Resource-Limited Genetic Programming: The Dynamic Approach

Sara Silva
Evolutionary and Complex Systems Group
Centre for Informatics and Systems
of the University of Coimbra
Polo II, 3030 Coimbra, Portugal
sara@dei.uc.pt

Ernesto Costa
Evolutionary and Complex Systems Group
Centre for Informatics and Systems
of the University of Coimbra
Polo II, 3030 Coimbra, Portugal
ernesto@dei.uc.pt

ABSTRACT

Resource-Limited Genetic Programming is a bloat control technique that imposes a single limit on the total amount of resources available to the entire population, where resources are tree nodes or code lines. We elaborate on this recent concept, introducing a dynamic approach to managing the amount of resources available for each generation. Initially low, this amount is increased only if it results in better population fitness. We compare the dynamic approach to the static method where a constant amount of resources is available throughout the run, and with the most traditional usage of a depth limit at the individual level. The dynamic approach does not impair performance on the Symbolic Regression of the quartic polynomial, and achieves excellent results on the Santa Fe Artificial Ant problem, obtaining the same fitness with only a small percentage of the computational effort demanded by the other techniques.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms, Experimentation, Performance

Keywords

Evolutionary computation, genetic programming, bloat, code growth, limited resources, dynamic limits

1. INTRODUCTION

Genetic Programming (GP) is a problem solving methodology that evolves 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].

Several explanations for bloat have been proposed (see [2–5] for recent work), and many different bloat control techniques have been tried with various degrees of success, most of them based on parsimony pressure (see [6–10] for a small sample). But no other method was ever as popular as the traditional usage of a static tree depth limit imposed on the individuals accepted into the population, on a tree-based GP system [11]. This may not even be a good bloat control method [12,13], but it is still widely used, in spite of its several disadvantages. This limit effectively avoids the growth of trees beyond a certain point, but it does nothing to control bloat until the limit is reached. Its strict nature 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.

Various approaches have been tried in order to overcome these difficulties. Some rely on choosing specialized genetic operators to keep tree growth under control, without imposing strict limits [14, 15]. Recent work on Dynamic Limits [16, 17] has achieved promising results without the need for specific operators.

A different approach, based on limiting the total amount of tree nodes of the entire population, instead of imposing limits at the individual level [18] has been further explored very recently [19], introducing and testing the concept of Resource-Limited GP, in a simple symbolic regression problem. The present work elaborates on this idea, testing it with a harder problem and extending it with the inspiration provided by the mentioned work on Dynamic Limits [16,17].

The next section describes previous work regarding the two main concepts involved in this paper: Dynamic Limits and Resource-Limited GP. Section 3 explains how these concepts were hybridized to produce a new technique we call Dynamic Resource-Limited GP. Section 4 describes the experiments performed in two problems, Symbolic Regression and Artificial Ant. Section 5 reports the results obtained, while Section 6 discusses some methodological issues. Finally, Section 7 draws some conclusions and points towards future directions of this work.

2. PREVIOUS WORK

Since the present work is mainly based on the hybridization of two existing concepts, Dynamic Limits and Resource-Limited GP, this section is dedicated to describing them.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

2.1 Dynamic Limits

Dynamic Maximum Tree Depth [16] is a bloat control technique inspired in the traditional tree depth limit. Traditionally, whenever crossover creates an offspring that is deeper than the limit, one of its parents is chosen for the new generation instead. The same happens in Dynamic Maximum Tree Depth but, unlike the traditional limit, this one is *dynamic*, meaning it can be changed during the run. Initially set with a low value (but at least as high as the depth of the initial random trees), it is raised whenever needed to accommodate a new best-of-run individual that would otherwise break the limit. The result is a succession of limit risings, as the best solution becomes more accurate and more complex.

The original idea has been extended to implement a *heavy* dynamic limit that is lowered once the new best-of-run individual allows it, and to deal with non tree-based GP by introducing a dynamic limit on *size*, where size is the number of nodes, regardless of depth.

Dynamic Maximum Tree Depth was tested in two simple problems, Symbolic Regression of the quartic polynomial and Even-3 Parity, where it proved to effectively control bloat maintaining the ability to find good solutions [16]. Two initial values for the dynamic limit were tested, 6 (the depth of the initial random trees) and 9, and even the most restrictive option (6) did not pose any problems, 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 [17].

2.2 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 [18]. Very recently, it has been further developed by [19], where the concept of Resource-Limited GP was introduced. 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 the offspring, followed by their parents. Each of these groups is ordered by fitness, regardless of size. The queued candidates are then given the resources they need (their number of nodes) 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, or until population size restrictions 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.

```
sort offspring by fitness
sort parents by fitness
list = offspring followed by parents

if steady, my_popsize = initial_popsize
if low,    my_popsize = previous_popsize

---

resources_used = 0
accept_list = empty

for all individuals in list

  resources_i = resources needed by individual

  if resources_used + resources_i <= resource_limit
    accept_list = accept_list + individual
    resources_used = resources_used + resources_i

  if length of accept_list = my_popsize
    break for

new generation = accept_list
```

Figure 1: Pseudo code of the resource allocation procedure.

The resource-limited approach 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. 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 [20–23].

After the resources have reached the exhaustion point and the population size has been reduced, a new generation of individuals may use them more sparingly and leave enough unused to allow the population size to increase again. Although not a frequent occurrence, this has introduced two different implementation options: After accepting as many individuals as the previous population size, (1) use the remaining resources to allow the survival of additional individuals of the previous generation - the parents who have not yet been accepted - by continuing the resource allocation procedure until the resources are exhausted, or until the initial population size is reached, or (2) do not use them, thus never allowing the population size to increase. The first option was designated as *Steady*, for it enforces a steady usage of resources, and the second was called *Low*, because it allows a possible low usage of resources. Figure 1 shows the pseudo code of the resource allocation procedure. See Figure 3 for an example.

Resource-Limited GP was tested on a simple problem, the Symbolic Regression of the quartic polynomial. To compare its performance with the traditional usage of depth limits at the individual level, a static resource limit (14500) was found that would provide an amount of cumulative resources (used during the entire run) similar to the cumulative amount used with the traditional limit on depth 17. The results showed that both Steady and Low techniques behaved in a similar manner, achieving the same performance as the traditional depth limit [19].

3. DYNAMIC RESOURCE-LIMITED GP

Like the traditional depth limit, the original Resource-Limited GP relies on a static limit, imposed in the beginning of the run and never changed until the end. This hardly reflects the needs of a search process that must grow its individuals in search of better solutions. Although Resource-Limited GP has the natural ability to compensate higher tree size with lower population size, in complex problems this may lead to a dangerous shrinking of population size, as code growth proceeds. On the other hand, providing enough static resources to last until the end of the run may lead to the occurrence of bloat from the very beginning. The need for a dynamic resource limit becomes obvious.

The dynamic approach to Resource-Limited GP naturally arises from the hybridization of the two concepts described in the last section. A dynamic resource limit is implemented, one that is initially set with a low value, and raised whenever it results in better mean population fitness. The details are described next.

After generating the offspring, the candidates to the new generation are ordered and given the available resources, following the procedure described in Section 2.2. The allocation continues until the resources are exhausted, or until the initial population size is reached, according to the Steady option. 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 *Dyn* and *DynLight*, respectively. DynLight is expected to implement a limit that is raised much easier, 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. Figure 2 shows the pseudo code of the reselection procedure. See Figure 3 for an example.

```
if dyn     , my_meanpopfit = best_meanpopfit
if dynlight, my_meanpopfit = previous_meanpopfit

-----------------------------------------------

reject_list = list - accept_list
current_meanpopfit = mean fitness of accept_list

if current_meanpopfit better than best_meanpopfit
  best_meanpopfit = current_meanpopfit

for all individuals in reject_list

  tmp_accept_list = accept_list + individual
  new_meanpopfit = mean fitness of tmp_accept_list

  if new_meanpopfit better than my_meanpopfit
    accept_list = tmp_accept_list
    resources_i = resources needed by individual
    resources_used = resources_used + resources_i

  else
    break for

  if length of accept_list = initial_popsize
    break for

new generation = accept_list
resource_limit = resources_used
```

Figure 2: Pseudo code of the reselection procedure.

Variables:

```
initial_popsize = 10
previous_popsize = 6
resource_limit = 400
best_meanpopfit = 42
previous_meanpopfit = 35
```

Candidates to the new generation:
(from left to right, 6 children followed by 6 parents, each group sorted by fitness - higher is better)

Id	C1	C2	C3	C4	C5	C6	P1	P2	P3	P4	P5	P6
Fitness	80	70	60	60	50	10	90	40	40	20	10	10
Size	90	100	50	100	80	80	80	70	70	10	20	10

New generation obtained with each technique:
(Dyn and DynLight begin the reselection from the Steady results)

Id	C1	C2	C3	C4	C5	C6	P1	P2	P3	P4	P5	P6
Steady	✓	✓	✓	✓	✗[1]	✗[1]	✗[1]	✗[1]	✗[1]	✓	✓	✓
Low	✓	✓	✓	✓	✗[1]	✗[1]	✗[1]	✗[1]	✗[1]	✓	✓	✗[2]
Dyn	✓	✓	✓	✓	✔	✘[3]	–	–	–	✓	✓	✓
DynLight	✓	✓	✓	✓	✔	✔	✔	✘[4]	–	✓	✓	✓

Reasons for not accepting individual:
[1] Resources not available
[2] `previous_popsize` exceeded - stop procedure
[3] `new_meanpopfit` worse than `best_meanpopfit` - stop procedure
[4] `initial_popsize` exceeded - stop procedure

Figure 3: Example of resource allocation and reselection procedures.

4. EXPERIMENTS

The aim of these experiments is to check whether the reported performance of the Resource-Limited GP on the Symbolic Regression of the quartic polynomial (see Section 2.2) holds on a much harder problem like the Artificial Ant with the Santa Fe trail, and to test the dynamic approach (both Dyn and DynLight methodologies, see Section 3) on both problems.

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 $\{+, -, \times, \div, sin, cos, log, exp\}$ (protected as in [11]) and $\{x\}$, respectively. 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 [11]. For both problems, an initial population of 500 individuals (Ramped Half-and-Half initialization [11] 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 reproduction used the Lexicographic Parsimony Pressure tournament [8] and selection for survival used no elitism. All the results presented refer to mean values found over 50 runs, for each of the following techniques:

None → no limits
Depth → traditional tree depth limit
Steady → static resources, forced steady usage
Low → static resources, possible low usage
Dyn → dynamic resources, improvement of best-of-run
DynLight → dynamic resources, improvement of previous

The first technique (None) does absolutely nothing to control the growth of trees. The second technique (Depth) uses the traditional tree depth limit [11] with the typical value of 17. We use it because of its high popularity. The following two techniques, Steady and Low, implement the original Resource-Limited GP as described in Section 2.2. They use limited static resources set at 14500 (Symbolic Regression) or 135000 (Artificial Ant). The last two techniques, Dyn and DynLight, implement the dynamic approach to Resource-Limited GP, described in the previous section. They use limited dynamic resources, initially set at exactly the same amount used by the initial random generation. As described, Dyn raises the resource limit only if it causes an improvement of the best-of-run mean population fitness, while DynLight raises it as long as it results in better mean population fitness than the previous generation.

When choosing the static resource limit for Steady and Low, we used the value suggested for the Symbolic Regression problem [19], and applied the same rationale to find a suitable value for the Artificial Ant problem (see Section 2.2): a limit such that, by the end of 50 generations, the total amount of resources provided during the entire run is similar to the amount used by the Depth technique [19]. Regarding the initial dynamic limit for Dyn and DynLight, we chose the lowest possible value, drawing inspiration (see Section 2.1) from the better performance achieved by the original Dynamic Limits under the same conditions [16].

All the experiments were performed using the GPLAB toolbox [24]. Statistical significance of the null hypothesis of no difference was determined with Kruskal-Wallis nonparametric ANOVAs at $p = 0.01$.

5. RESULTS

The following plots are based on the mean values over the 50 runs performed for each experiment. They cover four important elements of the Resource-Limited GP concept: resource usage, fitness *versus* computational effort, population size (defined as the number of individuals), and tree size (defined as the number of nodes). Most of the plots show the results obtained per generation, where each value is not dependent on the values of previous generations, the exception being the fitness *versus* computational effort plots. The computational effort can be roughly expressed as the total number of nodes evaluated – 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). Although the evolution lasted 50 generations for most techniques, some were given additional time (namely Dyn and DynLight in the Artificial Ant problem) to compensate for their very low resource usage, thus providing comparable results.

We also present boxplots regarding resource usage and fitness, where each technique is represented by a box and pair of whiskers. Each box has lines at the lower quartile, median, and upper quartile values, and the whiskers mark the furthest value within 1.5 of the quartile ranges. Outliers are represented by +, and × marks the mean.

5.1 Symbolic Regression

Figure 4 shows the amount of resources used per generation by each of the techniques listed in Section 4, on the Symbolic Regression problem (see also Figure 6, left, for a boxplot regarding the amount of cumulative resources). Due to its enormous resource usage, the None curve is only partially shown. When the 50 generations are complete this line reaches somewhere between 8×10^4 and 9×10^4. The dotted line represents the static resource limit (14500) imposed to techniques Steady and Low. The plot shows that both these techniques consume more resources than Depth until they reach the limit, after which their resource usage tends to (forcibly) stabilize very close to the limit. The three techniques (Depth, Steady, Low) do not show any statistically significant differences between their overall (cumulative) resource usage by the end of the run. As expected, both dynamic techniques (Dyn, DynLight) show significantly lower resource usage when compared to the rest. Also expected was the easier increase of DynLight resources when compared to Dyn, resulting in significantly higher resource usage by the end of the run. Although the Dyn and DynLight curves represent the real resource usage of the techniques, and not the dynamic resource limit, in practical terms the difference between them is perfectly irrelevant.

Figure 5 shows the relationship between best (lowest) fitness and the computational effort spent in achieving it, for all techniques. There are obvious differences in the effort demanded (during the same number of generations) by the different techniques (see also Figure 6, left). The None technique represents a terrible waste of resources; as intended, both static techniques (Steady, Low) spend approximately the same as Depth; both dynamic techniques (Dyn, DynLight) use very few resources. These findings had already been provided in the description of Figure 4, since the computational effort is just another way of expressing the cumulative amount resources used. However, the present figure provides the important information that, regardless of the effort demanded by each technique, after 50 generations they have all achieved similar fitness (note the logarithmic scale), with no significant differences between them. Figure 6 shows the boxplots regarding both computational effort (total amount of resources used in the entire run) and best fitness of run.

Figure 10 reveals what happens during the run regarding the evolution of tree size (number of nodes) and its consequences on population size (number of individuals). Note that Figure 4 is obtained by "multiplying" these two plots.

The left plot shows the mean tree size measured along the 50 generations, for all techniques. It is not surprising to see that Depth, the only technique that imposes restrictions at the individual level, has the slowest growth of mean tree size, finishing with a significantly lower mean tree size than

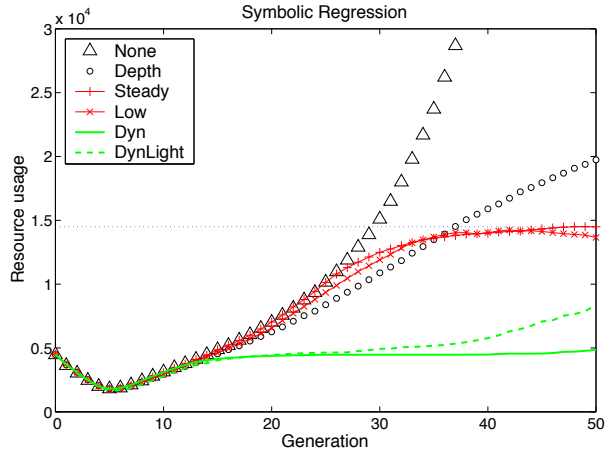

Figure 4: Resource usage by each technique, per generation, on the Symbolic Regression problem. The dotted line represents the static resource limit.

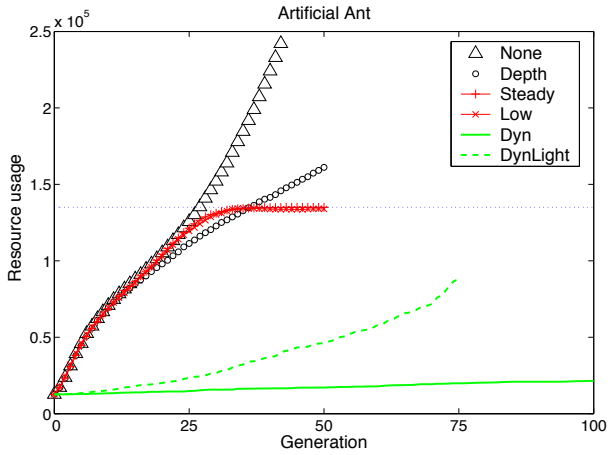

Figure 7: Resource usage by each technique, per generation, on the Artificial Ant problem. The dotted line represents the static resource limit.

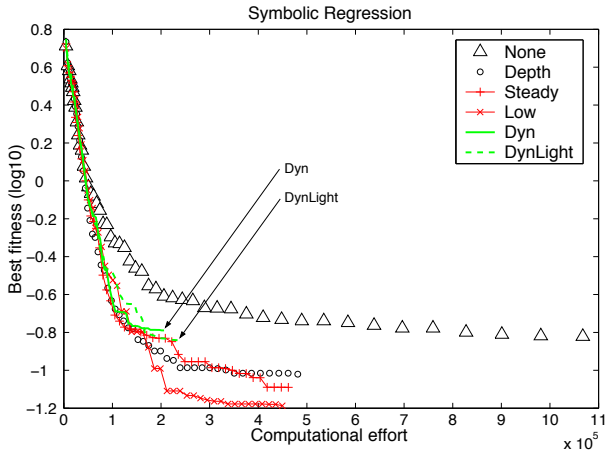

Figure 5: Best fitness as a function of computational effort, on the Symbolic Regression problem. The arrows mark the end of the run on Dyn and DynLight.

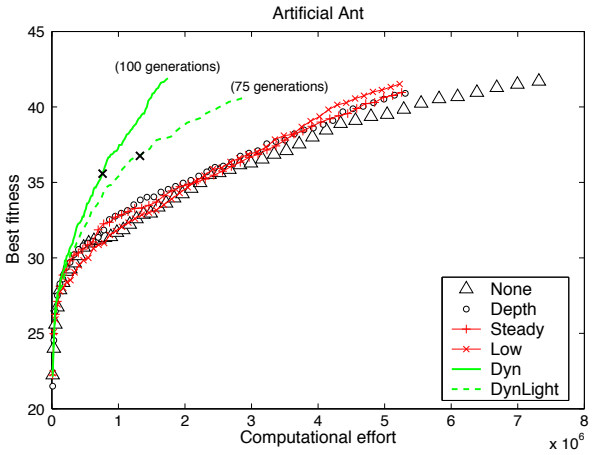

Figure 8: Best fitness as a function of computational effort, on the Artificial Ant problem. The crosses mark generation 50 on Dyn and DynLight.

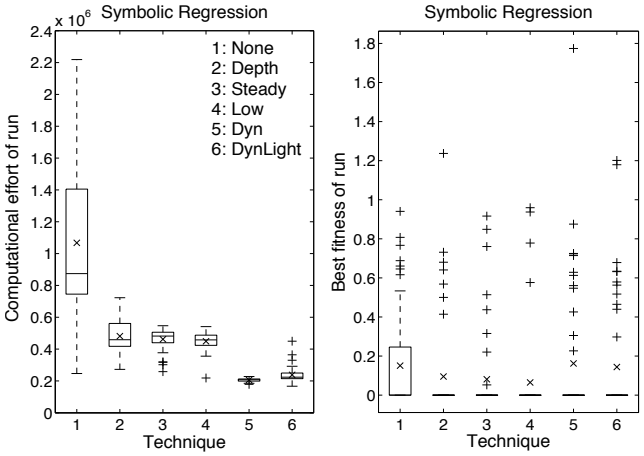

Figure 6: Boxplots of computational effort (left) and best fitness of run (right), on the Symbolic Regression problem.

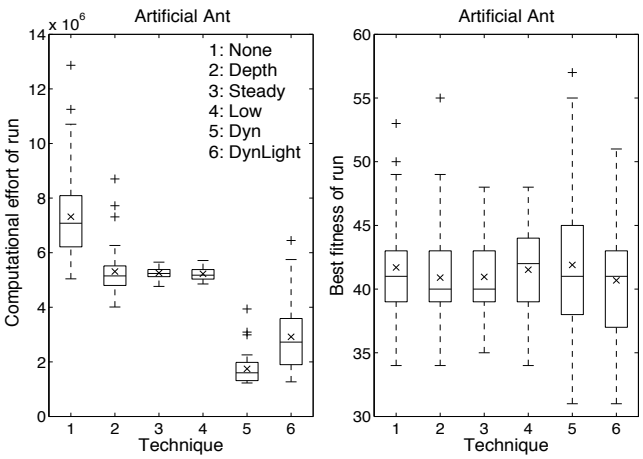

Figure 9: Boxplots of computational effort (left) and best fitness of run (right), on the Artificial Ant problem.

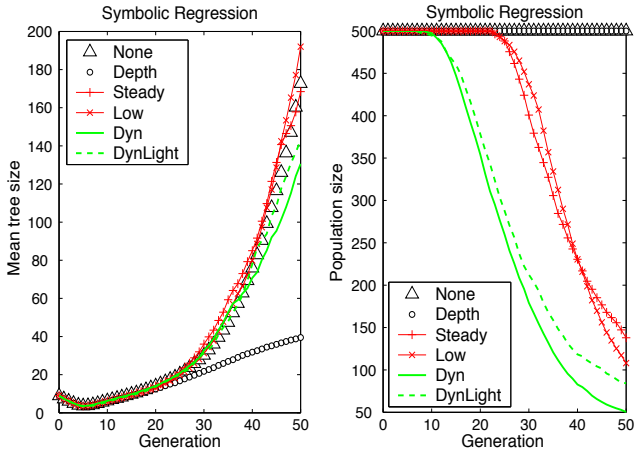

Figure 10: Evolution of mean tree size (left) and population size (right), on the Symbolic Regression problem.

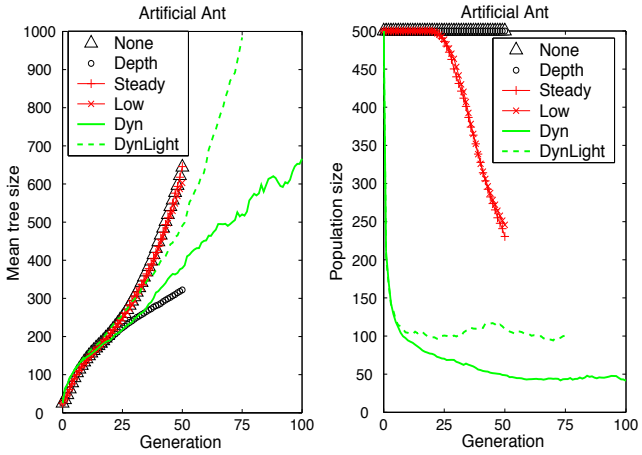

Figure 11: Evolution of mean tree size (left) and population size (right), on the Artificial Ant problem.

the rest. All the other techniques seem to allow free growth of their trees, with no significant difference in final mean tree size, except between Low and Dyn. The code growth observed in this plot inevitably triggers the main ability of all resource-limited techniques, both static and dynamic: reducing the population size to compensate for the higher tree size.

The right plot shows the evolution of population size along the 50 generations of the run. It reveals a sharp decrease of population size for all resource-limited techniques. Because of the initially fewer resources available to the dynamic techniques, these start dropping much sooner, around generation 10, while the static techniques only start around generation 20. By the end of the run, the population sizes of the dynamic techniques show a significant difference between them, and both appear to begin stabilizing. On the contrary, the static techniques also finish with no significant difference between them, but both maintaining a steep decrease in population size.

It should be noted (not shown in the plots) that half of the runs converge to the optimal solution around generation 15, and by the end of the 50 generations only 8% (Low) to 28% (None) of the runs are still looking for the optimal solution. The percentages of non-converged runs for the other techniques are 14% (Depth), 16% (Steady), 24% (Dyn), and 22% (DynLight). Summing up, the None technique has the lowest convergence rate (72%), followed by both dynamic approaches (76-78%) and the remaining techniques (84-92%).

5.2 Artificial Ant

Figure 7 shows the resource usage per generation for each of the tested techniques, on the Artificial Ant problem (see also Figure 9, left, for a boxplot regarding the amount of cumulative resources). Additional generations are shown for both Dyn and DynLight, allowing these techniques to reach appropriate fitness levels for comparison with the other techniques (see Figure 8). While the other techniques used 50 generations, Dyn and DynLight were given 100 and 75, respectively. As in Figure 4 (see Section 5.1), the None curve is only partially shown because of its high resource usage, reaching higher than 3×10^5 in generation 50. The dotted line once again represents the static resource limit (135000 for this problem) of techniques Steady and Low. The behavior of Depth and both static techniques (Steady, Low), namely the relationship between the three, is the same as previously observed for the Symbolic Regression problem (see Section 5.1), with no significant differences between their overall (cumulative) resource usage (see Figure 9, left). Also previously observed was a significantly lower resource usage by the dynamic techniques Dyn and DynLight. However, the DynLight limit (or actual usage, see comments in Section 5.1) now seems to emphasize its lightness and 'take off' from Dyn much sooner after the run begins.

Figure 8 shows the best (highest) fitness against the computational effort, or amount of resources, spent to obtain it. Techniques Dyn and DynLight where given additional generations (50 more and 25 more, respectively) because their very low resource usage would not allow them to reach comparable results otherwise. The crosses in their curves mark the point when generation 50 was completed. As it is, the differences between the best fitness achieved by any of the techniques are not statistically significant, thus providing the same base of comparison we had in the Symbolic Regression problem. Figure 9 shows the boxplots regarding both computational effort (total amount of resources used in the entire run) and best fitness of run.

Figure 11 shows the evolution of tree size (number of nodes) and population size (number of individuals) during the run. In the left plot, the Depth technique has a slow growth of tree size, as in the previous problem, but this time it is accompanied by the Dyn technique until around generation 40, when both curves diverge. By the end of the run (additional generations included), Depth shows significantly lower mean tree size that the other techniques, while Dyn shows no significant difference from None or the static techniques (Steady, Low). DynLight finishes with significantly higher mean tree size than the rest.

Regarding the evolution of population size, on the right plot, the static techniques behave much like they did in the previous problem, dropping steeply from around generation 20 and finishing with no significant differences be-

tween them, still decreasing steeply. The dynamic techniques, however, cause a vertiginous drop in population size from the very beginning of the run, tending to stabilize soon after. DynLight stabilizes much sooner than Dyn, and by the end of the run (additional generations included) the difference between both is statistically significant.

6. DISCUSSION

The results presented in the previous section show the effectiveness of the resource-limited techniques in reducing the amount of resources used and still obtaining the same results. Although the advantages of reducing computational effort are unquestionable, a deeper methodological issue remains. Because none of these techniques imposes restrictions at the individual level, nothing is done to directly counteract code growth. So is Resource-Limited GP preventing bloat, or simply learning to live with it? The emergent rule that prevents the survival of 'not good enough for their size' individuals (see Section 2.2) actually ensures that bloat does not occur, as long as we define bloat as "an excess of code growth without a corresponding improvement in fitness" (from Section 1). But how much code growth is considered *excessive*? And how much fitness gain is considered a *corresponding* improvement? The fact is, the methodology presented here allows seemingly free code growth, and the lower resource usage is only obtained thanks to a prompt reduction of population size.

We know it is possible to obtain similar results when code growth is restricted. The Depth technique, as well as the extremely restrictive Dynamic Limits [16, 17], have proved it. The ideal bloat control method should restrict the proliferation of introns without hindering the exploration of the search space. Curiously, in the Artificial Ant problem (where introns represent 70-80% of the code, not shown) the best intron control technique was Depth, the only one acting at the individual level, but in the Symbolic Regression problem (intron percentage less than 20%, not shown) it was the population-level techniques that achieved the best results. How much could we gain from imposing limits at the individual level *and* at the population level? Would the performance get equally good in both problems, or would the evolutionary process simply collapse under such pressure?

Resource-Limited GP alone does not seem to impose too much pressure on the evolution, even using the dynamic approach. The population size does suffer a large reduction, frequently dropping as low as only 10% of its initial size (see Figs. 10 and 11), but still the population diversity (based on the variety measure [25], the percentage of distinct individuals in the population) suffers little or nothing when compared to Depth (not shown). Of course the *absolute* number of unique individuals is inevitably decreased as the population size drops, but if anything, this seems to improve the convergence ability.

The performance of Dynamic Resource-Limited GP in the two problems tested ranged from good to excellent. In the Symbolic Regression problem, it provided similar fitness with significantly lower resource usage, but the convergence rate (see Section 5.1, last paragraph) to the optimal solution was lower than with the static approach. However, in the more complex problem of the Artificial Ant, the same fitness level was also achieved with significantly lower resource usage, with the results showing fine prospects of reaching the optimal much easier than the other techniques. Methodological issues aside, the dynamic approach to Resource-Limited GP appears to be a strong candidate to reducing the high computational effort expended by most GP systems.

7. CONCLUSIONS AND FUTURE WORK

The dynamic approach to Resource-Limited GP was implemented, tested and compared to the original static resource limit, and to the traditional usage of tree depth limits. It achieved good performance in the Symbolic Regression of the quartic polynomial, obtaining similar fitness with significantly lower resource usage, but a lower convergence rate than the static approach or the traditional depth limits. 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.

Although Dynamic Resource-Limited GP is already a hybridization of ideas from both Dynamic Limits [16, 17] and Resource-Limited GP [19], we plan to actually test how both techniques behave when used in conjunction. Drawing inspiration from the heavy dynamic limit [17], we also intend to implement a version of the dynamic approach where the available resources can be *reduced* whenever the evolutionary process justifies it. We may also adopt different criteria to decide when to raise (or decrease) the resource limit, and try different rankings for placing the individuals in the queue used by the allocation procedure. Finally, a careful comparison between all the mentioned techniques, both pure and hybridized, with detailed analysis of the strengths and weaknesses of each, should be performed in order to combine their best features in a single and powerful bloat control method.

8. ACKNOWLEDGEMENTS

We acknowledge grant SFRH/BD/14167/2003 from Fundação para a Ciência e a Tecnologia, Portugal. We also thank Pedro J. N. Silva from Universidade de Lisboa for carefully reviewing the manuscript, and the Biomathematics Group at ITQB/UNL for allowing us the use of their computational resources. Finally, we acknowledge the GECCO reviewers for providing so many useful suggestions.

9. REFERENCES

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

[2] T. Soule and R. B. Heckendorn. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3:283–309, 2002.

[3] S. Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106, 2003.

[4] M. Brameier and W. Banzhaf. Neutral variations cause bloat in linear GP. In C. Ryan, T. Soule, M. Keijzer *et al.*, editors, *EuroGP-2003*, pages 286–296, Essex, UK, 2003. Springer.

[5] M. J. Streeter The root causes of code growth in genetic programming In C. Ryan, T. Soule, M. Keijzer *et al.*, editors, *EuroGP-2003*, pages 449–458, Essex, UK, 2003. Springer.

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

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

[8] S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. Langdon, E. Cantú-Paz, K. Mathias, *et al.*, editors, *GECCO-2002*, pages 829–836, New York, NY, USA, 2002. Morgan Kaufman.

[9] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, *et al.*, editors, *EuroGP-2003*, pages 204–217, Essex, UK, 2003. Springer.

[10] L. Panait and S. Luke. Alternative bloat control methods. In K. Deb, R. Poli, W. Banzhaf, *et al.*, editors, *GECCO-2004*, LNCS, pages 630–641, Seattle, WA, USA, 2004. Springer.

[11] J. R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. The MIT Press, Cambridge, Massachusetts, USA, 1992.

[12] C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, *et al.*, editors, *GP'96*, pages 291–296, Stanford University, California, USA, 1996. MIT Press.

[13] W. B. Langdon, R. Poli. An analysis of the MAX problem in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, *et al.*, editors, *GP'97*, pages 222–230, San Francisco, California, USA, 1997. Morgan Kaufman

[14] C. J. Kennedy and C. Giraud-Carrier. A depth controlling strategy for strongly typed evolutionary programming. In W. Banzhaf, J. Daida, A. E. Eiben, *et al.*, editors, *GECCO-1999*, pages 1–6, Orlando, Florida, USA, 1999. Morgan Kaufman.

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

[16] S. Silva and J. S. Almeida. Dynamic maximum tree depth - a simple technique for avoiding bloat in tree-based GP. In E. Cantú-Paz, J. A. Foster, K. Deb, *et al.*, editors, *GECCO-2003*, LNCS, Chicago, IL, USA, 2003. Springer.

[17] S. Silva and E. Costa. Dynamic limits for bloat control - variations on size and depth. In K. Deb, R. Poli, W. Banzhaf, *et al.*, editors, *GECCO-2004*, LNCS, pages 666–677, Seattle, WA, USA, 2004. Springer.

[18] N. Wagner and Z. Michalewicz. Genetic programming with efficient population control for financial time series prediction. In E. D. Goodman, editor, *GECCO-2001 Late Breaking Papers*, pages 458–462, San Francisco, CA, USA, 2001.

[19] S. Silva, P. J. N. Silva, and E. Costa. Resource-limited genetic programming: Replacing tree depth limits. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, *et al.*, editors, *ICANNGA-2005*, pages 243–246, Coimbra, Portugal, 2005. Springer.

[20] S. Luke, G. C. Balan, and L. Panait. Population implosion in genetic programming. In E. Cantú-Paz, J. A. Foster, K. Deb, *et al.*, editors, *GECCO-2003*, LNCS, pages 1729–1739, Chicago, IL, USA, 2003. Springer.

[21] F. Fernandez, L. Vanneschi, and M. Tomassini. The effect of plagues in genetic programming: A study of variable-size populations. In C. Ryan, T. Soule, M. Keijzer *et al.*, editors, *EuroGP-2003*, pages 317–326, Essex, UK, 2003. Springer.

[22] F. Fernandez, M. Tomassini, and L. Vanneschi. Saving computational effort in genetic programming by means of plagues. In R. Sarker, R. Reynolds, H. Abbass, *et al.*, editors, *CEC-2003*, pages 2042–2049. IEEE Press, 2003.

[23] M. Tomassini, L. Vanneschi, J. Cuendet, and F. Fernandez. A new technique for dynamic size populations in genetic programming. In *CEC-2004*, pages 486–493. IEEE Press, 2004.

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

[25] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, Alaska, USA, 1998. IEEE Press.