration of active nodes towards the inputs. This bias towards small networks is why CGP does not suffer from bloat.

4 Experiments

The aim of the experiments presented is to identify if NGD, length bias or another factor is responsible for the lack of bloat in CGP. This is achieved by removing the main assumption behind each theory as to why CGP does not suffer from bloat. For the NGD theory this is achieved by preventing NGD from occurring and for the length bias theory this is achieved by removing the length bias. The results obtained on each task are also compared to a neutral search, to ensure that the fitness functions used are not producing a pressure to create small programs.

All of the experiments are investigated using the six bit parity and the Pagie1 [16] symbolic regression tasks. The parity task uses AND NAND OR and NOR

node functions[2] and the fitness is calculated as the number of incorrect outputs produced when all possible inputs are swept. The Pagie1 task, Equation 2, uses $+ - \times \% \; e^n$ and $\ln(|n|)$ node functions and the fitness is calculated as the sum of the absolute differences between the correct and actual outputs when both inputs are swept from -5 to 5 in 0.4 increments. In all cases, unless otherwise stated, the following parameters are used: $(1+4)$-ES, three percent probabilistic mutation[3], one hundred columns, one row and allowed ten thousand generations before terminating the search. Each experiment is repeated fifty times in order to produce reliable averages.

$$y(x_1, x_2) = \frac{1}{1 + x_1^{-4}} + \frac{1}{1 + x_2^{-4}} \tag{2}$$

4.1 Regular CGP

The first experiment is to apply regular unaltered feed-forward CGP to the two tasks. This is to confirm the result that CGP does not exhibit bloat [3] and provide results to which the other experiments can be compared against.

4.2 No Neutral Genetic Drift

The NGD theory as to why CGP does not suffer from bloat is reliant upon CGP actually exhibiting NGD. NGD can be prevented in CGP by only allowing *active* genes to be mutated. This causes the inactive genetic material to become static and thus cannot drift. Inactive nodes can still become active however if an active node connects to them when mutated. By only allowing active genes to be mutated, CGP is functionally equivalent but without NGD.

If CGP without NGD is shown not to exhibit bloat then NGD cannot be the cause of CGP not bloating. Conversely if CGP without NGD is shown to exhibit bloat then NGD must be the cause of CGP not bloating.

Interestingly the method of only allowing active genes to be mutated has the opposite goal of accumulating mutation [17], a CGP mutation method designed to heighten NGD.

4.3 Recurrent CGP

The length bias theory as to why CGP does not suffer from bloat is reliant upon CGP exhibiting a length bias. Length bias occurs as nodes can only connect to previous nodes in the network. However, if this restraint is removed then length bias no longer applies. This restraint can be removed by placing no restrictions on where each node can gather its inputs i.e. by allowing recurrent as well as feed forward connections. This form of CGP is referred to as recurrent CGP as it allows for recurrent connections. Allowing recurrent connections means that

[2] The XOR gate is omitted to increase the difficulty of the tasks.
[3] Where each gene is mutated with a given probability.

the probability of a given node being active is no longer a function of its position within the genotype. Therefore length bias has been removed.

The implementation of recurrent CGP is identical to that of feed-forward CGP except that no restraints are placed on where each node can connect its inputs. Under these conditions it is possible for a node to be used as an input to another node before it has calculated its own output value. Therefore, before each fitness evaluation all of the active nodes are initialised to output zero. During the fitness functions the outputs are read from the program in the same way as for regular feed-forward CGP: 1) apply a set of inputs 2) update every active node once from inputs to outputs (allowing the clocked feedback) 3) the program results are read from the output nodes.

If recurrent CGP does not suffer from bloat then the cause of feed-forward CGP not exhibiting bloat cannot be due to length bias. However if recurrent CGP does suffer from bloat then the cause of feed-forward CGP not exhibiting bloat must be due to length bias.

4.4 Neutral Search

It is possible that the fitness functions used to investigate bloat may themselves produce a pressure to create small program sizes; for instance if they require small program sizes to solve the given task. Although this is unlikely, it should be investigated and found to be untrue in order for the results of the other experiments to be valid.

This is achieved by comparing the percentage of active nodes used by the six bit parity and Pagie1 tasks with the percentage used by a neutral search. A neutral search is where the fitness is set to zero regardless of the programs functionality i.e. it is a neutral fitness landscape. If it is shown that the six bit parity and Pagie1 tasks use a lower percentage of active nodes than that used by a neutral search then it would indicate that these tasks are applying an evolutionary pressure to produce small program sizes which could be responsible for CGP not bloating.

In order to make a fair comparison between the percentage of active nodes used by neutral search and the six bit parity and Pagie1 tasks, the number of inputs and outputs of the evolved programs must be consistent. That is, when comparing neutral search to the six bit parity task the neutral search must also evolve solutions with six inputs and one output; and equivalently for the Pagie1 task. This is because the number of inputs and outputs is likely to influence the percentage of active nodes.

The experiment is investigated for 1, 5, 10, 50, 100, 500 and 1000 nodes (columns with rows set to one), to identify if the results vary over a range of topology limits.

5 Results

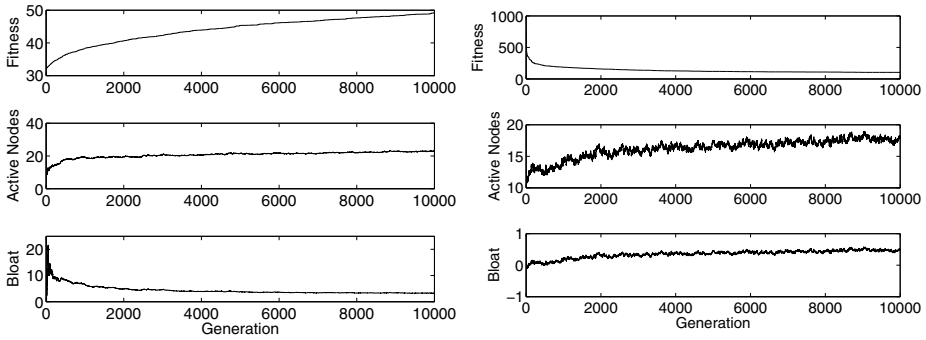
The results of the described experiments are now presented. In all but the neutral search experiments, the results are given graphically showing the fitness,

number of active nodes and bloat values of the best member of the population at each generation averaged over the fifty runs. The bloat value is calculated using Equation 1. The technique is identified as bloating if the bloat value rises continuously throughout evolutionary time.

5.1 Regular CGP

The results of applying regular feed-forward CGP to the six bit parity and Pagie1 tasks are given in Figure 3. Here it can be seen that CGP is not exhibiting bloat during evolution with the bloat value actually falling in the six bit parity case. Although it can be seen that the number of active nodes does increase over evolutionary time, it does so only when the fitness also improves and is therefore not bloat as defined in Section 3.

The initial high values of bloat seen in Figure 3 for the six bit parity task is thought to be because of the high increase in active nodes during the beginning of the search. It appears that the initial randomly generated chromosomes have too few active nodes to solve the task. This causes a sharp increase in the number of active nodes during the first few generations. This appears in the bloat value until these additional active nodes causes a significant increase in fitness at which point the bloat value starts to fall.



(a) Six Bit Parity **(b)** Pagie1

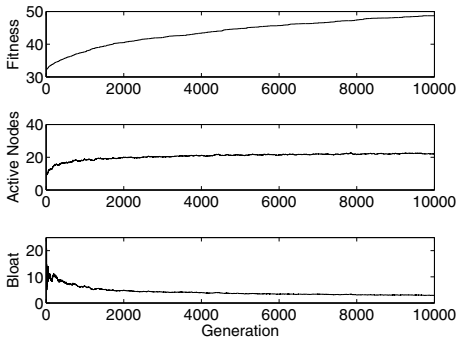
Fig. 3: Regular feed-forward CGP

5.2 No Neutral Genetic Drift

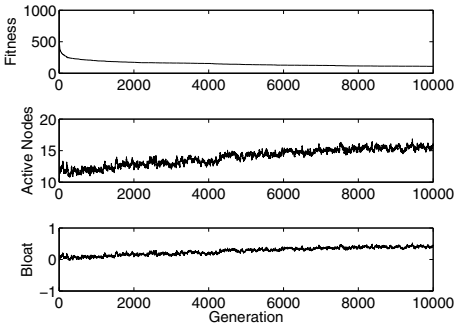
The results of applying CGP without NGD are given in Figure 4 for both the six bit parity and Pagie1 tasks. Here it can be seen that CGP without NGD is still not exhibiting bloat and so NGD cannot be the cause of CGPs lack of bloat.

5.3 Recurrent CGP

The results of applying recurrent CGP are given in Figure 5 for both the six bit parity and Pagie1 tasks. Here it can be seen that recurrent CGP is still not exhibiting bloat and so length bias cannot be the cause of CGPs lack of bloat.

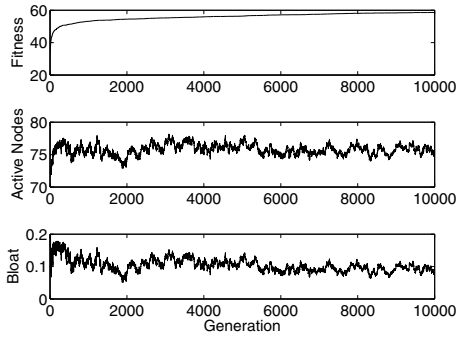


(a) Six Bit Parity

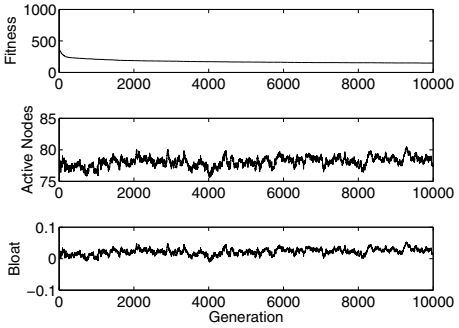


(b) Pagie1

Fig. 4: Feed-forward CGP without NGD



(a) Six Bit Parity



(b) Pagie1

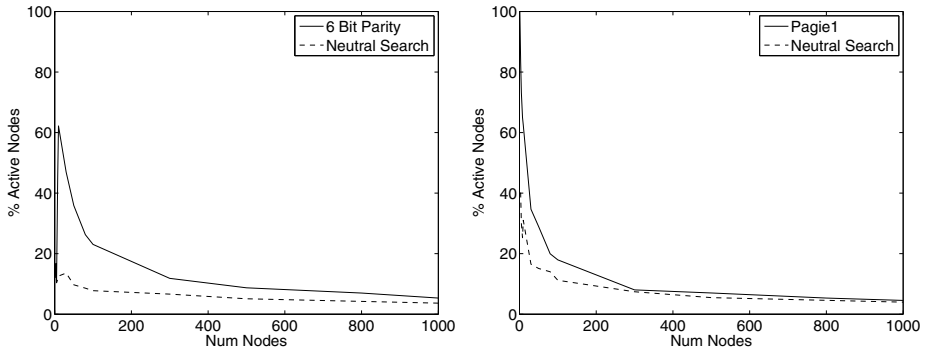
Fig. 5: Recurrent CGP

5.4 Neutral Search

Figure 6 gives a comparison between the percentage of active nodes, over a range of available nodes[4], for the neutral and guided search problems[5]. The figure clearly shows that for small numbers of available nodes the percentage of active nodes used by the guided searches far exceeds the percentage used by the neutral searches. However, for higher numbers of available nodes the percentage of active nodes used by the guided searches approach that used by the neutral searches. Therefore it can be concluded that the six bit parity and Pagie1 tasks are not producing an evolutionary pressure to create small program sizes and are therefore not responsible for the observed lack of bloat in the previous results.

[4] Where available nodes refers to the product of the rows and columns. As rows was always set as one however available nodes and columns are equivalent.

[5] Where a guided search is the opposite to a neutral search i.e. toward a real task.



(a) Six Bit Parity **(b)** Pagie1

Fig. 6: Regular feed-forward CGP applied to the two tasks and equivalent neutral searches

6 Discussion

As is seen in Section 5.2, CGP without NGD does not exhibit bloat and so NGD is unlikely to be the cause of CGP not exhibiting bloat. However it is important to note that this result does not suggest that the presence of inactive genes themselves are not responsible. It was shown in Section 5.3 that recurrent CGP also does not suffer from bloat, and as length bias only applies to feed-forward CGP it therefore cannot be the cause of CGP not exhibiting bloat. It was also shown in Section 5.4 that the tasks used to study CGP and bloat did not themselves produce a bias towards small program lengths; strengthening the conclusions.

Interestingly it is reported in [4] that CGP struggles to increase the number of active nodes during evolution even when a given task requires it; due to length bias. In Section 5.4 however it can be seen that CGP consistently used more active nodes on both tasks than for the neutral searches when given a low number of available nodes. This indicates that CGP is increasing the number of active nodes when the task requires it. This effect is also seen in the increasing number of active nodes during evolution in Sections 5.1 and 5.2. This however is in keeping with the results found in [4] which investigated the effect of length bias in extreme cases where it was required that CGP used a very high percentage of active nodes; here the experiments were for real tasks typical of the applications of CGP.

It has been shown previously for CGP that using a large number of available nodes aids the search considerably [5]. This was thought to be because large numbers of available nodes produced a high percentage of inactive nodes aiding the search through NGD. However it was later shown in [4] that length bias causes very few inactive nodes to be present among the active nodes, weakening the effect. Interestingly, the results given in Section 6 show that when CGP is allowed a very large number of nodes there is no evolutionary pressure to use less

active nodes than that used by a neutral search i.e. there is no pressure to increase smaller program sizes. Based on this result the authors propose an alternative explanation. Consider a genotype for which the phenotype consists of a small number of active nodes, any single connection gene mutation is likely to have a large effect on the operation of that phenotype. If however a genotype encodes a phenotype with a high number of active nodes, any single connection gene mutation is likely, on average, to have a much smaller effect on the operation of the overall phenotype. Therefore, using a high number of nodes creates a search space in which the fitness changes more gradually with a given number of connection gene mutations. This smoother search space is likely to be easier for evolution to navigate and hence make for a more efficient search. The reason this does not result in CGP evolving larger and larger program sizes is because these larger programs are not fitter, they are more *evolvable* and therefore there is no direct pressure to increase the program size. However, this hypothesis currently has no empirical evidence and is left for future investigation.

Another interesting result is that recurrent CGP outperformed feed-forward CGP on the six bit parity task[6]; a task which does not require recurrent connections. This is due to the fixed order of inputs applied to each circuit when evaluating the fitness function. Recurrent CGP was producing the correct outputs based on the current inputs *and* previous inputs, not on the current inputs alone. Although these evolved circuits would therefore not operate correctly as parity generators, it does show the ingenuity of evolution and how using recurrent programs for feed-forward tasks can provide an unexpected, albeit unfair, advantage.

7 Conclusion

Although this paper does not give a possible cause of CGP not exhibiting bloat, it does help disprove two previous explanations found in the literature; namely NGD and length bias. Additionally it has been shown that CGP increases the number of active nodes when a given task requires it; although this effect has limitations as shown in [4]. A new hypothesis has also been presented as to why using large numbers of available nodes is beneficial for CGP. That is, using large program sizes could help smooth out the search space making for easier navigation.

References

1. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3), 309–344 (2006)
2. Silva, S., Costa, E.: Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genetic Programming and Evolvable Machines 10(2), 141–179 (2009)

[6] After ten thousand generations feed-forward CGP scored an average fitness of 49.22 whereas recurrent CGP scored an average fitness of 58.62.

3. Miller, J.: What bloat? Cartesian genetic programming on Boolean problems. In: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp. 295–302 (2001)
4. Goldman, B.W., Punch, W.F.: Length bias and search limitations in Cartesian genetic programming. In: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, pp. 933–940. ACM (2013)
5. Miller, J., Smith, S.: Redundancy and computational efficiency in Cartesian genetic programming. IEEE Transactions on Evolutionary Computation 10(2), 167–174 (2006)
6. Rothlauf, F., Goldberg, D.E.: Representations for Genetic and Evolutionary Algorithms. Physica-Verlag (2002)
7. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)
8. Miller, D.J.F. (ed.): Cartesian Genetic Programming. Springer (2011)
9. Vassilev, V.K., Miller, J.F.: The Advantages of Landscape Neutrality in Digital Circuit Evolution. In: Miller, J.F., Thompson, A., Thompson, P., Fogarty, T.C. (eds.) ICES 2000. LNCS, vol. 1801, pp. 252–263. Springer, Heidelberg (2000)
10. Yu, T., Miller, J.F.: Neutrality and the evolvability of boolean function landscape. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 204–217. Springer, Heidelberg (2001)
11. Poli, R., Langdon, W.W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming (2008), Published via, `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`
12. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 877–884. ACM (2010)
13. Soule, T., Heckendorn, R.B.: An analysis of the causes of code growth in genetic programming. Genetic Programming and Evolvable Machines 3(3), 283–309 (2002)
14. Langdon, W., Soule, T., Poli, R., Foster, J.: The evolution of size and shape. Advances in Genetic Programming 3, 163 (1999)
15. Kimura, M.: The neutral theory of molecular evolution. Cambridge University Press (1984)
16. McDermott, J., White, D.R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., et al.: Genetic programming needs better benchmarks. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, pp. 791–798. ACM (2012)
17. Goldman, B.W., Punch, W.F.: Reducing wasted evaluations in cartesian genetic programming. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş., Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 61–72. Springer, Heidelberg (2013)