Resource-Limited Genetic Programming: Replacing Tree Depth Limits

Sara Silva[1], Pedro J.N. Silva[2], Ernesto Costa[1]
[1]Centro de Informática e Sistemas da Univ. Coimbra, Dep. Engenharia Informática
Polo II – Pinhal de Marrocos, 3030 Coimbra, Portugal
[2]Centro de Genética e Biologia Molecular, Dep. Biologia Vegetal
Fac. Ciências Univ. Lisboa, Campo Grande, 1749-016 Lisboa, Portugal
E-mail: {sara,ernesto}@dei.uc.pt, pedro.silva@fc.ul.pt

Abstract

We propose replacing the traditional tree depth limit in Genetic Programming by a single limit on the amount of resources available to the whole population, where resources are the tree nodes. The resource-limited technique removes the disadvantages of using depth limits at the individual level, while introducing automatic population resizing, a natural side-effect of using an approach at the population level. The results show that the replacement of individual depth limits by a population resource limit can be done without impairing performance, thus validating this first and important step towards a new approach to improving the efficiency of GP.

1 Introduction

Genetic Programming (GP) solves complex problems by evolving populations of computer programs, using Darwinian evolution and Mendelian genetics as inspiration. Bloat is an excess of code growth caused by the genetic operators in search of better solutions, without a corresponding improvement in fitness. It is a serious problem in GP, often leading to the stagnation of the evolutionary process [1].

The traditional approach to maintaining code growth under control is by imposing a tree depth limit on the individuals accepted into the population, on a tree-based GP system [2]. Several other techniques have been used with various degrees of success (reviews and recent work in refs. 3–6), but none was ever as popular as the traditional depth limits.

This paper describes how to implement a simple technique to replace the traditional tree depth limits. It is based on a single limit imposed on the amount of resources available to the whole GP population, where resources are the tree nodes or other elements in non tree-based GP (*e.g.* code lines). The resource-limited technique removes most of the disadvantages of using depth limits at the individual level, while introducing automatic population resizing, a natural side-effect of using an approach at the population level.

Previous work, focused on financial time series prediction, also used limits on the total number of nodes in the population [7]. The results presented were, however, scarce, and the implications of the idea have not been explored any further.

Section 2 of this paper deals with several aspects of tree depth limits, while Sect. 3 introduces and explains the limited-resources technique. Section 4 describes the experiments made, Sect. 5 relates the results obtained, and finally Sect. 6 draws some conclusions and points towards future directions of this work.

2 Tree Depth Limits

Tree-based GP traditionally uses a depth limit to avoid excessive growth of its individuals [2]. When crossover creates an offspring that violates this limit, one of its parents is chosen for the new generation instead.

Traditional depth limits effectively avoid the growth of trees beyond a certain point, but they do nothing to control bloat until the limit is reached. They may also prevent the optimal solution to be found for problems of unsuspected high complexity. This may happen either because the number of possible nodes in a tree of maximum depth is not enough to represent the solution, or because a maximum depth may prove too hard a restriction to find the solution, regardless of the number of necessary nodes (for example, in symbolic regression of the quartic polynomial, $x^4 + x^3 + x^2 + x$, tree-based GP usually finds a solution of depth 7, whereas solutions of depth 6, 5, and even 4[1] are rarely found). Last but not least, depth limits cannot be used on non tree-based GP systems.

Various approaches have been tried in order to overcome the difficulties mentioned above. Some rely on

[1]The factored form of the polynomial, $(x^2 + 1)(x^2 + x)$, can be represented with a tree of depth 4.

choosing specialized genetic operators to keep tree growth under control, without imposing strict limits [8, 9].

Recent work on *dynamic* limits has achieved promising results without the need for specific operators [10, 11]. It introduces a dynamic tree depth limit, initially set at a low value, but increased whenever needed to accommodate an offspring that, although deeper than the limit, is better than any other individual found during the run [10], and optionally decreased again whenever possible [11]. The idea has even been extended to create a dynamic *size* limit, where size is the number of tree nodes regardless of depth (which would enable its use in non tree-based GP), but this variation did not perform so well as the original idea [11]. All in all, new and better approaches to bloat control are still needed.

3 Limited Natural Resources in GP

Individuals in GP are built with small elements called functions and terminals. Each individual is made from a certain number of these elements, jointly designated as nodes. We can refer to the number of nodes of an individual as the amount of resources the individual uses (which is also directly related to the computational effort needed to evaluate it).

We propose limiting the total amount of resources the *population* can use in each generation. We can think of it as limiting the amount of natural resources available to a given biological population, where each individual competes with the others for its share, and the weakest individuals perish when resources are scarce. In GP, resources become scarce when the total number of nodes in the population exceeds the predefined limit. Beyond this point, not all offspring are guaranteed to be accepted into the new generation. The allocation of resources to individuals (ensuring their survival) is mainly based on fitness, with size playing a secondary role.

All the candidates to the new generation are queued by fitness, regardless of their size, and given the resources they need in a first come, first served basis. The individuals requiring more resources than the amount still available are skipped (do not survive) and the allocation continues until the end of the queue. Some resources may remain unused. A rule emerges from this procedure, promoting the survival of the best individuals and the rejection of 'not good enough for their size' individuals, where the relationship between size and fitness is not explicitly programmed, but a product of the evolutionary process itself.

The resource-limited approach is expected to cause a steady decrease of the population size (defined as the number of individuals) as long as code growth continues. According to recent work [12–15], this factor may bring some benefits in terms of convergence to good solutions. It is also expected that, after the resources have reached the exhaustion point, eventually some new generation will use them more sparingly, having all its individuals surviving and still leaving some resources available. We have considered two options on how to deal with this situation: (1) use the exceeding resources to allow the survival of the best individuals of the previous generation - the parents that would otherwise be replaced by their offspring - by applying the same allocation procedure described above (but ensuring that the population size never increases beyond its initial value), or (2) do not use the exceeding resources, thus never allowing the population size to increase again.

4 Experiments

The aim of these experiments is to see whether traditional tree depth limits can be replaced by resource limits as described above, without impairing performance in terms of fitness and computational effort. To perform a fair comparison, the resource limit for the new technique must be such that, during the entire run, the total amount of resources used is roughly the *same* for both approaches. We also want to study the differences in the evolution of population size and mean tree size inside the population.

A simple problem was used for the experiments: symbolic regression of the quartic polynomial ($x^4 + x^3 + x^2 + x$, with 21 equidistant points in the interval -1 to $+1$). An initial population of 500 individuals (Ramped Half-and-Half initialization [2] with maximum depth 6) was evolved for 50 generations, even if the optimal solution was found earlier. Tree crossover was the only genetic operator used, and reproduction rate was set at 0.1. The function and terminal sets were $\{+, -, \times, \div, sin, cos, log, exp\}$ (protected as in ref. 2) and $\{x\}$, respectively. Selection for reproduction used the lexicographic parsimony pressure tournament [5] and selection for survival used no elitism (in the traditional sense only, since the resource-limited approach can be considered highly elitist). A total of 50 runs were performed with each of the following techniques:

None	$\rightarrow$ no limits
Depth	$\rightarrow$ tree depth limit
R$_{\text{steady}}$	$\rightarrow$ limited resources, forced steady usage
R$_{\text{low}}$	$\rightarrow$ limited resources, possible low usage

The first technique (None) uses in fact no technique: nothing is done to control the growth of trees. The second technique (Depth) uses the traditional tree depth limit as described in Sect. 2, with the typical value 17. The following two techniques, R$_{\text{steady}}$ and R$_{\text{low}}$, use limited resources as described in Sect. 3, set at 14500. The

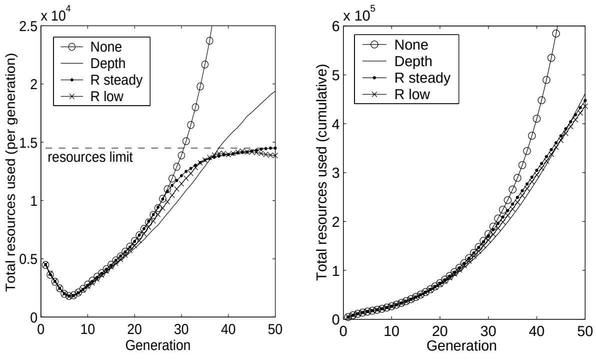

Fig. 1. Evolution of the amount of resources used by the population (left: per generation, right: cumulative).

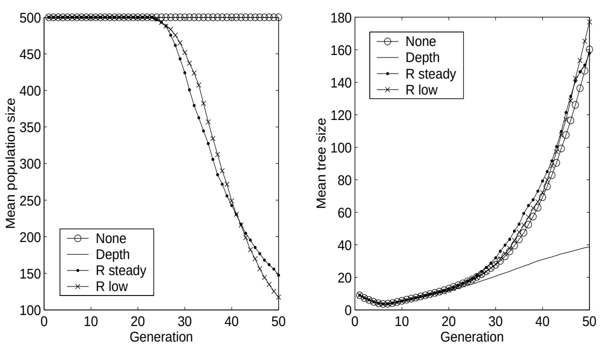

Fig. 2. Evolution of the population size (left plot) and mean tree size inside the population (right plot).

difference between them is that Rsteady forces the usage of all available resources (whenever possible), while Rlow leaves any exceeding resources unused (see Sect. 3 for details).

When looking for the ideal resources limit, 14500, we searched (in multiples of 500) for such a value that, when used to cap the amount of resources used per generation with the None technique, would result in an amount of *cumulative* resources (used during the entire run) similar to the cumulative amount used with the Depth technique. Capping at a lower value would have saved resources, but so would lowering the tree depth limit, and one must not forget we want to provide the same resources to both techniques.

All the experiments were performed using the GPLAB toolbox [16]. Statistical significance of the null hypothesis of no difference was determined with Kruskal-Wallis non-parametric ANOVAs at $p = 0.01$.

5 Results

The following results are based on the mean values over the 50 runs performed for each experiment.

Figure 1 shows the mean resources usage (per generation and cumulative) obtained with all techniques. Although the resources limit was chosen so that Depth and Rsteady would produce similar behavior (see Sect. 4), one could expect the Rsteady line to run close to None until it reached the resources limit (left plot, dashed line) and then stick to the limit until the end of the run. Instead, it diverts from None much sooner and barely reaches the limit on the last few generations, a result of the dynamics of the resource allocation procedure described in Sect. 3. The Rlow technique does not behave much differently from Rsteady. Consequently, the cumulative lines of Depth and both resource-limited techniques (Rsteady, Rlow) show very similar evolution (right plot), with no significant differences between them.

However similar the behavior of Depth, Rsteady and Rlow may look in Fig. 1 (except during the last 10 or 15 generations in the left plot), Fig. 2 shows a different reality, both in terms of population size and mean tree size inside the population (note that the left plot of Fig. 1 is obtained by “multiplying” both plots of Fig. 2). Regarding mean tree size, both Rsteady and Rlow run close to None during the entire run, while Depth diverts completely around generation 25 (right plot). In terms of population size, in both resource-limited techniques it starts dropping steeply (and not coincidentally) also by generation 25 (left plot).

Regardless of the relationship between population size and mean tree size, we are interested in looking at a different and important issue, fitness, and how it relates to the computational effort spent to obtain it. Effort can be roughly expressed as the total number of nodes evaluated – in other words, the total amount of resources used. Figure 3 shows best (lowest) fitness as a function of computational effort. Apart from the fact that the None technique proves to be a terrible waste of resources (which comes as no surprise), the remaining three techniques show similar behavior (note the logarithmic scale). The differences in the best fitness achieved during the run are not statistically significant between any two techniques.

6 Conclusions and Future Work

The replacement of the traditional tree depth limit in GP by a technique based on limited resources available to the entire population can be done without impairing performance, and at the same time removing most of the disadvantages of using depth limits.

The resource-limited technique could not be expected to perform better than tree depth limits in a simple problem like the one we have presented, where the resources available for both techniques were the same, and clearly more than needed to easily find the optimal solution. The

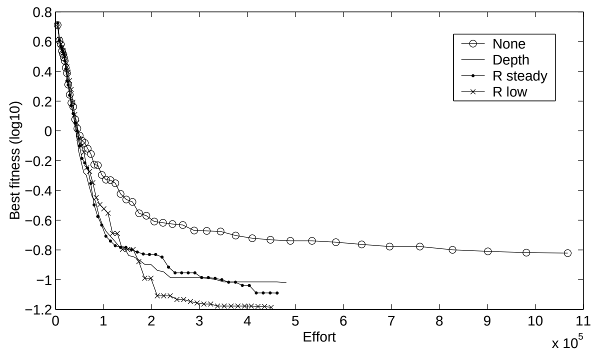

Fig. 3. Best fitness as a function of computational effort.

superiority of the new approach lies in its ability to automatically compensate higher tree size with lower population size, thus providing the necessary elements to cope with highly complex problems where tree depth techniques may fail, not for lack of resources, but for lack of flexibility. Unlike tree depth techniques, the resource-limited approach is also easily adaptable to non tree-based GP systems.

Still, the resource-limited technique may be criticized because, like the traditional depth limit, it does not prevent bloat from occurring freely until the limit is reached. The solution we propose for this problem is to adapt the idea of dynamic tree depth limits [10, 11] to the resource-limited approach: initially low, the limit is only raised if that results in improvement of the population fitness. This will be the next of a series of steps towards the achievement of more efficient GP systems.

Acknowledgements

We acknowledge grant SFRH/BD/14167/2003 from Fundação para a Ciência e a Tecnologia, Portugal. We also thank the Biomathematics Group at ITQB/UNL for allowing us the use of their computational resources.

References

[1] Banzhaf, W., Nordin, P., Keller, R. E., Francone, F. D. (1998). Genetic Programming - An Introduction. Morgan Kaufmann, San Francisco, CA.

[2] Koza, J. R. (1992). Genetic programming - on the programming of computers by means of natural selection. The MIT Press, Cambridge, MA.

[3] Soule, T., Foster, J. A. (1999). Effects of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation 6(4):293–309

[4] Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In Ryan, C. *et al.* (eds.), Proceedings of EuroGP-2003. Springer, Berlin, pp. 204–217

[5] Luke, S., Panait, L. (2002). Lexicographic parsimony pressure. In Langdon, W. B. *et al.* (eds.), Proceedings of GECCO-2002. Morgan Kaufmann, San Francisco, CA, pp. 829–836

[6] Panait, L., Luke, S. (2004). Alternative bloat control methods. In Deb, K. *et al.* (eds.), Proceedings of GECCO-2004. Springer, Berlin, pp. 630–641

[7] Wagner, N., Michalewicz, Z. (2001). Genetic programming with efficient population control for financial time series prediction. In Goodman, E. D. (ed.), GECCO-2001 LBP, pp. 458–462

[8] Kennedy, C. J., Giraud-Carrier, C. (1999). A Depth Controlling Strategy for Strongly Typed Evolutionary Programming. In Banzhaf, W. *et al.* (eds.), Proceedings of GECCO-1999. Morgan Kaufman, San Francisco, CA, pp. 1–6

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

[10] Silva, S., Almeida, J. S. (2003). Dynamic maximum tree depth. In Cantú-Paz, E. *et al.* (eds.), Proceedings of GECCO-2003. Springer, Berlin, pp. 1776–1787

[11] Silva, S., Costa, E. (2004). Dynamic limits for bloat control. In Deb, K. *et al.* (eds.), Proceedings of GECCO-2004. Springer, Berlin, pp. 666–677

[12] Luke, S., Balan, G. C., Panait, L. (2003). Population implosion in genetic programming. In Cantú-Paz, E. *et al.* (eds.), Proceedings of GECCO-2003. Springer, Berlin, pp. 1729–1739

[13] Fernandez, F., Vanneschi, L., Tomassini, M. (2003). The effect of plagues in genetic programming: A study of variable-size populations. In Ryan, C. *et al.* (eds.), Proceedings of EuroGP-2003. Springer, Berlin, pp. 317–326

[14] Fernandez, F., Tomassini, M., Vanneschi, L. (2003). Saving computational effort in genetic programming by means of plagues. In Sarker, R. *et al.* (eds.), Proceedings of CEC-2003. IEEE Press, Piscataway, NJ, pp. 2042–2049

[15] Tomassini, M., Vanneschi, L., Cuendet, J., Fernandez, F. (2004). A new technique for dynamic size populations in genetic programming. In Proceedings of CEC-2004. IEEE Press, Piscataway, NJ, pp. 486–493

[16] Silva, S. (2004). GPLAB - a genetic programming toolbox for MATLAB. http://gplab.sourceforge.net