Size-Based Tournaments for Node Selection

Thomas Helmuth
Computer Science
University of Massachusetts
Amherst, MA 01003
thelmuth@cs.umass.edu

Lee Spector
Cognitive Science
Hampshire College
Amherst, MA 01002
lspector@hampshire.edu

Brian Martin
Cognitive Science
Hampshire College
Amherst, MA 01002
btm08@hampshire.edu

ABSTRACT

In genetic programming, the reproductive operators of crossover and mutation both require the selection of nodes from the reproducing individuals. Both unbiased random selection and Koza 90/10 mechanisms remain popular, despite their arbitrary natures and a lack of evidence for their effectiveness. It is generally considered problematic to select from all nodes with a uniform distribution, since this causes terminal nodes to be selected most of the time. This can limit the complexity of program fragments that can be exchanged in crossover, and it may also lead to code bloat when leaf nodes are replaced with larger new subtrees during mutation. We present a new node selection method that selects nodes based on a tournament, from which the largest participating subtree is selected. We show 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.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program modification*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

genetic programming, node selection, tournaments, crossover, mutation, bloat

1. INTRODUCTION

In standard, tree-based genetic programming (GP), the node selection mechanism determines how nodes are selected from individuals for genetic operations. Many mechanisms for node selection, while non-deterministic, favor the selection of certain nodes over others. The method of biasing node selection has an impact on the size, shape, and location of the nodes that are replaced and transfered during crossover and mutation, which in turn affects the progression of evolution. We base the performance of a node selection technique primarily on its effectiveness at finding solutions and secondarily on its ability to control code bloat.

With a small number of exceptions[1], most GP practitioners use one of two common node selection techniques. The first, unbiased selection, chooses a random node with uniform distribution from all nodes in the individual. This method, while conceptually simple, does not differentiate between nodes of different sizes or locations. Since trees with average branching factors greater than or equal to 2 have more terminal nodes than internal nodes, unbiased selection chooses terminal nodes more frequently than any other size of node. In general, it is believed that swapping small subtrees, or *building blocks*, through crossover is more likely to create better children than by swapping terminals.

Many practitioners avoid unbiased selection in order to select non-terminal nodes more often during crossover and mutation. The most often used alternative, which we will call Koza 90/10, was introduced in [7]. This technique selects an internal node with 90% probability, and a terminal node with 10% probability, biasing the selection towards internal nodes. When selecting an internal node, all internal nodes are given equal weight, which means that smaller internal nodes are selected more than larger ones, for the same reasons that terminal nodes are selected most often with unbiased selection. Additionally, the arbitrary choice of 90% and 10% leaves open the question of better percentages or additional levels for different sized internal nodes. Despite these drawbacks, Koza 90/10 remains very popular, even with little evidence for beneficial effects on performance or bloat prevention.

We introduce *size-based tournaments*, a simple node selection mechanism that differentiates between nodes of different sizes in order to improve performance and control code bloat. This mechanism uses tournaments to choose a node, similar to the tournaments commonly used in tournament selection of parent individuals. We show that size-based tournaments improve performance on three standard test problems compared to unbiased and Koza 90/10 mechanisms. Additionally, our selection mechanism does not increase code bloat compared to the other mechanisms.

[1]Prior work on less frequently used node selection techniques is described in Section 3.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

Table 1: Parameters for experiments.

Parameter	Symbolic Regression	Artificial Ant	11-Multiplexer
Runs	500	500	500
Population Size	500	500	100
Maximum Generations	51	51	101
Crossover Probability	0.9	0.9	0.9
Mutation Probability	0.05	0.05	0.05
Reproduction Probability	0.05	0.05	0.05
Max Depth	17	17	17
Max Depth for Generated Code	5	5	5
Initialization Procedure	Ramped-Half-And-Half	Ramped-Half-And-Half	Ramped-Half-And-Half

Following this section, we give a brief overview of code bloat. In the Section 3 we discuss prior node selection mechanisms that have been proposed in the literature to improve performance or control code bloat. The algorithm for size-based tournaments is discussed in Section 4. In Section 5, we discuss the testing of node selection methods on three problems and give our results. Finally, we discuss the results and give directions for further research.

2. CODE BLOAT

Growth in program sizes during runs, known as *code bloat*, has harmful effects on the solution finding abilities of evolution in many domains. Researchers have made many attempts to avoid it [9, 10, 12, 8], as well as posited numerous explanations for it, including its utility as a defense against destructive crossover and removal bias.

Code bloat as a *defense against destructive crossovers* refers to the fact that as the ratio of viable to inviable code decreases, crossover is less likely to reduce fitness [1]. Stated differently, as the number of *inviable* subtrees – those which may be exchanged with no effect on fitness – increases, the likelihood of crossover affecting viable code, and therefore reducing fitness, decreases. Evolution applies selective pressure for larger programs with more inviable code, which are less negatively affected by crossover.

Removal bias refers to the tendency for a fitness neutral offspring produced by crossover to be larger than its parent [11]. Soule and Foster argue that this occurs because inviable subtrees tend to be smaller than average. Thus, the average-sized subtrees that replace them increase the size of the offspring.

A more detailed discussion of the causes of code bloat is beyond our scope; it suffices to say that the usefulness of the presented technique does not depend on any single cause of code bloat. In fact, our mechanism's possible bloat reducing properties are of secondary importance; we are primarily interested in its performance improvements and provide little analysis of bloat in our later experiments.

3. PRIOR WORK

Depth-dependent crossover [3, 4] attempts to select a higher percentage of building blocks by choosing nodes based on their depth from the root. A depth is randomly selected, with a bias towards selecting shallower nodes. After the depth is selected, a node at that depth is chosen based on a uniform random selection. Depth-dependent crossover only partially solves the problem of bias towards selecting small nodes. Both small and large nodes may be present at the shallower depths from which many of the nodes are selected using depth-dependent crossover. The authors try to solve this problem by revising their algorithm to select a node from a given depth by using a subtree-size-based selection ratio, which weights the selection of nodes at a level based on the size of their subtree. This begins to look like our size-based tournaments, except that it uses a more complex and arbitrary mechanism for selection.

In prior work on *size fair* genetic operators, measures are taken to ensure that new code introduced into programs by genetic operators is roughly the same size as old code that it replaces [8, 2]. This may achieve some of the goals for which node selection methods are sometimes employed, for example with regard to program bloat, but it does not directly address the size or location of nodes selected for modification. Additionally, size fair operators require more complicated implementation techniques, such as the generation of code of a specific size for size fair mutation.

4. SIZE-BASED TOURNAMENTS

Size-based tournaments take their inspiration from tournaments that are commonly used for parent selection in evolutionary algorithms. When a node needs to be chosen for crossover or mutation, we first randomly choose n nodes with a uniform distribution to participate in the tournament. Then, the largest of the n nodes is selected for use by the genetic operator, where size is defined as the number of nodes in the subtree.

Size-based tournaments bias node selection toward larger subtrees, and can be tuned by changing the tournament size. The motivations for size-based tournaments are the same those for Koza 90/10 and depth-dependent crossover, while using a simpler mechanism than these techniques. Size-based tournaments differentiate between internal nodes of different sizes, whereas Koza 90/10 treats all internal nodes equally. Our mechanism favors larger nodes, whereas depth-dependent crossover prefers shallower nodes, which are not necessarily larger.

Size-based tournaments are simple to understand and implement, especially for a GP practitioner who is familiar with parent selection tournaments. Only one parameter is required, the tournament size n. Unbiased selection can be seen as a special case of size-based tournament selection with a tournament size of 1. Furthermore, the simplicity of size-based tournament selection makes it easier to use, with fewer parameters that must be fine-tuned to the problem.

Table 2: Performance results for symbolic regression with 500 runs. Methods labeled "Tourn n" represent size-based tournaments of size n.

Method	Computational Effort	Successes
Unbiased	120,000	146
Koza 90/10	123,500	138
Tourn 2	77,000	206
Tourn 3	96,000	175
Tourn 4	136,000	151
Tourn 5	288,000	85

5. EXPERIMENTS AND RESULTS

We have tested size-based tournaments with a range of tournament sizes against unbiased node selection and Koza 90/10 selection on three standard problems: symbolic regression, artificial ant, and 11-multiplexer. We conducted these runs using ECJ and the parameter settings in Table 1. All problems come from the standard formulations described by Koza [7].

In order to compare the performances of the node selection methods, we computed the number of successes and the computational effort for each set of runs. Computational effort is a measure of the expected number of fitness evaluations that the algorithm would have to perform to have a 99% confidence of finding a solution. A lower computational effort signifies a more efficient algorithm, in that it will likely be able to find a solution using fewer fitness evaluations. We also collected and analyzed the mean best fitnesses of the runs. The results are consistent with the conclusions presented below, but space limitations prevent us from presenting a full discussion of this data here.

5.1 Symbolic Regression

We tested the node selection methods on symbolic regression using the function $f(x) = x^4+x^3+x^2+x$. Each run used 20 randomly selected test cases from the range $[-1, 1]$. Only one terminal, x, was used in this problem, along with eight internal operators: Add, Mul, Sub, Div, Sin, Cos, Exp, and Log. The first four operators have arity 2, and the remainder have arity 1. Since the average arity of the operators is 1.5, program trees in this domain may have more internal nodes than terminals. Accordingly, unbiased selection is expected to perform similarly to Koza 90/10 for this problem.

The number of successes and computational efforts for this problem are given in Table 2. Size-based tournaments of size 2 and 3 do best on this problem, as shown by their success rates and computational efforts. Tournaments of size 5 have the worst performance. The remaining methods, including unbiased and Koza 90/10, fall in between.

Figure 1 shows mean program sizes throughout each set of runs. All node selection methods display code bloat, with some more resistant than others. It is important to note that the majority of solutions found for all selection mechanisms fall between generations 4 and 14. This indicates that this symbolic regression problem favors small program sizes. But, this does not explain the differences in performance between different node selection mechanisms. All methods have small programs near the beginning and experience code bloat, yet performance does not appear to correlate with mean program sizes. For example, Koza 90/10 and size-based tournaments of size 2 have very similar mean size growth curves, yet have significantly different numbers of successes and computational efforts. Thus a node selection method's influence on program sizes is not the primary cause of changes in performance.

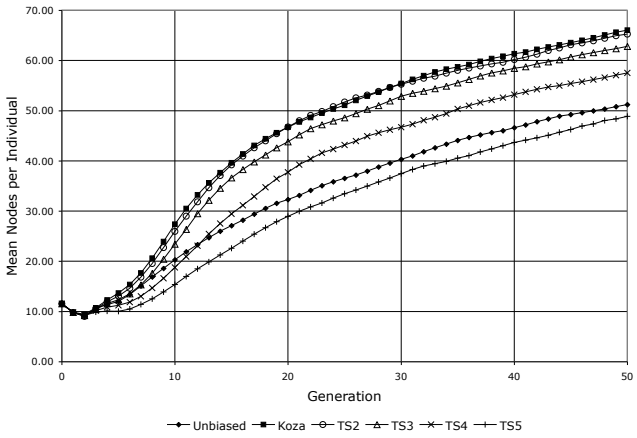

Figure 1: Mean program sizes during each set of runs. TSn represents size-based tournament selection with a tournament size of n.

Table 3: Performance results for the artificial ant problem with 500 runs. Methods labeled "Tourn n" represent size-based tournaments of size n.

Method	Computational Effort	Successes
Unbiased	2,203,500	22
Koza 90/10	4,560,000	12
Tourn 2	2,214,000	24
Tourn 3	2,016,000	27
Tourn 4	4,368,000	12
Tourn 5	5,461,000	9

5.2 Artificial Ant

Next, we used the node selection methods on the artificial ant problem on the Santa Fe trail. The terminals for this problem are Left, Right, and Move, and the operators are IfFoodAhead and Progn2 with arity 2, and Progn3 with arity 3.

Table 3 shows the performance measures of the different node selection methods on the artificial ant problem. We found that unbiased selection and size-based tournament selection with tournament sizes of 2 and 3 produced approximately double the number of solutions and had about half of the computational effort compared to Koza 90/10 and size-based tournament selection with tournament sizes of 4 and 5.

The mean program sizes curves look very similar to those for the symbolic regression problem, and are therefore omitted. On the other hand, solutions for all node selection methods were discovered throughout the runs, without any correlation to program sizes. This indicates that solving the artificial ant problem is not as dependent on size of tree as the symbolic regression problem, with solutions being found at a wide range of tree sizes. Additionally, no correlation was found between growth rates and performance measures.

Table 4: Performance results for the 11-multiplexer problem with 500 runs. Methods labeled "Tourn n" represent size-based tournaments of size n.

Method	Computational Effort	Successes
Unbiased	1,044,000	178
Koza 90/10	1,207,000	154
Tourn 2	924,000	199
Tourn 3	1,000,000	185
Tourn 4	910,000	188
Tourn 5	1,044,000	170

5.3 11-Multiplexer

Finally, we tested each node selection method on the 11-Multiplexer problem. This problem uses 11 terminals and the four operators And, Or, Not, and If. The operators And and Or have arity 2, Not has arity 1, and If has arity 3.

We show the number of successes and the computational effort for each node selection method in Table 4. The results show that all size-based tournament selection runs outperformed Koza 90/10, and either outperformed or essentially tied unbiased selection. Size-based tournaments of size 4 had the best computational effort, and those of size 2 found the most solutions.

Again, the mean program sizes curves look very similar to those for the symbolic regression problem, and are therefore omitted. As with the artificial ant problem, solutions were discovered throughout runs, and no correlation is seen between growth rates and performance.

6. CONCLUSIONS AND FUTURE WORK

We have described size-based tournaments, a new method for node selection that is simple in concept and implementation while providing performance benefits over more complex and arbitrary algorithms. We have shown size-based tournaments to be a principled and effective method for selecting nodes by testing it on three common test problems. Additionally, our results indicate weaknesses of Koza 90/10 selection, and to a lesser extent, unbiased node selection.

We have not tested size-based tournaments on more difficult problems, such as those that would produce human-competitive results. Testing on such problems may show different results than those for these small problems. Another avenue of future work would be to experiment with dynamic or adaptive tournament sizes throughout a run. For instance, tournament sizes could start large and decrease throughout a run, an idea that has conceptual links to simulated annealing [6]. Or, tournament size could be linked to individuals in an adaptive fashion, similar to self-tuning depth-dependent crossover [5].

Acknowledgments

We would like to thank Kyle Harrington, Daniel Gerow, Nathan Whitehouse, Robert Walls, and David Jensen for conversations that helped develop the ideas presented here. Thanks also to Hampshire College for support of the Hampshire College Institute for Computational Intelligence.

This material is based upon work supported by the National Science Foundation under Grant No. 1017817. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] L. Altenberg. *The evolution of evolvability in genetic programming*, pages 47–74. MIT Press, Cambridge, MA, USA, 1994.

[2] R. Crawford-Marks and L. Spector. Size control via size fair genetic operators in the pushgp genetic programming system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 733–739, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[3] T. Ito, H. Iba, and S. Sato. Depth-dependent crossover for genetic programming. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 775 –780, May 1998.

[4] T. Ito, H. Iba, and S. Sato. Non-destructive depth-dependent crossover for genetic programming. In *Proceedings of the First European Workshop on Genetic Programming*, pages 71–82, London, UK, 1998. Springer-Verlag.

[5] T. Ito, H. Iba, and S. Sato. *A self-tuning mechanism for depth-dependent crossover*, pages 377–399. MIT Press, Cambridge, MA, USA, 1999.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

[8] W. B. Langdon. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1:95–119, April 2000.

[9] S. Luke and L. Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, Fall 2006.

[10] S. Silva and E. Costa. 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.

[11] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *In 1998 IEEE International Conference on Evolutionary Computation*, pages 781–186. IEEE Press, 1998.

[12] M. Terrio and M. I. Heywood. Directing crossover for reduction of bloat in GP. In W. Kinsner, A. Seback, and K. Ferens, editors, *IEEE CCECE 2003: IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1111–1115. IEEE Press, 12-15 May 2002.