Comparing Tree Depth Limits and Resource-Limited GP

Sara Silva, Ernesto Costa
Evolutionary and Complex Systems Group
Centre for Informatics and Systems of the University of Coimbra
Polo II - Pinhal de Marrocos, 3030 Coimbra, Portugal
{sara,ernesto}@dei.uc.pt
http://cisuc.dei.uc.pt/ecos/

Abstract- In this paper we compare two different approaches for controlling bloat in Genetic Programming, tree depth limits and resource-limited GP. Tree depth limits operate at the individual level, avoiding excessive code growth by imposing a maximum depth to each individual. Resource-limited GP is a new technique that operates at the population level, limiting the total amount of resources the entire population can use. We compare their dynamics and performance on three problems: Symbolic Regression, Even Parity, and Artificial Ant. The results suggest that resource-limited GP is superior to tree depth limits, but we question this superiority and discuss possible ways of combining the strengths of both approaches, to further improve the results.

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 common and serious problem in GP, often leading to the stagnation of the evolutionary process [1]. Several explanations for bloat have been proposed [2–4], and many different bloat control techniques have been tried with various degrees of success, most of them based on parsimony pressure [5–9], some relying on specialized genetic operators [10, 11]. But no other method was ever as popular as the usage of a static tree depth limit imposed on the individuals accepted into the population, on a tree-based GP system [12].

Recent work on dynamic limits [13,14], a variation of the traditional depth limits, has addressed some of the disadvantages inherent to the original method and achieved promising results in two simple test problems, but until now its performance on problems of higher complexity was unknown. A different approach to bloat control, based on limiting the total amount of tree nodes of the entire population, instead of imposing limits at the individual level [15] has been explored very recently [16, 17], introducing and successfully testing the concept of resource-limited GP, in two problems of different complexity. These two approaches, although aiming at solving the same problem and using the same rationale, operate at different levels of the GP paradigm: one acts at the individual level, the other at the population level. Both achieve promising results in controlling bloat without impairing performance but, because they aim at different targets, they produce radically different dynamics of the evolutionary process. The present work provides a comparative study of the performance of both approaches in terms of fitness reached and effort expended to achieve it, and also a comparison of their evolutionary behavior in terms of code growth and population size.

Sections 2 and 3 of this paper describe the two bloat control approaches, tree depth limits and resource-limited GP, with some detail. Section 4 explains the experiments performed on three different problems, Symbolic Regression, Even-Parity, and Artificial Ant. Section 5 reports the results obtained, while Section 6 discusses these results and raises some methodological issues, pointing towards future developments of this work. Section 7 concludes.

2 Tree Depth Limits

Tree-based GP traditionally uses a depth limit to avoid excessive growth of its individuals [12]. When a genetic operator creates an offspring that violates this limit, one of its parents is chosen for the new generation instead. This technique effectively avoids the growth of trees beyond a certain point, but it does nothing to control bloat until the limit is reached. The static nature of the depth limit may also prevent the optimal solution to be found for problems of unsuspected high complexity. Also, depth limits cannot be used on non tree-based GP systems.

Dynamic Maximum Tree Depth [13] is a recent bloat control technique inspired in the traditional tree depth limit. It also imposes a depth limit on the individuals accepted into the population, but this one is dynamic, meaning it can be changed during the run. The dynamic limit is initially set with a low value, but at least as high as the maximum depth of the initial random trees. Any new individual who breaks this limit is rejected and replaced by one of its parents instead (as with the original static limit), unless it is the best individual found so far. In this case, the dynamic limit is raised to match the depth of the new best-of-run and allow it into the population. The result is a succession of limit risings, as the best solution becomes more accurate and more complex.

In the original Dynamic Maximum Tree Depth, the dynamic limit was never decreased. It was later modified to implement a heavy dynamic limit that is lowered once the new best-of-run individual allows it [14]. Another contemporaneous variation was the introduction of a size limit, where size is the number of nodes regardless of depth, something that would allow the technique to be used in non tree-based GP systems.

The dynamic limits were tested in two simple problems,
Symbolic Regression of the quartic polynomial and Even-3 Parity, where they proved to effectively control bloat maintaining the ability to find good solutions [13]. Two initial values for the dynamic depth limit were tested, 6 (the depth of the initial random trees) and 9, with the lower value achieving the same results using much smaller (in terms of mean tree size) populations. Although the heavy limit was able to achieve even better results under the same conditions, the dynamic size did not perform so well in one of the problems [14].

3 Resource-Limited GP

The idea of controlling bloat by using limits on the total number of nodes in the entire population, instead of imposing limits at the individual level, was introduced by [15]. Very recently, it has been further developed, introducing the concept of resource-limited GP [16]. 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 resource-limited 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. The candidates to the new generation are queued by fitness, regardless of their size, and are given the resources they need (their number of nodes) in a first come, first served basis. The parents are queued after the offspring, also sorted by fitness - resources allowing it, they may also be accepted into the new generation. 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, or until restrictions on their number of individuals apply. Some resources may remain unused. Some parents may survive while their offspring perish. 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.

The resource-limited approach removes most of the disadvantages of using depth limits at the individual level, in particular, providing the necessary applicability to non-tree based GP, while introducing automatic population resizing, a natural side-effect of using an approach at the population level. After the resource limit is reached, and as long as code growth continues, the population size (defined as the number of individuals) steadily decreases, something that may actually improve convergence to good solutions [18–21]. The technique was initially tested on a simple problem, the Symbolic Regression of the quartic polynomial, where it proved to be able to replace the traditional tree depth limit without impairing performance [16].

From the original static resource limit, and drawing inspiration from the work on dynamic limits (see previous section), a dynamic approach to resource-limited GP was soon developed [17]. A dynamic resource limit is implemented, one that is initially set with a low value (at least as high as the amount of resources used by the first generation), and raised whenever it results in better mean population fitness. After generating the offspring, the candidates to the new generation are ordered and given the available resources, following the procedure described above. The allocation continues until the resources are exhausted, or until the initial population size is reached. So far, this is the original resource-limited GP, but now comes the decision on whether to raise the resource limit.

The rejected individuals are now given a second chance. In turn, each of them is reconsidered as a candidate for the new generation, and as many as possible are accepted, as long as their inclusion causes an improvement of the mean population fitness. This improvement may be relative to the best-of-run mean population fitness, or to the mean population fitness of the previous generation, creating two different implementation options we will call DynRes and DynResLight, respectively. DynResLight is expected to implement a limit that is raised much more easily, hence the name. As soon as one of the previously rejected individuals is rejected again, the process of reselection stops and the resource limit is increased to provide the additional needed resources. The pseudo code of the resource allocation procedure for both static and dynamic variants, as well as an example, are available in [17].

The dynamic approach to resource-limited GP achieved good performance when compared to the static approach and to the traditional depth limits in the Symbolic Regression of the quartic polynomial, obtaining similar fitness with significantly lower resource usage. Its performance was excellent in the Santa Fe Artificial Ant problem, where the same results were also achieved using significantly less resources, and showing fine prospects of converging to the optimal solution much sooner than the other techniques.

4 Experiments

The aim of these experiments is to provide a comparative study of the two approaches described above: tree depth limits and resource-limited GP. We compare not only their performance in terms of fitness reached and effort expended to achieve it, but also their behavior in terms of code growth and population size. Given the large number of variations available for each technique (see Sections 2 and 3), we have decided to limit our experiments to a subset that we believe to be a good representative of the entire range of expected behaviors. Our options are explained later in this section.

All the experiments were performed on three different problems: Symbolic Regression, Even Parity, and Artificial Ant.

For the Symbolic Regression problem we used 21 points of the quartic polynomial \( (x^4 + x^3 + x^2 + x) \), equidistant in the interval \(-1\) to \(+1\). The function and terminal sets were \{+, −, ×, ÷, sin, cos, log, exp\} (protected as in [12]) and \{x\}, respectively. For the Even Parity problem we used only 3 bits, the simplest form of the problem. The function and terminal sets were \{nand, nor, and, or\} and
terminal sets were \{x_1, x_2, x_3\}. For the Artificial Ant problem we used the Santa Fe trail where each ant was given 400 time steps to search for the 89 food pellets available. The function and terminal sets were \{if-food-ahead, progn2, progn3\} and \{left, right, move\}, as defined in [12].

For all the problems, an initial population of 500 individuals (Ramped Half-and-Half initialization [12] with maximum tree depth 6) was evolved for at least 50 generations (see Section 5 for details), even if the optimal solution was found earlier. Tree crossover was the only genetic operator used, and reproduction rate was set at 0.1. Selection for survival used no elitism. All the results presented refer to mean values found over 50 runs, for each of the following techniques:

- **Depth** → static tree depth limit (traditional)
- **DynDepth** → dynamic tree depth limit (non-heavy)
- **DynRes** → dynamic resource limit (normal)
- **DynResLight** → dynamic resource limit (light)

The Depth technique is the traditional static tree depth limit [12], with the typical value of 17. We use it as a baseline for comparison because of its high popularity, and also to stress the improvements introduced when DynDepth is used instead. DynDepth is the dynamic limit in its original form [13], meaning it applies to tree depth and does not use the heavy variation (see Section 2 for details). Although the heavy depth limit performed better than the original form [14], we have decided that using it would only be fair if comparing to a heavy version of resource-limited GP (see Section 6). The initial value for the dynamic depth limit was chosen to be the lowest possible, being set at 6, the same value as the maximum depth of the initial trees and the one that produced better results. The following techniques, DynRes and DynResLight, implement the dynamic versions of resource-limited GP (see Section 3 for details). We have decided not to use the static version of the resource limit because we never considered it to be finished work, but only used to prove that, with a suitable limit, it can achieve the same fitness using approximately the same amount of resources, thus being able to replace tree depth limits without impairing performance [16]. As with the dynamic depth limit, the initial value for the dynamic resource limit was chosen to be the lowest possible, also as in the previous work [17], being set at the exact amount of resources used by the initial generation. As described, DynRes raises the resource limit only if it causes an improvement of the best-of-run mean population fitness, while DynResLight raises it as long as it results in better mean population fitness than the previous generation.

All the experiments were performed using the GLLAB toolbox [22]. 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 show the performance and dynamics of the four techniques on the three problems, as explained in the previous section. Each problem is approached in a different subsection, but their plots are arranged together in two groups, each addressing all the problems. The first group shows the amount of resources used per generation, and the relationship between fitness and the computational effort expended to achieve it. Each plot contains the results of the four experiments, based on the mean values over the 50 runs performed for each experiment. The amount of resources per generation is simply the sum of the number of nodes of all the trees belonging to each generation. The computational effort can be roughly expressed as the total number of nodes evaluated so far, in other words, the cumulative amount of resources used. The relationship between fitness and effort is obtained by plotting, for each generation, the best fitness achieved against the effort expended so far (mean values over 50 runs). The second group of plots shows the evolution of mean tree size (average number of nodes per tree) and population size (number of individuals). Each contains 50 light dotted lines corresponding to the 50 runs, as well as one dark solid line representing the mean values.

The evolution lasted for 50 generations for most problems and techniques, which proved to be enough time for the majority of runs to find the optimal solution in the two simple problems of Symbolic Regression and Even Parity. In the Artificial Ant problem, however, stopping all the runs after 50 generations would result in large differences in the fitness achieved by each technique, not because of divergent performances, but because of the widely different amounts of resources used by the various techniques. So we gave extra generations to the more sparing techniques, until they reached the resource usage of the less sparing ones, thus providing comparable results. The baseline was the amount of resources used by the Depth technique in 50 generations.

#### 5.1 Symbolic Regression

Figure 1 shows the amount of resources used per generation by each of the techniques listed in Section 4, on the Symbolic Regression problem. It shows that Depth is the least sparing technique, finishing the 50 generations with a much higher resource usage than the other techniques. Replacing this static limit by the dynamic tree depth limit produces a large difference in resource usage, as revealed by the DynDepth curve, that does not increase much from its initial value. This behavior is also apparent in both dynamic resource techniques (DynRes, DynResLight) but, as expected, DynResLight increases its resource usage more easily than DynRes.

Figure 4 shows the relationship between best (lowest) fitness and the computational effort spent in achieving it, for all techniques. Because the computational effort is just another way of expressing the (cumulative) amount of resources used, this plot once again reveals the large consumption difference between Depth and the remaining tech-
Figure 1: Resource usage by each technique, per generation, on the Symbolic Regression problem.

Figure 2: Resource usage by each technique, per generation, on the Even Parity problem.

Figure 3: Resource usage by each technique, per generation, on the Artificial Ant problem.

Figure 4: Best fitness versus computational effort, on the Symbolic Regression problem.

Figure 5: Best fitness versus computational effort, on the Even Parity problem.

Figure 6: Best fitness versus computational effort, on the Artificial Ant problem.
niques. But it also reveals that, regardless of the effort demanded by each technique, by the end of the run they have all achieved similar values of best fitness (note the logarithmic scale). Although the differences in computational effort are statistically significant (in all comparisons except between DynDepth and DynResLight), the differences between the best fitness achieved are not. DynRes was the less demanding technique in terms of computational effort.

Figure 7 reveals what happens during the run regarding the evolution of mean tree size (average number of nodes per tree). Figure 10 (left) shows the effect this code growth has on the population size (number of individuals) of the resource-limited techniques (the others do not vary the initial number of individuals, 500). In Figure 7, the large difference between the amounts of nodes used by Depth and DynDepth becomes once again apparent. It is also not surprising to see that both these techniques, the ones that impose restrictions at the individual level, have a much slower code growth than DynRes and DynResLight (note the scale range difference). The dynamic resource techniques allow an easy growth of their trees, which inevitably triggers the main ability of resource-limited GP: reducing the population size to compensate for the higher tree size. If Figure 10 one can see the number of individuals in the population steadily decreasing after the first quarter of the run, to finish with only a small percentage of its initial value. DynResLight is generally able to sustain a higher number of individuals than DynRes, due to its higher ability to increase the resource limit.

5.2 Even-3 Parity

Figure 2 shows the amount of resources used per generation by each of the tested techniques, on the Even Parity problem. The relative behavior of the several techniques is similar to what happened on the Symbolic Regression problem (previous subsection). Depth is the technique that uses the most resources, with DynDepth showing a strikingly more sparing behavior. Unlike on the previous problem, Depth is however capable of reducing its resource usage in the last stages of the run, probably due to the Lexicographic Parsimony Pressure tournament that is highly efficient in reducing the usually high amount of introns present in Parity problems [7]. DynResLight once again shows an easier increase of resource usage than DynRes, substantially higher than on the previous problem, resulting in a final resource usage much closer to the amount used by Depth.

Figure 5 shows the relationship between best (lowest) fitness and computational effort. As on the previous problem, although the differences in computational effort are statistically significant (with no exceptions), the differences between the values of best fitness achieved are not. Once again DynRes was the less demanding technique in terms of computational effort.

Figure 8 shows the evolution of mean tree size of all techniques, while Figure 10 (center) shows the variation of population size of both resource-limited techniques. The differences in code growth between the individual-level and population-level techniques are not as large as the ones verified on the previous problem. In the Even Parity problem, DynDepth is the only technique that stands out for its slower code growth rate. The population size drops much sooner and much steeply than on the previous problem and, unlike before, it soon stabilizes (DynRes) or even rises again (DynResLight). The much higher amount of resources used by DynResLight results, once again, in a generally higher number of individuals when compared to DynRes, although the behavior appears to be very heterogeneous between runs.

5.3 Artificial Ant

Figure 3 shows the resource usage per generation on the Artificial Ant problem. As mentioned before, additional generations were given to the more sparing techniques until they reached the same amount of cumulative resources used in 50 generations by the less sparing technique, Depth. So the techniques DynDepth, DynRes, DynResLight ran for 84, 230, 94 generations, respectively. The same tendencies verified on the previous problems are once again observed, even more markedly here. Depth uses a much larger amount of resources than DynDepth, and DynResLight quickly diverges from the most sparing technique, DynRes, easily surpassing DynDepth and practically reaching Depth. It should be noted, however, that 50 generations would not have been enough for DynResLight to exhibit such a notorious behavior, leading us to believe that, if given enough time, this technique could reveal itself to always be the most expensive of all.

Figure 6 shows the best (highest) fitness achieved against the computational effort expended. Considering that not all techniques were given the same number of generations, each of their curves contains a cross to mark the point when generation 50 was completed. After using the same amount of computational effort, the four techniques reveal statistically significant differences between their values of best fitness (except between Depth and DynResLight, where the difference is not significant). DynRes achieves the best fitness, followed by DynDepth and finally both Depth and DynResLight. This plot proclaims DynRes to be the winner technique, but it also reveals a tendency that could alter this verdict in the long term (see Section 6 ahead).

Figures 9 and 10 (right) show the evolution of mean tree size and population size on the Artificial Ant problem, for all techniques. Once again, replacing the static tree depth limit (Depth) with the dynamic depth limit (DynDepth) causes a dramatic decrease of the code growth rate. But like on the Symbolic Regression problem, it is the resource-limited techniques (DynRes, DynResLight) that produce the largest difference in code growth when compared to the tree depth techniques (note the different scale ranges). The evolution of mean tree size appears to be highly erratic in DynRes, with several sudden variations along the run, and much more consistent in DynResLight. The reason is probably related to the sharp decrease in population size that DynRes exhibits from the very beginning of the run, reaching and consistently maintaining very low values until the end of the run. The mean tree size is thus prone to sudden variations, because it is calculated using so few individuals. DynRes-
Figure 7: Evolution of mean tree size (average number of nodes per tree) on the Symbolic Regression problem.

Figure 8: Evolution of mean tree size (average number of nodes per tree) on the Even Parity problem.

Figure 9: Evolution of mean tree size (average number of nodes per tree) on the Artificial Ant problem.

Figure 10: Evolution of population size (number of individuals) with the dynamic resource techniques on the Symbolic Regression problem (left), Even Parity problem (center), and Artificial Ant problem (right).
Light does not reach such a small population size, but also drops and quickly stabilizes right after the beginning of the run.

6 Discussion and Future Work

From the previous results one can tell that the non-light version of the dynamic resource limit (DynRes) was the best performing technique. On the two simple problems of Symbolic Regression and Even Parity, it managed to reach the same best fitness demanding significantly less computational effort or, in other words, using significantly less resources than the remaining techniques. On the more complex problem of the Artificial Ant, if reached significantly higher fitness using the same amount of computational resources.

According to the same criteria, the technique scoring the second place was the dynamic tree depth, DynDepth. It is interesting to note that DynRes and DynDepth do not use the same approach to bloat control, the first being population-based, and the second an individual-based technique. As one of the main goals of this study was to understand which of the two approaches achieves the best results in a set of different problems, we stress the importance of this ranking and look closer at the results achieved in Section 5.3, namely Figure 6. For two thirds of the run, the best performances were achieved by both resource-limited techniques, DynRes and DynResLight (although DynDepth always followed closely). But the initially steep curves of these two techniques soon began to loosen their inclination, while both tree depth techniques managed to maintain a more or less constant slope. The result was that DynDepth achieved a higher fitness by the end of the run, thus scoring second place. If this tendency persists from this point onward, one can easily guess that DynRes will be rapidly surpassed by DynDepth and, given enough generations, maybe also by the currently worst performing technique, Depth.

The reasons for this predicted long term failure of the resource-limited techniques, after such a bright initial superiority, may be precisely related to the problem we are trying to solve: bloat. Regardless of our formal definition of bloat (see [17] for a short discussion on this topic), when looking at Figure 9 we cannot help but intuitively conclude that tree depth limits can prevent this problem much better than resource-limited GP (if the latter can prevent it at all). By looking at Figure 10 (right) we can also guess that the main ability of resource-limited GP, compensating higher tree size with lower population size, may in fact become its biggest handicap. The population size may drop too low to enable any further exploration of the search space.

After elevating a resource-limited technique to the highest rank, we are once again [17] forced to wonder if this is the right approach to bloat control. Given its population-based nature, resource-limited GP does nothing to directly counteract code growth, and because this is an ever-present force in any search procedure with variable-length representations [5, 23], the population reduction will always be a side effect of this approach. To counteract this useful but potentially harmful phenomenon, we must act against code growth directly, or provide a contrary force that will keep pushing the number of individuals upwards while the emergent features of resource-limited GP try to bring it downwards. To act against code growth directly without abandoning the advantages provided by the population-based concept, we may try to combine individual-based restrictions with the higher level resource limit. We may also once again draw inspiration from the variations introduced in the original dynamic limits [14] to implement a heavy limit that forces itself down whenever possible. Alternatively, we may focus our attention on the promising dynamic depth technique and find ways to widen its applicability to non tree-based technique without impairing its performance.

7 Conclusions

We have described and compared two different approaches to bloat control in Genetic Programming: tree depth limits and resource-limited GP. Tree depth limits operate at the individual level, avoiding excessive code growth by imposing a maximum value on the depth of each tree-based individual. Resource-limited GP operates at the population level, limiting the total amount of resources the entire population can use. Because they aim at different targets of the GP paradigm, they produce radically different dynamics of the evolutionary process. Among the several variations available for both approaches, we have decided to test four techniques that seem to cover the range of expected behaviors: the traditional tree depth limit, the dynamic depth limit in its original form, and the two dynamic resource techniques, normal and light.

The results obtained in three different problems proclaim the more restrictive (non-light) version of the dynamic resource limit as the best performing technique. On the two simple problems, Symbolic Regression of the quartic polynomial and Even-3 Parity, this technique reached the same fitness demanding significantly less computational effort. On the more complex problem of the Artificial Ant in the Santa Fe trail, if reached significantly higher fitness using the same amount of computational resources. However, a closer look at the results revealed that the best approach to bloat control is yet to be determined.

Acknowledgments

We acknowledge grant SFRH/BD/14167/2003 from Fundação para a Ciência e a Tecnologia, Portugal. We thank all the members of the Evolutionary and Complex Systems Group (Universidade de Coimbra) for the fruitful discussions on this topic, especially Nuno Costa for some interesting ideas. We also thank Pedro J. N. Silva (Universidade de Lisboa) for many useful suggestions provided along this work, and for carefully reviewing the manuscript. Finally, we thank the Biomathematics Group at ITQB/UNL for allowing us the use of their computational resources.
Bibliography


