Lexicographic Parsimony Pressure

Sean Luke
George Mason University
http://www.cs.gmu.edu/∼sean/

Liviu Panait
George Mason University
http://www.cs.gmu.edu/∼lpanait/

Abstract

We introduce a technique called lexicographic parsimony pressure, for controlling the significant growth of genetic programming trees during the course of an evolutionary computation run. Lexicographic parsimony pressure modifies selection to prefer smaller trees only when fitnesses are equal (or equal in rank). This technique is simple to implement and is not affected by specific differences in fitness values, but only by their relative ranking. In two experiments we show that lexicographic parsimony pressure reduces tree size while maintaining good fitness values, particularly when coupled with Koza-style maximum tree depth limits.

1 INTRODUCTION

Like many arbitrary-sized representations in evolutionary computation, genetic programming (GP) individuals tend to grow significantly in size when no code growth counter-agents are applied. This growth is relatively independent of significant increases in fitness. The phenomenon, known in GP circles as *bloat*, is shaping up to be a major impediment to GP's scalability to more difficult problems which necessitate longer evolutionary runs.

The chief way bloat is controlled in GP is through the use of breeding restrictions stipulating the maximum depth of a GP parse tree individual. Lately other approaches have taken root, most popularly various forms of *parsimony pressure*, where the size of an individual is taken into consideration during selection. Parsimony pressure has to date taken two basic forms: parametric parsimony pressure, where an individual's size directly changes its fitness, and pareto parsimony pressure, where an individual's size is considered as an additional objective in a pareto optimization scheme.

In this paper we present a new family of parsimony pressure techniques which we think may be particularly apropos to GP and other evolutionary systems with large numbers of fitness-equivalent individuals in a population. This family is collectively known as *lexicographic parsimony pressure*, and is based on the idea of placing fitness, then size in lexicographic order; that is, preferring smaller individuals only when fitness is identical (or in some versions, similar). Lexicographic parsimony pressure is simple to implement, and it is less tied to the specific absolute fitness values in the population than parametric techniques are, much in the same way that tournament selection touted over fitness-proportionate selection.

We open the paper with discussions of current bloat-control techniques, followed by a description of lexicographic parsimony pressure and variations we have tried. We then give the results of an experiment showing that in most cases lexicographic parsimony pressure produces equivalent best-fitness-of-run results with significantly smaller trees than does depth restriction, except in the Symbolic Regression domain, where it performs poorly. We then give the results of a second experiment where we show that combinations of lexicographic parsimony pressure and depth limiting work very well compared to depth limiting alone.

2 CONTROLLING BLOAT

The evolutionary computation community has tried a number of approaches to controlling the growth of arbitrary-sized individuals. First and foremost are a number of parsimony pressure techniques, which include consideration of an individual's size as part of the selection procedure. Genetic programming has popularized some other techniques. Below we list four such techniques, followed by a range of parsimony pressure approaches.

Maximum Size or Depth Restriction This approach simply limits the maximum size of an individual, usually by rejecting large children as part of the breeding process.

For example, much work in GP follows the technique used in Koza [1992], which restricts modification operators to produce new trees of depth less than 17.

Explicitly Defined Introns This GP-specific technique allows the inclusion of special nodes which adapt the likelihood of subtree crossover or mutation at specific positions in the tree [Nordin et al. 1996].

Code Editing One easy way to attack growth is to directly simplify and optimize an individual's parse tree. Soule et al. [1996] for example report strong results with this approach. However, Haynes [1998] warns that editing can lead to premature convergence.

Pseudo-Hillclimbing This technique rejects children if they are not superior to (or simply different from) their parents in fitness. If a child is rejected from joining the next generation, a copy of its parent joins the next generation in its stead. One effect of this technique is to replicate large numbers of parents into future generations; earlier individuals are generally smaller than later individuals (hence the bloat), this results in slower growth in average size. This technique has been reported with some success in [Langdon and Poli 1998; Soule and Foster 1998b].

2.1 PARSIMONY PRESSURE

Unlike the techniques mentioned earlier, parsimony pressure is not GP-specific and has been used whenever arbitrary-sized representations tended to get out of hand. Such usage to date can be divided into two broad categories: parametric parsimony pressure, where size is a direct numerical factor in fitness, and pareto parsimony pressure, where size is considered as a separate objective in a pareto-optimization procedure.

Parametric Parsimony Pressure This includes size metrics, along with raw fitness, as part of an equation in computing the final fitness of an individual. For purposes of the discussion, let f be the individual's raw fitness, where higher is better, and g be the fitness after parsimony pressure is considered. Let s be an individual's size, and let a, b, c be arbitrary constants.

The most widely-used approach to parametric parsimony pressure is to treat the individual's size as a linear factor in fitness, that is, $g = af - bs$. This technique has been used in both GP [Koza 1992] and in non-GP [Burke et al. 1998]. Soule and Foster [1998a] present an interesting analysis of linear parsimony pressure and when and why it can fail. Linear parsimony pressure is occasionally augmented with a limit, that is if $s \leq c$ then $g = af$, else $g = af + b(c - s)$ [Cavaretta and Chellapilla 1999]. Belpaeme [1999] used a similar limit, but considered maximal tree depth rather than size as the parameter. Nordin and Banzhaf [1995] also applied parametric parsimony pressure, believed to be linear, to evolve machine language GP strings.

Linear parsimony pressure has been used in combination with adaptive strategies. Zhang and Mühlenbein [1995] adjusted b based on current population quality. Iba et al. [1994] propose a similar technique, except they use information-theoretic functions for f and s. Linear parsimony pressure has also been applied in stages: first by setting $g = f$, then factoring in size only after the population has reached a sufficient quality [Kalganova and Miller 1999]. Some non-GP papers [Wu et al. 1999; Bassett and De Jong 2000] use a nonlinear parsimony pressure: $g = (1 - as)f$. Bassett and De Jong note that this has the added feature of increasing the penalty proportionally to the fitness.

The problem with parametric parsimony pressure is exactly that: it is parametric, rather than based on rank. One must tune the parsimony pressure so as not to overwhelm the fitness metric. This can be difficult when the fitness assessment procedure is nonlinear, as is usually the case: it may well be that a difference between 0.9 and 0.91 in fitness is much more significant than a difference between 0.7 and 0.9. Parametric parsimony pressure can thus give size an unwanted advantage over fitness when the difference in fitness is only 0.01 as opposed to 0.2. Unexpected strength in the size parameter can also arise when the population's fitnesses are converging late in the evolution procedure. These issues are similar to those which gave rise to the preference of tournament selection and other nonparametric selection procedures over fitness-proportionate selection.

Pareto Parsimony Pressure The recent trend in parsimony pressure has been to treat it as a separate objective in a nonparametric, pareto optimization scheme. Pareto optimization is used when the evolutionary system must optimize for two or more objectives at once, but it is not clear which objective is "more important". An individual A is said to *pareto-dominate* another individual B if A is as good as B on all objectives, and better than B in at least one objective. One pareto optimization scheme assumes that one individual has a higher fitness than another if it dominates the other. Another scheme bases the fitness of individuals on the number of other individuals they dominate.

Pareto parsimony pressure treats raw fitness as one objective to optimize, and the individual's size as another objective. One particularly enticing feature of pareto parsimony pressure is that there is nothing to tune. Unfortunately, the technique has so far had mixed results in the literature. Some papers report smaller trees and the discovery of more ideal solutions [Bleuler et al. 2001; DeJong et al. 2001], but tellingly they omit best-fitness-of-run results.[1] Ekart and Nemeth [2001] report the mean best-fitness-of-run, but it is *worse* than when not using the technique.

3 LEXICOGRAPHIC PARSIMONY PRESSURE

Lexicographic parsimony pressure is a straightforward multiobjective technique for optimizing both fitness and tree size, by treating fitness as the primary objective and tree size as a secondary objective in a lexicographic ordering. The procedure does not assign a new fitness value, but instead uses a modified tournament selection operator to consider size.

To select an individual, two individuals are chosen at random, and their fitnesses compared. If an individual has superior fitness, it is selected. If the fitnesses are the same, then sizes are compared, and the smaller individual is selected. If both fitness and size are the same, an individual is selected at random.

We think the procedure is attractive because it is based on the relative rank of individuals in a population rather than their explicit fitness values: thus specific differences in fitness are immaterial. All that matters is that one fitness is greater than another. Additionally, plain lexicographic parsimony pressure has nothing to tune. However, the procedure only works well in environments which have a large number of individuals with identical fitness. As it so happens, genetic programming is just such an environment, thanks to a large amount of inviable code (regions where crossover has no effect) and other events causing neutral crossovers and mutations.

Of course, there exist problem domains where few individuals have the same fitness. For these domains we propose two possible modifications of lexicographic parsimony pressure, both based on the notion of sorting the population, putting it into ranked buckets, and treating each individual in the bucket as if it had the same fitness. These two modifications are:

Direct Bucketing The number of buckets, b, is specified beforehand, and each is assigned a rank from 1 to b. The population, of size p, is sorted by fitness. The bottom $\lceil p/b \rceil$ individuals are placed in the worst ranked bucket, plus any individuals remaining in the population with the same fitness as the best individual in the bucket. Then the second worst $\lceil p/b \rceil$ individuals are placed in the second worst ranked bucket, plus any individuals in the population equal in fitness to the best individual in that bucket. This continues until there are no individuals in the population. Note that the topmost bucket with individuals can hold fewer than $\lceil p/b \rceil$ individuals, if p is not a multiple of b. Depending on the number of equal-fitness individuals in the population, there can be some top buckets that are never filled. The fitness of each individual in a bucket is set to the rank of the bucket holding it. Direct bucketing has the effect of trading off fitness differences for size. Thus the larger the bucket, the stronger the emphasis on size as a secondary objective.

[1] As we argue in an accompanying paper, ideal-solution counts are a very poor measure of quality. Not only are they statistically invalid, but in fact are not correlated, or as badly as *inversely correlated*, with mean best-fitness-of-run results.

Ratio Bucketing Here the buckets are proportioned so that low-fitness individuals are placed into much larger buckets than high-fitness individuals. A bucket ratio $1/r$ is specified beforehand. The bottom $\lceil 1/r \rceil$ fraction of individuals of the population are placed into the bottom bucket. If any individuals remain in the population with the same fitness as the best individual in the bottom bucket, they too are placed in that bucket. Of the remaining population, the next $\lceil 1/r \rceil$ fraction of individuals are placed into the next bucket, plus any individuals remaining in the population with the same fitness as the best individual now in that bucket, and so on. This continues until every member of the population has been placed in a bucket. Once again, the fitness of every individual in a bucket is set to the rank of the bucket relative to other buckets. As the remaining population decreases, the $\lceil 1/r \rceil$ fraction also decreases: hence higher-ranked buckets generally hold fewer individuals than lower-ranked buckets. Ratio bucketing thus allows parsimony to have more of an effect on average when two similar low-fitness individuals are considered than when two high-fitness individuals are considered.

Both bucketing schemes fill the buckets with remaining individuals equal in fitness to the best individual in the bucket. The purpose of this is to guarantee that all individuals of the same fitness fall into the same bucket and thus have the same rank. This removes artifacts due to the particular ordering of the population. Bucketing schemes require that the user specify a bucket parameter (either the number of buckets or the bucket ratio). This parameter guides how strong an effect parsimony can have on the selection procedure. Note however that this parameter is not a direct factor in fitness. Thus the specific difference in fitness between two individuals is still immaterial; all that matters is fitness rank.

We are aware of two papers in the literature which have used variations on lexicographic parsimony pressure. Lucas [1994] used a linear parametric function to evolve bitstrings used in context-free grammars: but the size was multiplied by a constant small enough to guarantee that the largest possible advantage for small size was less than

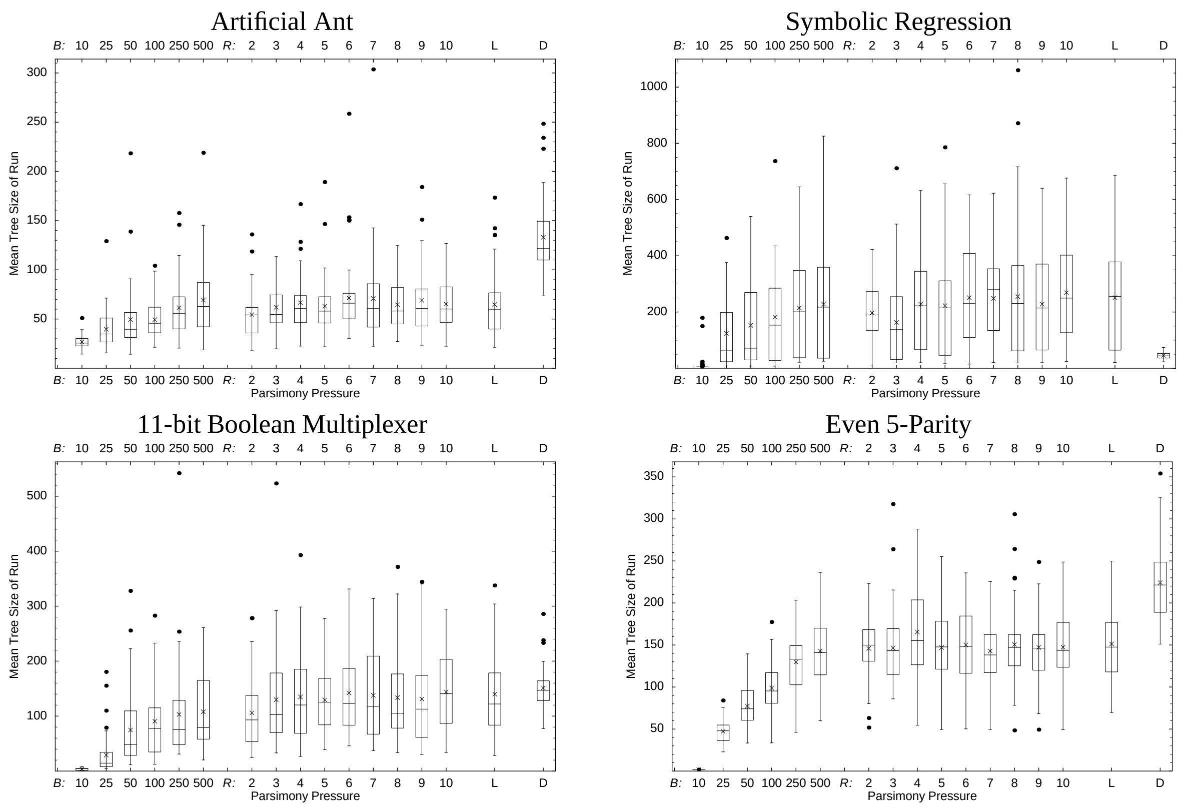

Figure 1: Boxplots of distributions of mean-tree-size-of-run for various parsimony pressure methods, as compared compared to plain depth limiting (labeled *D*). Lexicographic parsimony pressure is labeled *L*. Direct bucketing is labeled *B*, with the given number of buckets. Ratio bucketing is labeled *R* with the given ratio value.

the smallest difference in fitness. We believe the fitness was then fed into a fitness-proportional selection operator. In the *pygmies and civil servants* algorithm [Ryan 1994], crossover is always between one "civil servant" and one "pygmy". A pygmy is selected using linear parsimony pressure with a heavy weight for small size. A civil servant is selected using plain lexicographic selection. Both papers mention parsimony advantages only in passing.

4 EXPERIMENTS

Like most parsimony pressure literature, we chose to compare against the most popular technique for size restriction, namely Koza-style depth limiting. We performed two sets of experiments. The first experiment compared lexicographic parsimony pressure against depth limiting. The second experiment used lexicographic parsimony pressure in combination with depth limiting.

The experiments used population sizes of 1000. Without parsimony pressure, the depth ordered runs used plain tournament selection with a tournament size of 2. We chose four problem domains: Artificial Ant, 11-bit Boolean Multiplexer, Symbolic Regression, and Even-5 Parity. We followed the parameters specified in these four domains as set forth in Koza [1992]. Symbolic Regression used no ephemeral random constants. Artificial Ant used the Santa Fe food trail. Statistical significance was determined with ANOVAs at 95%. The evolutionary computation system used was ECJ 7 [Luke 2001].

As lexicographic ordering is influenced by likelihood of individuals having the same (or similar) fitness, it is useful to note the features of these four problem domains in this respect. Artificial Ant evolves trees to control an ant to eat as many food pellets as possible within 400 time steps. Fitness is simply the number of pellets, and the trail has only 89 of them, so there are relatively few fitness values an individual may take on. The 11-bit Boolean Multiplexer and Even-5 Parity problems both require the individual to learn a complex boolean function. 11-bit Boolean Multiplexer has integer fitness values ranging from 0 to 2048. It is known that 11-bit Boolean Multiplexer has relatively little inviable code, but most individuals' fitnesses fall into multiples of 32 or 64. Even-5 Parity has the fewest number of fitness values: only integer fitness values ranging

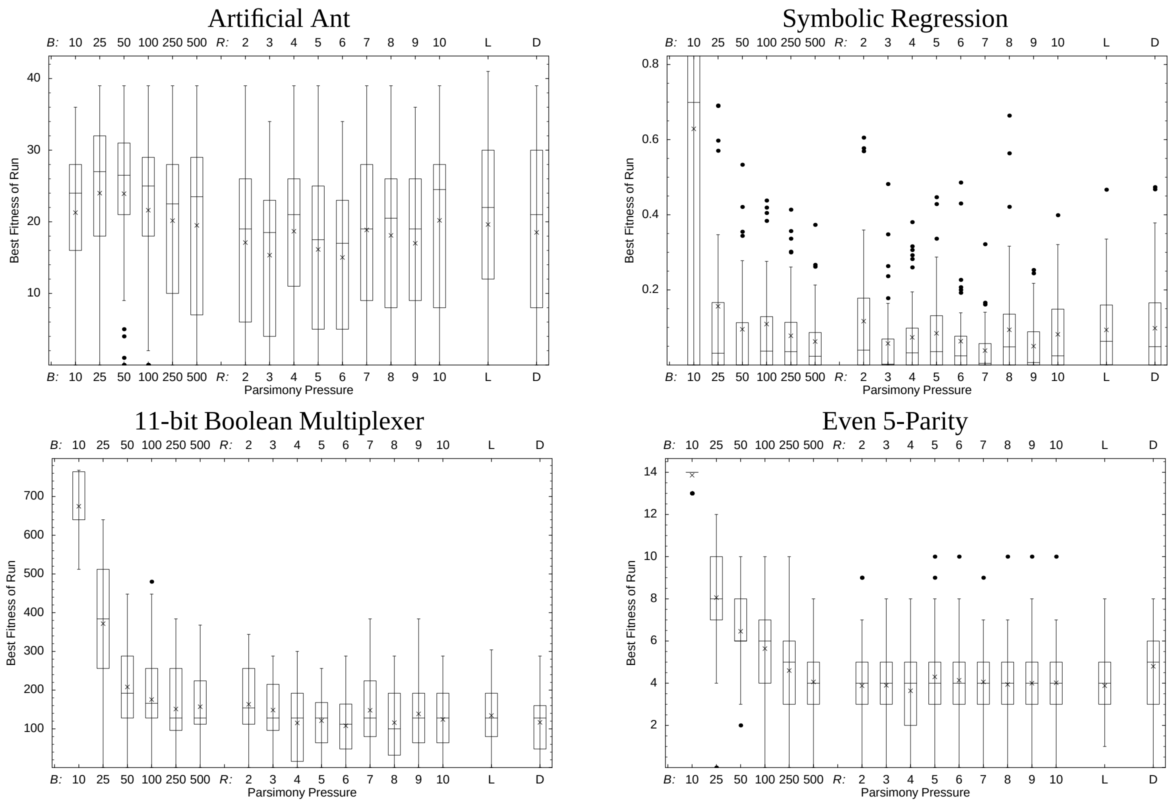

Figure 2: Boxplots of distributions of best-fitness-of-run for various parsimony pressure methods, as compared compared to plain depth limiting (labeled *D*). Lexicographic parsimony pressure is labeled *L*. Direct bucketing is labeled *B*, with the given number of buckets. Ratio bucketing is labeled *R* with the given ratio value. Lower fitness is better.

from 0 to 33. Symbolic Regression asks trees to fit a real-valued function within the domain [-1,1] but with any valid range; thus individuals can take on any real-valued fitness. However, Symbolic Regression suffers from a very large amount of inviable code, so many individuals in the population have the same fitness.

4.1 FIRST EXPERIMENT

The first experiment compared depth limiting against pure parsimony pressure approaches. Specifically, the techniques compared are:

- Lexicographic parsimony pressure with direct bucketing, using 10, 25, 50, 100, 250, or 500 buckets.
- Lexicographic parsimony pressure with ratio bucketing, using bucket ratios of 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, or 1/10.
- Plain lexicographic parsimony pressure.
- Depth limiting (to 17).

We did 50 runs per technique, and plotted boxplots[2] showing the distribution of the best fitness per run, and also of the average tree size per run. Runs continued for 51 generations, and did not stop on the discovery of the optimum. Results are shown in Figures 1 and 2.

In the Artificial Ant and Even 5-Parity problems, all parsimony pressure techniques yielded statistically significantly superior tree size results to depth limiting, and had statistically insignificant differences in fitness, except for direct bucket numbers of 10, 25, and 50 for Even 5-Parity, which had worse fitness values. Small-numbered direct bucketing yielded much better tree sizes. For Even-5 Parity, this came at the cost of much worse fitness values. Artificial Ant, there was no difference in fitness.

For 11-bit Boolean Multiplexer, all parsimony pressure techniques had smaller mean tree sizes than depth limiting, but only direct bucketing numbers of 10, 25, 50, and 100 had statistically significant differences. Similarly, all

[2]In a boxplot, the rectangular region covers all values between the first and third quartiles, the stems mark the furthest individual within 1.5 of the quartile ranges, and the center horizontal line indicates the median. Dots show outliers, and $\times$ marks the mean.

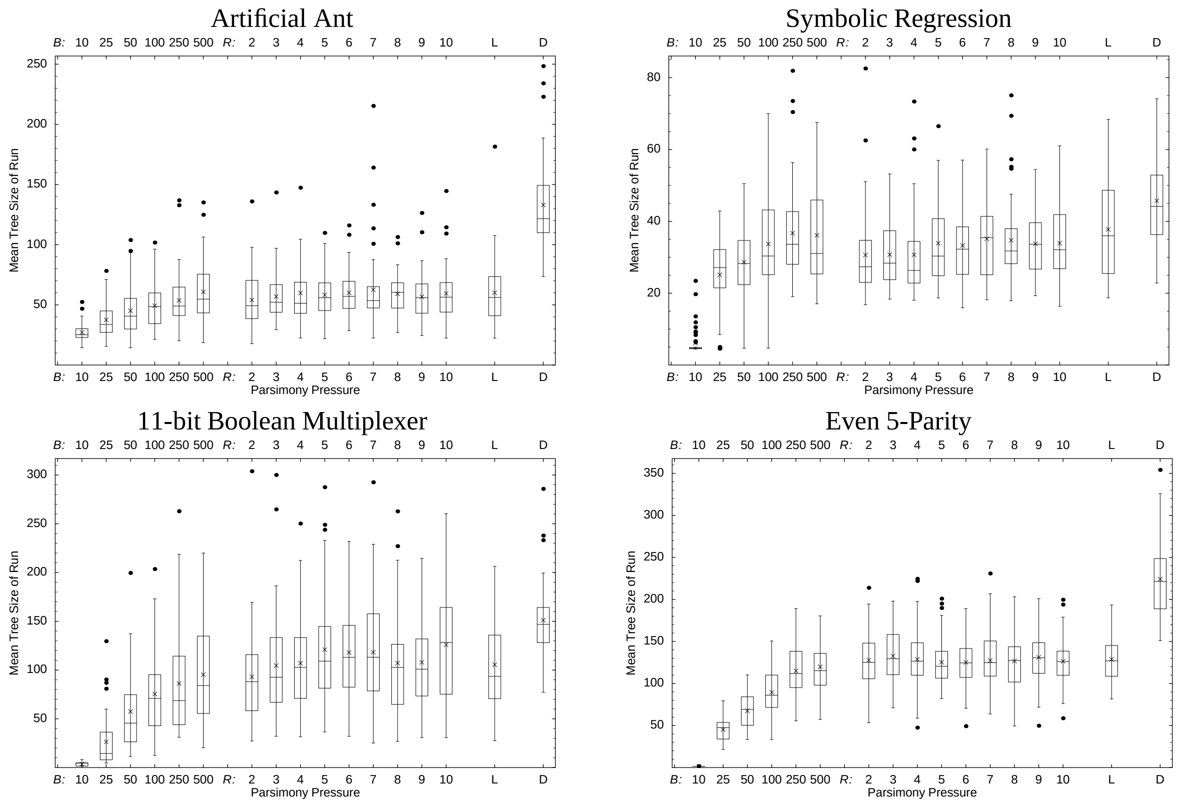

Figure 3: Boxplots of distributions of mean-tree-size-of-run for various parsimony pressure methods in combination with depth limiting, as compared compared to plain depth limiting (labeled *D*). Lexicographic parsimony pressure is labeled *L*. Direct bucketing is labeled *B*, with the given number of buckets. Ratio bucketing is labeled *R* with the given ratio value.

techniques had statistically insignificant differences in fitness except for direct bucking numbers of 10, 25, and 50, which had worse fitness.

The surprise came with Symbolic Regression. We had expected lexicographic parsimony pressure to yield poor tree sizes in this domain relative to depth limiting, and it did. But interestingly, bucketing also had poor tree sizes. Only direct bucking with 10 buckets yielded statistically significantly worse fitness than depth limiting.

Growth Curves For the Even-5 Parity problem, parsimony pressure techniques generally held tree growth to a standstill or began lowering tree sizes it by generation 30. For Artificial Ant, this occurred by about generation 10. In 11-bit Boolean Multiplexer, generation 40; most parsimony pressure techniques were lowering tree sizes by then as well. In Symbolic Regression, tree growth for all the parsimony pressure techniques rose in a quadratic curve similar to that found for unrestricted GP in this problem. With depth limiting in all four problem domains, mean tree growth continued to rise linearly.

Lexicographic parsimony pressure has an Achilles' heel: if GP can create incrementally better trees by tacking subtrees onto their periphery, then lexicographic parsimony pressure cannot act against it. As long as the trees are infinitesimally better, size does not come into play. Symbolic Regression has this property, and we had expected plain lexicographic parsimony pressure to do badly in this domain. But we were very surprised to see the poor performance of bucketing approaches as well.

4.2 SECOND EXPERIMENT

If depth limiting did well compared to lexicographic parsimony pressure in Symbolic Regression, and held its own reasonably in 11-bit Boolean Multiplexer, we wondered how the combination of the two techniques would fare. Our second experiment compared the same techniques as in the first experiment, but combined the parsimony pressure techniques with depth limiting. Again, we did 50 runs per technique. These results are shown in Figures 3 and 4.

This time, parsimony pressure plus depth limiting significantly outperformed depth limiting alone. In the Symbolic Regression, Artificial Ant, and Even-5 Parity problems, all

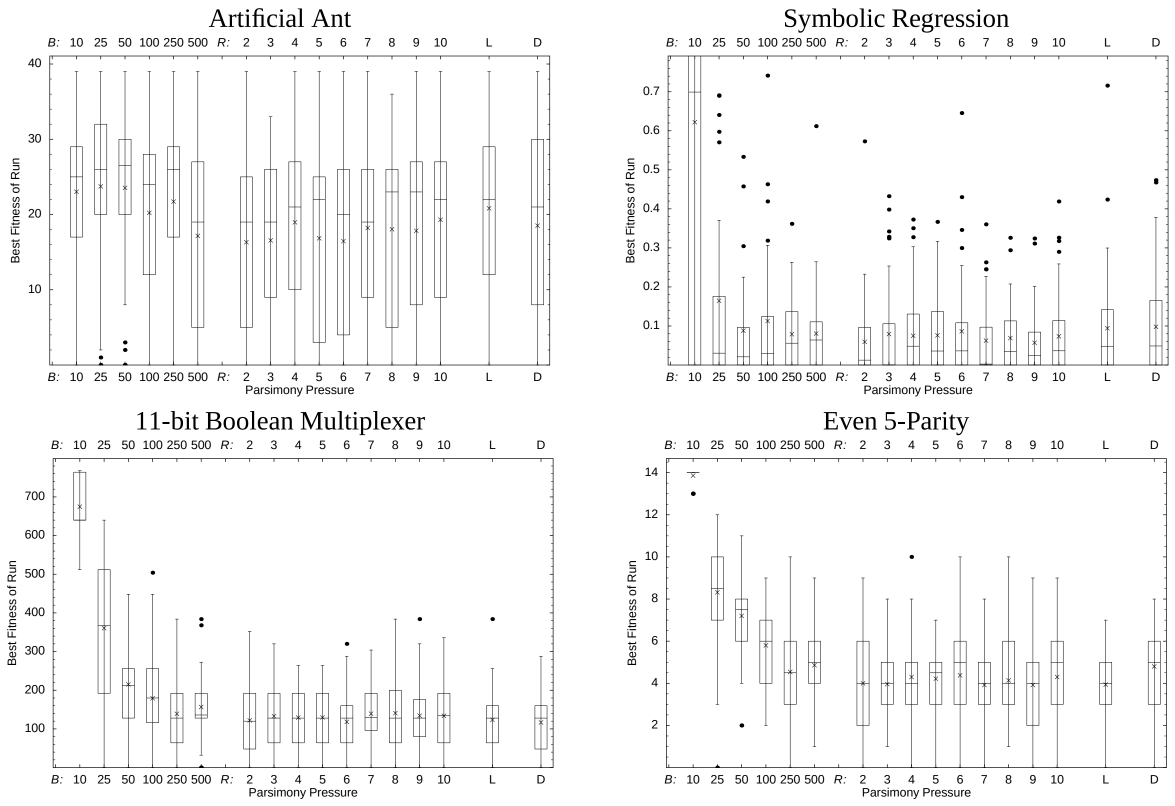

Figure 4: Boxplots of distributions of best-fitness-of-run for various parsimony pressure methods in combination with depth limiting, as compared compared to plain depth limiting (labeled *D*). Lexicographic parsimony pressure is labeled *L*. Direct bucketing is labeled *B*, with the given number of buckets. Ratio bucketing is labeled *R* with the given ratio value. Lower fitness is better.

applications of parsimony pressure plus depth limiting had statistically significantly superior tree sizes when compared to plain depth limiting. In the 11-bit Boolean Multiplexer, this was also the case except for ratio buckets of size 1/5, 1/7, and 1/10, which had statistically insignificant differences with depth limiting.

As before, there were no statistically significant differences in fitness in the Artificial Ant problem. Direct bucketing with 10, 25, and 50 buckets yielded statistically significantly worse fitness than depth limiting for the Even-5 Parity and 11-bit Boolean Multiplexer problems. In the Symbolic Regression problem, only direct bucketing with 10 buckets had statistically worse fitness than depth limiting.

Growth Curves In the Symbolic Regression and Even-5 Parity problems, parsimony pressure plus depth limiting flattened out tree growth by about generation 25. In the Artificial Ant problem, parsimony pressure plus depth limiting dropped sizes after about generation 5, flattening out at about generation 20. In the 11-bit Boolean Multiplexer, the same techniques began slowly lowering tree sizes at about generation 35.

5 CONCLUSIONS AND FUTURE WORK

In three of four problem domains, lexicographic parsimony pressure and its variants (direct bucketing and ratio bucketing, given reasonable parameter values) maintained the same mean best-fitness-of-run as did Koza-style depth limiting, with equivalent or significantly lower mean tree sizes. But in Symbolic Regression, where incrementally larger trees are often (just barely) superior in fitness, lexicographic techniques were practically helpless to stop bloat. However, a combination of depth limiting and lexicographic parsimony pressure consistently outperformed depth limiting in capping bloat, while maintaining statistically equivalent mean best-fitness-of-run values. Given its simple implementation and general applicability, we hope lexicographic parsimony pressure may prove a popular approach to bloat control. We plan to extend this work to other techniques such as layered tournaments which alternately consider fitness and size. We also plan to compare directly to parametric parsimony pressure and pareto-optimization-based methods in the future.

References

Bassett, J. K. and De Jong, K. A. (2000). Evolving behaviors for cooperating agents. In *International Syposium on Methodologies for Intelligent Systems*, pages 157–165.

Belpaeme, T. (1999). Evolution of visual feature detectors. In Poli, R., Cagnoni, S., Voigt, H.-M., Fogarty, T., and Nordin, P., editors, *Late Breaking Papers at EvoISAP'99: the First European Workshop on Evolutionary Computation in Image Analysis and Signal Processing*, pages 1–10, Goteborg, Sweden.

Bleuler, S., Brack, M., Thiele, L., and Zitzler, E. (2001). Multiobjective genetic programming: Reducing bloat using spea2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.

Burke, D. S., De Jong, K. A., Grefenstette, J. J., Ramsey, C. L., and Wu, A. S. (1998). Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6(4):387–410.

Cavaretta, M. J. and Chellapilla, K. (1999). Data mining using genetic programming: The implications of parsimony on generalization error. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1330–1337, Mayflower Hotel, Washington D.C., USA. IEEE Press.

DeJong, E. D., Watson, R. A., and Pollack, J. B. (2001). Reducing bloat and promoting diversity using multi-objective methods. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, California, USA. Morgan Kaufmann.

Ekart, A. and Nemeth, S. Z. (2001). Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73.

Haynes, T. (1998). Collective adaptation: The exchange of coding segments. *Evolutionary Computation*, 6(4):311–338.

Iba, H., de Garis, H., and Sato, T. (1994). Genetic programming using a minimum description length principle. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 12, pages 265–284. MIT Press.

Kalganova, T. and Miller, J. (1999). Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In Stoica, A., Keymeulen, D., and Lohn, J., editors, *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware (EH'99)*, pages 54–63, Piscataway, NJ. IEEE.

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

Langdon, W. B. and Poli, R. (1998). Genetic programming bloat with dynamic fitness. In Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C., editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 96–112, Paris. Springer-Verlag.

Lucas, S. (1994). Structuring chromosomes for context-free grammar evolution. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 130–135. IEEE.

Luke, S. (2001). ECJ 7: An EC and GP system in Java. http://www.cs.umd.edu/projects/plus/ec/ecj/.

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

Nordin, P., Francone, F., and Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In Angeline, P. J. and Kinnear, Jr., K. E., editors, *Advances in Genetic Programming 2*, pages 111–134. MIT Press, Cambridge, MA, USA.

Ryan, C. (1994). Pygmies and civil servants. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 11, pages 243–263. MIT Press.

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

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

Soule, T., Foster, J. A., and Dickinson, J. (1996). Code growth in genetic programming. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, Stanford University, CA, USA. MIT Press.

Wu, A., Schultz, A., and Agah, A. (1999). Evolving control for distributed micro air vehicles. In *IEEE Computational Intelligence in Robotics and Automation Engineers Conference*.

Zhang, B.-T. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38.