# Bloat Control Operators and Diversity in Genetic Programming: A Comparative Study

**E. Alfaro-Cid** evalfaro@iti.upv.es
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia, Spain

**J. J. Merelo** jmerelo@geneura.ugr.es
Dept. de Arquitectura y Tecnología de Computadores, Universidad de Granada, Granada, Spain

**F. Fernández de Vega** fcofdez@unex.es
Grupo de Evolución Artificial, Universidad de Extremadura, Mérida, Spain

**A. I. Esparcia-Alcázar** anna@iti.upv.es
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia, Spain

**K. Sharman** ken@iti.upv.es
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia, Spain



**Abstract**

This paper reports a comparison of several bloat control methods and also evaluates a recent proposal for limiting the size of the individuals: a genetic operator called *prune and plant*. The aim of this work is to test the adequacy of this method. Since a preliminary study of the method has already shown promising results, we have performed a thorough study in a set of benchmark problems aiming at demonstrating the utility of the new approach. Prune and plant has obtained results that maintain the quality of the final solutions in terms of fitness while achieving a substantial reduction of the mean tree size in all four problem domains considered. In addition, in one of these problem domains, prune and plant has demonstrated to be better in terms of fitness, size reduction, and time consumption than any of the other bloat control techniques under comparison. The experimental part of the study presents a comparison of performance in terms of phenotypic and genotypic diversity. This comparison study can provide the practitioner with some relevant clues as to which bloat control method is better suited to a particular problem and whether the advantage of a method does or does not derive from its influence on the genetic pool diversity.

**Keywords**

Bloat control, genetic programming, diversity.

## 1 Introduction

One of the main problems encountered by genetic programming (GP) researchers and which may also affect any evolutionary algorithm (EA) employing variable-length individuals is that of code bloat (Koza, 1992; Langdon and Poli, 2002; Luke, 2000b;

 

Poli et al., 2008a; da Silva, 2008). On the one hand, this phenomenon causes waste of computer resources due to the evaluation of large programs. On the other hand, it causes difficulties in the understanding of the final solutions, due to the fact that large parts of the program do not contribute to the final result. Finally, the application of genetic operators in parts of the program that do not contribute to fitness inhibits the convergence by hampering the modification of programs in a meaningful way.

Poli et al. (2008a) define bloat as "program growth without (significant) return in terms of fitness." It should not be mistaken for the growth of programs to acquire the necessary complexity to find a good solution to the problem at hand. The problem of bloat was reported in very early stages of the research in GP (Koza, 1992; McPhee and Miller, 1995; Soule et al., 1996; Nordin et al., 1996; Tackett, 1994) and so far there is not a generally accepted theory to explain this phenomenon, although significant advances have been made in the field.

Classical theories (based on empirical observations) that try to explain bloat include: the *hitchhiking theory*, the *defense against crossover theory*, the *replication accuracy theory*, the *removal bias theory* and the *nature of program search space theory*. Most of them are based on the concept of *introns*, areas of code that can be removed without altering the fitness value of the solution; from the concept of introns in biology, non-coding regions of the DNA (Rzhetsky and Ayala, 1999), that is, those that eventually do not end up as part of a protein.

The hitchhiking theory is proposed by Tackett (1994), who proved that random selection in conjunction with standard subtree crossover (i.e., when fitness is completely ignored) does not cause code growth and therefore it is concluded that fitness is the cause of the increasing of the size. Then, it is argued that the preservation of important bits of code by the crossover operator leads to introns or hitchhikers attached to these important bits being propagated by generation with them. The defense against crossover theory (Nordin et al., 1996) goes a step further. Nordin et al. (1996) argue that the role played by the introns is that of increasing the number of nodes of the tree, making it more difficult to destroy with crossover. That is, if a solution has a large amount of redundant data it has fewer chances of being modified in a meaningful way by crossover. The replication accuracy theory (McPhee and Miller, 1995) argues that the success of a solution lies in its ability to have offspring that are similar to the parent fitnesswise. This is achieved by evolving toward bloated solutions that increase replication accuracy. The removal bias theory (Soule and Foster, 1998; Soule, 1998) states that, given that redundant data tend to be low in the tree (i.e., closer to the leaves than to the root) and applying crossover to redundant data does not modify the fitness of a solution, the evolution will favor the replacement of small branches. Since there is not such a bias for insertion, small branches will be replaced by average-size branches leading consequently to bigger trees. Finally, the nature of program search space theory (Langdon and Poli, 1997) is, unlike the previous theories, not based on introns. This theory relies on the idea that above a certain size, the distribution of fitness does not vary with size. Since in the search space there are more big tree structures than small ones, during the search process the GP will tend to find bigger trees.

Later on, Luke (2000a) presented experimental evidence against the claim that it is the crossover between introns that causes the bloat problem, concluding that: "tree growth is the cause of inviable code growth," not the consequence. The mechanism he proposes to explain code growth is a generalization of the removal bias theory. Usually there is an inverse relationship between the depth at which a node is located and its influence on the cost value generated in the tree evaluation, that is, the closer to the

root of the tree, the more influence on the fitness. Therefore, in very fit individuals there is a bias toward choosing deeper crossover points that will not damage the evaluation of the tree. Such a bias has two consequences, on the one hand this bias can promote larger parents. On the other hand, removed subtrees rooted in a deep node are more likely to be small, but the inserted subtrees have no such size bias.

Recently, new theories, more theoretically motivated, have been introduced. In Poli's work (Poli, 2001; Poli and McPhee, 2003), a *size evolution equation* was developed, which provided an exact formalization of the dynamics of average program size. Also, the *crossover bias theory* (Dignum and Poli, 2007, 2008; Poli et al., 2007, 2008b) states that while the mean size of programs is unaffected by crossover, higher moments of the distribution are. The population evolves toward a distribution where small programs have a higher frequency than longer ones. However, very small programs are unlikely to solve the problem. Therefore, large programs have a selective advantage and the population average size increases.

Some of these theories have led to general methods for controlling bloat. However, most bloat control methods in the literature are in fact motivated by empirical observations. A variety of approaches to tackle bloat have been suggested (see Bhattacharya, and Nath, 2001; Luke and Panait, 2006; Poli et al., 2008a for a review of the topic). Next, we present some of these approaches.

Probably the most popular methods to date for controlling bloat are those based on the imposition of size limits and the inclusion of structural complexity in the cost function. Both were initially proposed by Koza (1992). The first approach imposes a size limitation for the generation of individuals in the initial population, size limitation in the generation of subtrees for subtree mutation, and restriction of the crossover and mutation operators so that children larger than the permissible size are not included in the population. Size limitation can be made in terms of maximum allowed depth or maximum allowed length of a tree. The depth of a tree is the maximum length of the path of any node to the root. The length of a tree is the number of nodes of the tree. Although Koza (1992) proposed depth limits, the utility of length limits has also been assessed by other authors (Crane and McPhee, 2006). The second approach considers size as a criterion to include in the cost function. Usually the size term has to be weighted so that there is a balance between the emphasis placed on size and the emphasis placed on fitness. The main problem of this approach is the choice of the parameter that weights the size against the fitness-related value in the fitness function. A solution for this problem is the use of multi-objective techniques that consider size and fitness as separate objectives (Ekart and Nemeth, 2001) or setting the parsimony coefficient using the recently devised *covariant parsimony pressure method* (Poli and McPhee, 2008), which recalculates the parsimony coefficient at each generation to ensure that the mean value size of the population remains constant along the evolution.

Some other methods include the following.

- **Editing Operators.** Koza (1992) introduced an editing operator that periodically would simplify the trees, eliminating the subtrees that do not add anything to the final solution (e.g., subtrees that are multiplied by zero). However, he did not use it in his algorithm because the test did not conclusively show any benefit for its inclusion (drawbacks were not proved either). Ekart (1999) introduced a mutation operator that modifies the structure but does not alter the cost of a tree (i.e., it performs the algebraic simplification of the tree expression), in a similar way to Koza's editing operator.

- **Modification of Existing Genetic Operators or Creation of New Ones.** Luke (2000c) presented two methods for the initialization of trees that allow control over the expected tree size. Van Belle and Ackley (2002) created a special operator called *uniform subtree mutation* to prevent bloating. It consists of the application of standard subtree mutation a binomially distributed number of times. This binomial distribution is based on the size of the tree. This mutation operator increases the probabilities of undergoing mutation of the nodes in larger trees. In addition, Langdon (2000) and Crawford-Marks and Spector (2002) introduced *size fair operators*, *size fair crossover*, and *size fair mutation*. In size fair operators, the size of the subtree to be deleted is calculated and this is used to guide the random choice of the second crossover point. Other operators used in the 1990s are *hoist* and *shrink* (see Section 2).

- **Inclusion of Parsimony Pressure in the Selection Method.** This is accomplished either by selecting a proportion of individuals based on size (see *proportional tournament* in Section 3.1) or by doing two tournaments, one on fitness and another on size (see *double and lexicographic tournament* in Section 3.1). Other methods assign very bad fitness to big solutions in order to bias the selection (see *tarpeian* in Section 3.1).

- **Removal of Oversized Individuals from the Population.** Fernández and Martín (2004) proposed a method that works at population level instead of at individual level. It is called *the plague* and it cyclically removes some oversized individuals from the population after some generations. A similar method, called *death by size* (Panait and Luke, 2004; Luke and Panait, 2006) is used in conjunction with steady-state algorithms. The selection of individuals to kill is done by size instead of fitness.

Recently, an extensive study was published by Luke and Panait (2006) where several approaches to bloat control were examined and conclusions were drawn on their success in reducing average program size while maintaining the quality of the best-of-run result. The paper reports results on a per-problem basis and also identifies which settings of the method's parameters work well independently of the problem under consideration.

The seed for the work presented in this paper is Luke and Panait's work as well as a comparison study of bloat control methods we carried out to learn which one best suited the problem we were dealing with at that point: a bankruptcy prediction problem (Alfaro-Cid et al., 2008a). Our aim when applying a bloat control method was for GP to generate comprehensible classification models, that is, models that are easy to analyze and draw conclusions from. This aspect of creating models that are easy to understand is fundamental for an application design for making decisions to be used by people unrelated to the evolutionary computation field. An example would be the bankruptcy prediction tool reported in our previous work (Alfaro-Cid et al., 2008a). In Alfaro-Cid et al. (2008a), we compared four bloat control methods to Koza's method of restricting the tree depth. Three of the bloat control methods were chosen among those that provided better results in Luke and Panait (2006): double tournament, tarpeian, and lexicographic parsimony pressure with ratio bucketing. Linear parametric parsimony pressure was not included in spite of the good results it provided in Luke and Panait (2006). The reason was the inherent difficulty in tuning the parsimony pressure. The paper by Luke and Panait (2006) attributes the success of linear parametric parsimony pressure to a lucky choice of problem domains. The fourth method was prune and

plant, a method introduced in Fernández de Vega et al. (2005) by one of the authors of this paper. Surprisingly, prune and plant beat all the other methods for the bankruptcy prediction problem.

The objective of this work is to test the adequacy of this new bloat control method in a broader scenario by applying it to a set of benchmark problems. One of the issues we are looking at is the importance of maintaining diversity while controlling bloat. Our assumption is that prune and plant maintains a good diversity since the plant operation allows that all the genetic material is kept in the genetic pool. We studied both the genotypic and phenotypic diversity of all methods under comparison, as well as their correlation with fitness and size.

The paper is structured as follows. Section 2 describes the prune and plant operator. Section 3 explains the experimental procedure followed. Section 4 presents a comparison study to select the most appropriate parameter for the prune and plant operator. Section 5 shows the experimental results. Section 6 discusses the results obtained and, finally, in Section 7 some conclusions and future working lines are drawn.



## 2 Prune and Plant

This new bloat control approach is described for the first time in Fernández de Vega et al. (2005) and published in English for the first time in Alfaro-Cid et al. (2007) and more recently in Alfaro-Cid et al. (2008b). Prune and plant is inspired by the strategy of the same name used in agriculture. It is used mainly for fruit trees and it consists of pruning some branches of trees and planting them in order to grow new trees. Once these branches are planted, they take root and grow to become adult trees. Adapting the idea in the context of GP, a selected individual will have one of its branches pruned and substituted by a terminal uniform randomly selected from the terminal set. The pruned branch will be planted in the population as a new tree. This way, both the offspring trees will, on average, be of smaller size than the ancestor, effectively reducing average tree size.

Although the idea was inspired by the agricultural metaphor, some authors have used similar operators in the past, mainly in the 1990s. Angeline (1997) used several mutation operators, among them the operator shrink, which removes a branch of a tree and replaces it with a terminal. The aim of the work was to demonstrate that macromutation could perform as well as subtree crossover under identical experimental conditions. That is, there was no intention of using the shrink operator as a way to control bloat. In fact, he used it in combination with a grow operator that did exactly the opposite: to replace a terminal node with a randomly generated subtree. Chellapilla (1997) used the same shrink operator but this time under the name of trunc. Again the aim of the paper was to test the viability of algorithms that did not include any recombination operator; the aim was not controlling bloat.

On the other hand, Kinnear (1993) introduced a new mutation operator called hoist. Hoist selects an inner node and returns a copy of this subtree as a new individual. Although the objective of the work was the application of GP to the task of evolving iterative sorting algorithms, Kinnear did mention that hoist "was created in order to increase the probability even further of the results of genetic operators creating smaller individuals." Subsequently, hoist has been used in Araujo (2004), where it was called cut.

In Fernández de Vega et al. (2005) prune and plant was compared with hoist for the artificial ant problem (see description of the problem in Section 3.2). Prune and plant outperformed hoist in both the quality of the results and the computational effort

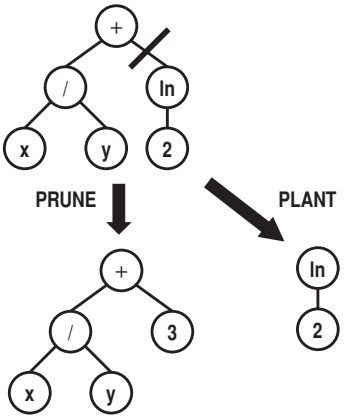


Figure 1: Prune and plant operator. A branch of the original tree is pruned and constitutes a new tree; it is substituted by a random leaf.

needed to obtain them. Summarizing, although the inspiration for prune and plant came from the observation of a real-world practice, it could be considered as the combination of two mutation operators introduced separately in the 1990s: shrink and hoist.

We have implemented prune and plant as a bloat-control genetic operator. Once an offspring is created by crossover or reproduction, prune and plant takes place with a certain probability. It is quite an unusual operator since it creates two offspring from a single parent (see Figure 1). This means that in a generational approach (like ours), the more frequently prune and plant is applied, the less crossover and reproduction operations are required to create the new population. The parameter to set is the rate of application of the operator. Since there are no previous studies on the amount of prune and plant to use, we have compared prune and plant rates ranging from 0.1 to 1, with 0.1 increment steps (that is, 10% to 100% of the individuals will get the new operator applied to them).

## 3 Description of the Experimental Procedure

The procedure used for the comparison study closely follows the experiments presented by Luke and Panait (2006).

The GP implementation is based on ECJ.[1] The setting of parameters applied during evolution follows the one used in Luke and Panait (2006): ramped half and half initialization, generational evolution, tournament selection (size of tournament equals 7), 10% of reproduction, 90% of biased subtree crossover (internal node selection rate equals 0.9), no mutation, and a population of 1,000 individuals running for 50 generations.

In order to assess the efficiency of the prune and plant method, it has been compared with four other bloat control methods: double tournament, proportional tournament, tarpeian, and lexicographic parsimony pressure with ratio bucketing. This choice was motivated by the good results shown by these methods across all problem domains in Luke and Panait (2006). All the bloat control methods have been combined with

[1] http://www.cs.gmu.edu/~eclab/projects/ecj

Koza's style tree depth restrictions, since strong experimental evidence has been provided that for the set of problems and bloat control methods addressed in Luke and Panait (2002), any method augmented this way is superior to the bloat control method alone. Also, depth limiting alone is used as a baseline. These methods will be described next.

### 3.1 Methods Tested

Double tournament is a method proposed by Luke and Panait (2006) that applies two layers of tournament in sequence, one for fitness and the other for size. The individuals in the final tournament group are not chosen at random with replacement from the population, but they are the winners of a previous tournament. If the final tournament selects based on fitness, then the previous tournaments select on size (or vice versa). Therefore, the algorithm has three parameters: a fitness tournament size ($F$) that we have fixed to 7; a parsimony tournament size ($D$); and a switch (*do-fitness-first*) which indicates whether the qualifiers select on fitness and the final selects on size or the other way around. We have set *do-fitness-first* to $true$. Since values of $D$ as small as 2 put too much pressure on parsimony (Luke and Panait, 2006), $D$ is permitted to hold real values in the range $[1, 2]$, so that the smaller individual wins with a probability $D/2$. We have set $D$ to 1.4. Therefore, two individuals are selected using tournament selection of size 7 based on fitness. Then the two individuals undergo selection based on size: the smaller individual wins with a probability 0.7. The reason for this choice of parameters is that this setting proved to work well on a variety of problems in Luke and Panait (2006).

In proportional tournament, the selection algorithm selects an individual using tournament selection with size 7. However, a proportion $P$ of the tournaments is based on size rather than fitness. If $P$ is set to 0, the method is equivalent to standard tournament selection, but if the value of $P$ is increased, the emphasis on size increases and, consequently, the emphasis on fitness decreases. As in the previous bloat control method, the choice of $P = 0.2$ was motivated by the results reported by Luke and Panait (2006).

The tarpeian method was introduced by Poli (2003). It limits the size of the population by making uncompetitive a fraction $W$ of individuals with above average size. Before the evaluation process, those individuals are assigned a very low fitness, which dramatically reduces their possibilitiy of being selected. Since this happens before the evaluation, it reduces the number of evaluations necessary. We have fixed $W$ to 0.3 because this setting performed well across all problem domains in Luke and Panait (2006).

Lexicographic parsimony pressure (Luke and Panait, 2006) treats fitness as the first objective of the optimization and size as a secondary objective. In plain lexicographic parsimony pressure, an individual is considered superior to another if it is better in fitness; if they have the same fitness, then the smallest individual is considered superior. We have used a variation of the method especially suited for environments where few individuals have the same fitness: lexicographic parsimony pressure with ratio bucketing. Here the individuals from the population are sorted into ranked buckets and those individuals in the same bucket are treated as if they had the same fitness. The buckets are proportioned so that low-fitness individuals are placed into larger buckets than high-fitness individuals. A parameter $r$ fixes the size of the buckets. The bottom $1/r$ fraction of individuals of the population are placed in the bottom bucket. Of the remaining individuals, the bottom $1/r$ fraction are placed into the next bucket, and so

on, until all individuals have been placed in a bucket. Again, our choice of $r$ equal to 2 was motivated by the good results across all problem domains that this setting obtained in Luke and Panait (2006).

### 3.2 Description of Test Problems

The methods have been tested in four classical problems in the GP literature (Koza, 1992; Luke, 2000b): artificial ant, 11-bit Boolean multiplexer, 5-bit even parity, and symbolic regression. The four problem domains were also proposed in Luke and Panait (2006).

The artificial ant problem is to find a strategy that can successfully navigate an artificial ant along a trail on a grid. Food pellets are spread along the path to guide the ant. In this case, the trail to follow is the Santa Fe trail, which contains 89 food pellets. The terminal set used consists of three operations: *move*, *right*, and *left*, to move the ant forward, turn to the right, or turn to the left. The function set consists of three functions: *if food ahead* looks into the square the ant is currently facing and then executes one of its two arguments depending upon whether that square contains food or is empty, *prog2* and *prog3* take two and three arguments, respectively. Those arguments are executed in sequence. The trail must be navigated in 400 time steps and the fitness of a solution is given by the number of remaining pellets. That is, the fitness can take integer values between 0 (the optimum) and 89.

Boolean multiplexer problems are a class of addressing problems. Instances have $n$ address bits followed by $2^n$ data bits. The address bits index a bit in the data bits, and the output of the multiplexer is the value of that bit. The 11-bit multiplexer has 11-bit instances, 3 address bits plus $2^3$ data bits. The choice of the terminal set is quite straightforward. It includes all the address and data bits, that is, $A_0$, $A_1$, $A_2$, $D_0$, $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, and $D_7$. The function set is formed by several Boolean operators: *and*, *or*, *nor*, and *if*. The individuals are evaluated over all possible fitness cases, that is $2^{11}(= 2{,}048)$ cases and fitness is calculated as the number of wrong predictions. Therefore, for the 11-bit multiplexer, individuals have fitness values that are integers ranging from 0 (the optimum) to 2,048.

The 5-bit even parity problem takes as an input a random Boolean instance of five elements and returns as an output whether there is an even number of 1s. The terminal set consists of all five Boolean inputs, that is, $D_0$, $D_1$, $D_2$, $D_3$, and $D_4$, and the function set consists of four Boolean operators: *and*, *or*, *nor*, and *nand*. Again, the individuals are evaluated along the $2^5$ possible fitness cases and the fitness to minimize is given by the number of wrong predictions. Fitness will be an integer value ranging from 0 (the optimum) to 32.

The symbolic regression problem implemented attempts to find a function that fits the quartic polynomial $x^4 + x^3 + x^2 + x$ within the domain $[-1, 1]$. Ephemeral random constants are not used. The terminal set includes the $x$ value and the function set consists of eight real functions: addition, subtraction, multiplication, division, sine, cosine, exponential, and logarithm. The division and logarithm functions are protected. The division returns a fixed 1 value when the denominator is very close to 0. The logarithm returns a 0 value when the argument is very close to 0. Fitness is calculated as the summation of the absolute value of the difference between the output generated by the quartic function and the output generated by the individual under evaluation at each of the training points. The training points are 20 real values generated randomly for each run in the interval $[-1, 1]$. Therefore, in this case, fitness is an unbounded real value.

Further description of these problems can be found elsewhere (Koza, 1992; Luke, 2000b; Luke and Panait, 2006). The implementation of the problems and bloat control methods is available in the ECJ library.

### 3.3 Experimental Setup and Measures

Each experiment was run 30 times on a 2.8 GHz Celeron CPU with 1 GB RAM. The statistical significance of the results was determined by ANOVA tests at 95% confidence when comparing fitness and execution time and at 99.995% confidence when comparing size, following the guidelines presented by Luke and Panait (2006). The reason for this choice was to ensure we detected any deterioration in fitness, while the mean tree size should be much smaller to be considered significantly better. The performance of a given method is considered acceptable if it equals or improves the best fitness of the benchmark (with 95% confidence) and it reduces the mean tree size of the final generation (with 99.995% confidence) within an execution time at least as short as the benchmark.

In addition, the evolution of mean tree size and diversity by generation for each bloat control method and for each problem was considered. We aim to prove that those methods that reduce bloat while maintaining diversity work better than the others. We studied the evolution of genotypic and phenotypic diversity along the run and also the correlation between diversity, size, and fitness in the final generation. Making the distinction between genotypic and phenotypic diversity is important because they are not always correlated (Burke et al., 2004). In fact, bloat is an example of how variations in the genotype may not affect the phenotype. Entropy was chosen as the diversity measure because it provides information not only about how many different phenotypes/genotypes are in the population but also how the population is distributed over them.

Phenotypic entropy (Burke et al., 2004; Tomassini et al., 2003) is used as a measure of phenotypic diversity and is calculated as in Rosca (1995):

$$-\sum_{k} p_k \ln p_k \tag{1}$$

where $p_k$ is the proportion of the population occupied by population partition $k$ and a partition is assumed to be each possible fitness value. For problems with continuous fitness functions or those with discrete fitness functions that can take many possible values, a partition can be defined that includes a subset of fitness values. In our case, this was not necessary. Three of the problems under consideration have discrete fitness functions that can take 90 values in the artificial ant case, 2,049 values in the multiplexer problem, and 33 values in the parity problem. The symbolic regression problem uses a continuous fitness function with an unbounded number of possible fitness values but, in practice, only a very small subset of values are used. High entropy values represent the situation where many individuals have a unique fitness value or few individuals share the same fitness value. On the other hand, low entropy represents the scenario where many individuals have the same fitness.

Genotypic entropy (Tomassini et al., 2003) is measured as well using Equation (1). In this case, $p_k$ is the proportion of the population that have identical genotype. High entropy values mean that most individuals have unique genotypes, while low entropy means that many individuals share the same genotype.

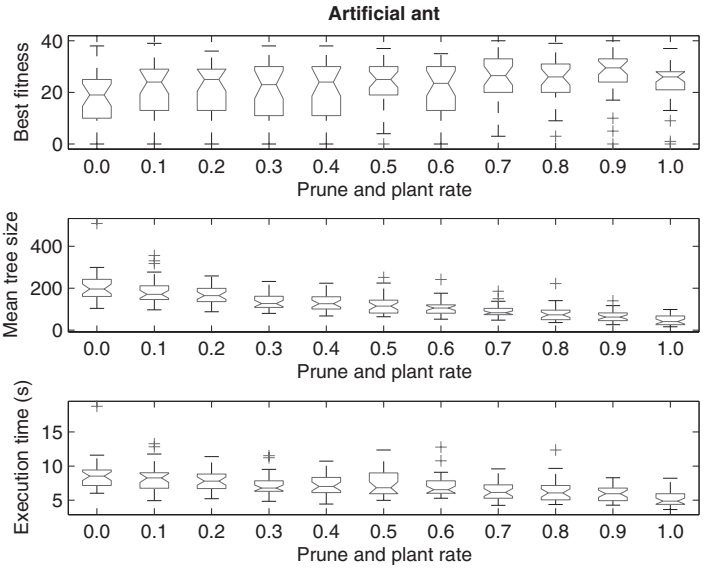


Figure 2: Boxplots of best fitness, mean tree size and execution time for all the prune and plant rates under comparison for the artificial ant problem. Lower values are better.

## 4 Setting the Prune and Plant Rate

Prior to comparing prune and plant with other bloat control methods we have analyzed different settings of the prune and plant rate. Once an offspring is created by crossover (with a probability of 0.9) or reproduction (with a probability of 0.1), prune and plant takes place with a certain probability. The choice of the probability value is what we want to assess in this section. We are looking for a prune and plant rate that maintains the fitness accuracy with a level of confidence of 95%, that reduces the size with a level of confidence of 99.995%, and improves or equals the execution time with a level of confidence of 95%. As a benchmark for comparison, we are using the executions with a prune and plant probability equal to 0. This is equivalent to using as a bloat control method depth limiting alone. We tested probability values from 0.1 to 1 with 0.1 increment steps. They were tested in the four problem domains. ANOVA tests were performed to the set of results obtained with the various settings of prune and plant for the four problem domains. Figures 2–5 show the box plots of the results.

For the artificial ant problem, all the settings of the prune and plant rate between 0.3 and 0.8 (inclusive) meet the objectives of maintaining the fitness performance, significantly reducing the average size of trees and maintaining the execution time. In addition, prune and plant rates over 0.3 significantly reduce the execution time.

For the 11-bit Boolean multiplexer, the prune and plant settings that meet our three requirements are all those in the range from 0.3 to 0.7 (inclusive). All prune and plant rates over 0.2 significantly reduce the execution times.

In the 5-bit parity case, there are only two settings of the prune and plant rate that achieved a competitive performance, those are the probabilities of 0.4 and 0.5. All prune and plant rates over 0.4 significantly reduce the execution times.

Finally, for the symbolic regression problem, prune and plant achieved the best results. All the settings between 0.4 and 1.0 have similar fitness (with 95% confidence)

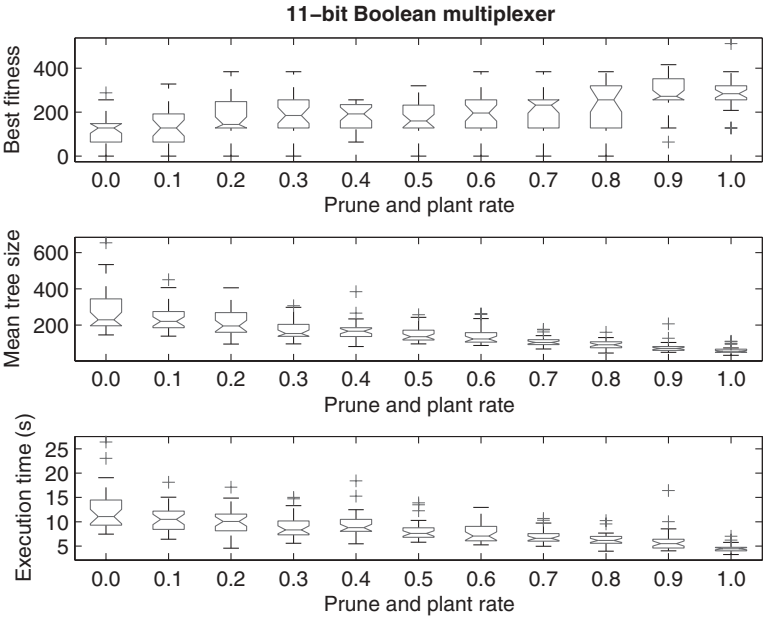


Figure 3: Boxplots of best fitness, mean tree size, and execution time for all the prune and plant rates under comparison for the 11-bit Boolean multiplexer problem. Lower values are better.

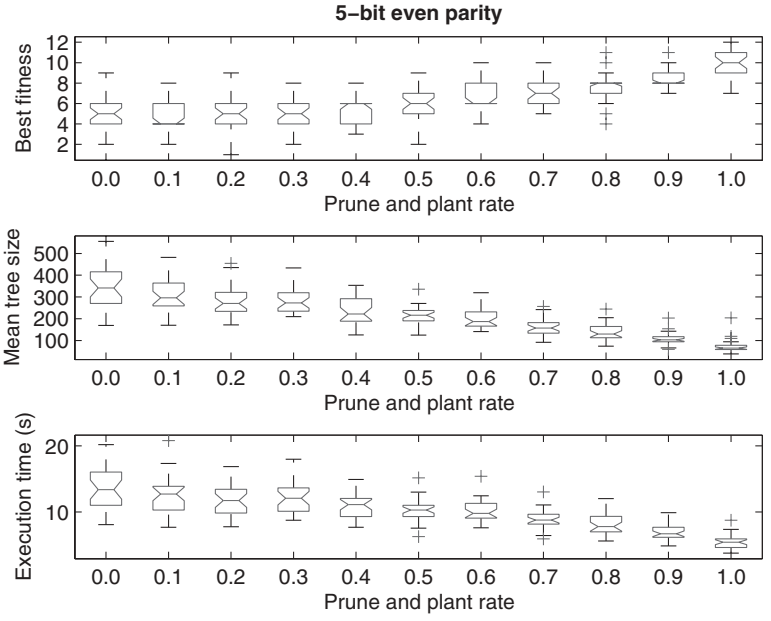


Figure 4: Boxplots of best fitness, mean tree size, and execution time for all the prune and plant rates under comparison for the 5-bit even parity problem. Lower values are better.

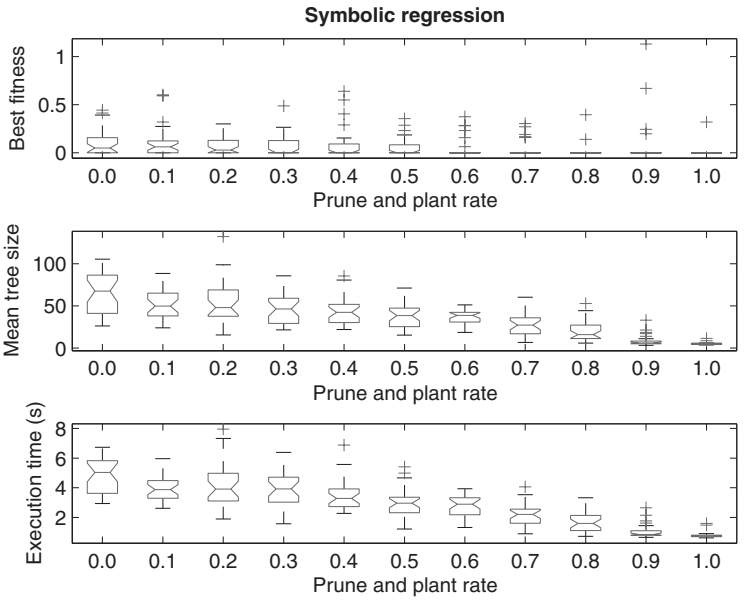


Figure 5: Boxplots of best fitness, mean tree size and execution time for all the prune and plant rates under comparison for the symbolic regression problem. Lower values are better.

as compared to depth limiting and reduce significantly the tree size (with 99.995% confidence). In addition, the settings between 0.7 and 1.0 significantly improve the fitness results obtained with depth limiting. Regarding the execution times, all the settings with a probability of prune and plant equal to or higher than 0.3 have completed their runs in a significantly shorter time as compared with the benchmark. In fact, if the prune and plant rate is set to 1, the execution time is reduced, on average, in a 83.3% with respect to depth limiting.

Thus, summarizing, there are two settings of prune and plant that succeed in reducing significantly the average tree size while maintaining the quality of the fitness and the execution time in all four problem domains, namely 0.4 and 0.5. Also, both settings reduce significantly the execution time for the artificial ant, the multiplexer and the symbolic regression problems, and the probability of prune and plant of 0.5 reduces significantly the execution time for the parity problem too. Thus, for the remaining of the paper we will use a prune and plant rate of 0.5 unless otherwise indicated.

## 5 Bloat Control Methods Comparison Study Experimental Results

This section presents the results obtained in the comparison study of the bloat control methods presented in Section 3.1 and prune and plant. The average numerical results of best fitness in the final generation, mean tree size of the final population, and execution time are shown for each bloat control method and for each problem. Also, the evolution of mean tree size and entropy by generation for each bloat control method and for each problem were considered. All comparative statements are supported by statistical evidence, unless otherwise stated.

Table 1: Summary of results. In all columns, lower values are better. Those methods that have achieved an acceptable performance for a particular problem (i.e., they significantly reduce the average size of the population in the final generation—with a confidence level of 99.995%—while maintaining the performance of the benchmark fitnesswise, with a confidence level of 95%) have been marked with a ✓ on the right-hand side.

| | Best fitness | Mean tree size | Execution time (s) | |
|---|---|---|---|---|
| Artificial ant | | | | |
| Depth limiting | $18.3333 \pm 9.2039$ | $208.5002 \pm 75.7427$ | $8.6769 \pm 2.3537$ | |
| Prune and plant | $22.8333 \pm 10.3327$ | $125.7326 \pm 48.1247$ | $7.4798 \pm 1.9244$ | ✓ |
| Double tournament | $17.5667 \pm 10.8300$ | $79.9338 \pm 23.9214$ | $5.3811 \pm 1.0913$ | ✓ |
| Proportional tournament | $21.0333 \pm 11.5892$ | $128.6386 \pm 47.7301$ | $6.7294 \pm 0.4256$ | ✓ |
| Lexicographic | $12.7333 \pm 10.4879$ | $43.5439 \pm 23.3367$ | $4.7192 \pm 1.0394$ | ✓ |
| Tarpeian | $17.8000 \pm 12.2795$ | $104.9418 \pm 28.0234$ | $5.6361 \pm 1.1589$ | ✓ |
| 11-Bit Boolean multiplexer | | | | |
| Depth limiting | $127.3000 \pm 76.3473$ | $282.2122 \pm 119.0371$ | $12.3829 \pm 4.5304$ | |
| Prune and plant | $168.1333 \pm 92.0599$ | $147.7298 \pm 42.0592$ | $8.2032 \pm 2.0218$ | ✓ |
| Double tournament | $160.6667 \pm 93.4375$ | $101.4277 \pm 40.0012$ | $6.4401 \pm 1.6374$ | ✓ |
| Proportional tournament | $187.7333 \pm 78.2916$ | $154.9070 \pm 55.3088$ | $8.1769 \pm 1.9746$ | ✓ |
| Lexicographic | $138.9333 \pm 100.7390$ | $101.4045 \pm 100.4082$ | $8.1084 \pm 4.0726$ | ✓ |
| Tarpeian | $126.9333 \pm 73.3734$ | $152.9906 \pm 56.8558$ | $7.1786 \pm 1.5608$ | ✓ |
| 5-Bit even parity | | | | |
| Depth limiting | $5.2333 \pm 1.6955$ | $344.0059 \pm 98.7059$ | $13.6728 \pm 3.5352$ | |
| Prune and plant | $5.7000 \pm 1.8033$ | $212.4720 \pm 42.2468$ | $10.2409 \pm 1.7919$ | ✓ |
| Double tournament | $4.7667 \pm 1.6955$ | $176.1017 \pm 53.1801$ | $9.0613 \pm 2.2732$ | ✓ |
| Proportional tournament | $5.2000 \pm 1.6692$ | $262.4602 \pm 70.3482$ | $11.1541 \pm 2.5844$ | |
| Lexicographic | $4.1333 \pm 1.6761$ | $159.5602 \pm 54.7550$ | $9.6802 \pm 2.3630$ | ✓ |
| Tarpeian | $5.1000 \pm 1.6049$ | $227.2182 \pm 77.9377$ | $9.6104 \pm 2.1868$ | ✓ |
| Symbolic regression | | | | |
| Depth limiting | $0.1041 \pm 0.1362$ | $66.0143 \pm 24.9509$ | $4.8186 \pm 1.2170$ | |
| Prune and plant ($p = 0.5$) | $0.0529 \pm 0.0964$ | $39.5036 \pm 15.1435$ | $2.9897 \pm 0.9622$ | ✓ |
| Prune and plant ($p = 1$) | $0.0107 \pm 0.0584$ | $5.2327 \pm 1.5062$ | $0.8046 \pm 0.2108$ | ✓ |
| Double tournament | $0.2416 \pm 0.2903$ | $21.1966 \pm 13.0905$ | $2.2251 \pm 0.9195$ | |
| Proportional tournament | $0.1051 \pm 0.1688$ | $48.4189 \pm 16.4794$ | $3.8127 \pm 0.9391$ | |
| Lexicographic | $0.0553 \pm 0.0900$ | $42.1130 \pm 28.4999$ | $3.8723 \pm 1.3088$ | ✓ |
| Tarpeian | $0.1666 \pm 0.2449$ | $37.4765 \pm 14.6797$ | $2.7251 \pm 0.9252$ | ✓ |

### 5.1 Analysis of Fitness and Size Results in the Final Generation

As mentioned previously, the performance of a method is considered acceptable if it significantly reduces the average size of the population in the final generation (with a confidence level of 99.995%) and it maintains the performance of the benchmark fitness-wise (with a confidence level of 95%). Table 1 summarizes the average results of 30 runs for each method and problem. Those methods that have achieved an acceptable performance for a particular problem have been marked with a ✓ on the right-hand side.

When solving the artificial ant problem, all the methods under comparison have acceptable performance as compared to depth limiting alone. The lexicographic method improves the fitness obtained with depth limiting, prune and plant, and proportional tournament, and it achieves a size reduction greater than any other method. Regarding the execution times, all the bloat control methods finish in shorter times than depth

limiting. Double tournament, lexicographic, and tarpeian's execution times are significantly shorter than the time needed by proportional tournament and prune and plant.

For the 11-bit Boolean multiplexer problem, all five methods achieve an acceptable performance as compared to depth limiting alone. All the methods finish in a significantly shorter time than the benchmark, but there are no significant differences when the methods are compared to each other.

For the 5-bit even parity problem, all the methods under comparison achieve an acceptable performance except proportional tournament, which fails to significantly reduce the tree size. The lexicographic method improves the fitness obtained by prune and plant. Regarding the execution time, all methods perform significantly better than depth limiting.

In the symbolic regression problem, double tournament results show a very good reduction of the average tree size but they fail to keep the fitness level. On the other hand, proportional tournament achieves good performance from the fitness point of view but fails to achieve a significant size reduction. The remaining three methods, prune and plant, lexicographic, and tarpeian, are equivalent fitnesswise and sizewise. The execution time of the lexicographic method is significantly shorter than those of tarpeian and prune and plant.

Prune and plant has proven to be very well suited for the symbolic regression problem, especially when it is used at a high rate. Table 1 includes the results obtained with prune and plant with probability 1 for the symbolic regression problem. The average fitness value obtained with this setting of prune and plant is more than five times smaller than the one obtained with lexicographic (second best). The size reduction obtained with prune and plant is four times bigger than the one achieved by double selection (second best) and the execution time is nearly three times shorter than the time spent by double tournament (second best). We must remember that the size and time achievements of double tournament were at the expense of a very poor fitness.

### 5.2 Analysis of Size Evolution

Figure 6 shows the evolution of the average tree size by generation for all bloat control methods. For the artificial ant problem, the plot illustrates the previous conclusions: lexicographic and double tournament achieve the best size reduction, while prune and plant and proportional tournament, although clearly improving on the depth limiting size control, fail to beat the performance of the other methods. It can be seen that all methods (but lexicographic) follow the same trend: there is an initial sudden increase of the average size, followed by a valley and then the average tree size starts increasing steadily. For the lexicographic method, on the contrary, after the initial peak, the average size decreases slowly until it stabilizes.

In the 11-bit Boolean multiplexer, all the methods initially show a slight reduction on the average tree size and then it starts to increase, at a very high rate in the depth limiting case and more slowly for the other methods. The curve of the lexicographic method has a peculiar behavior because at the initial 20 generations, the tree size grows faster than for any other method, but then it starts to fall to the level of the double tournament.

In the 5-bit even parity problem, all the methods show a steady increase of the average tree size by generation except the lexicographic method, whose individuals grow fast in the first half of the run (e.g., faster than prune and plant and double tournament) and then stabilize at the same size level until the final generation.

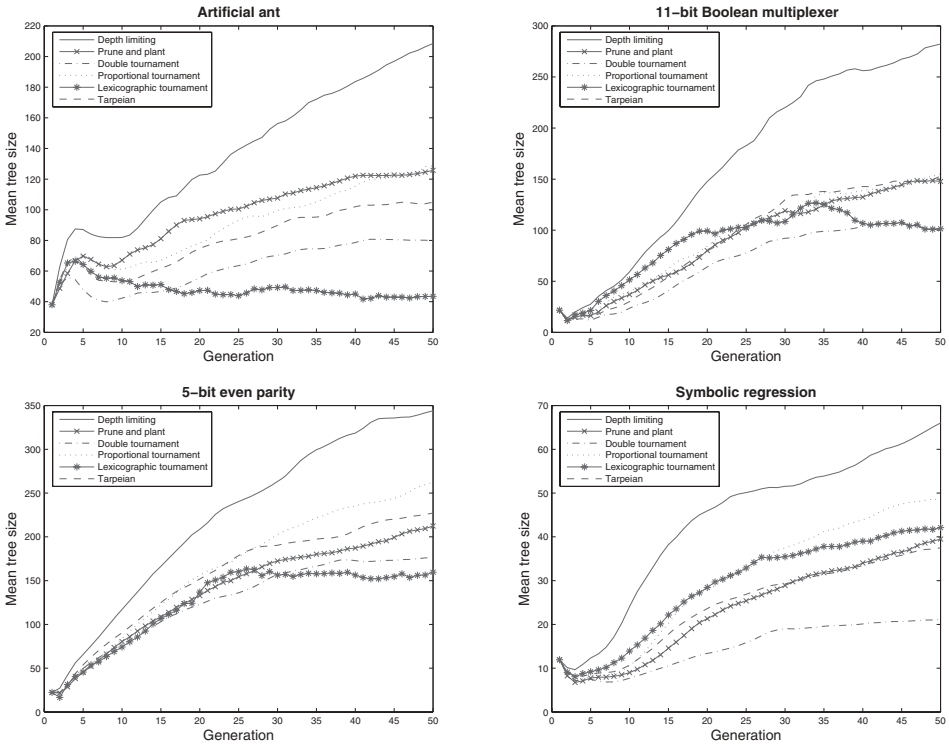


Figure 6: Evolution of mean tree size by generation for all the bloat control methods. In general, depth limiting barely controls bloat, while lexicographic tournament puts too high a pressure on size.

All methods show the same trend for the symbolic regression problem: an initial sudden reduction of size followed by a steady rise after the initial generations. Double tournament shows an extremely good reduction in size as can be seen in the figure but at the expense of a deterioration in fitness.

In general, the dynamic behavior of all methods is quite similar, modifying size in the same way, which leads us to think that the underlying dynamics depend more on the problem itself than on the bloat control operators; however, there is a difference in the width of the variation. As a consequence, some methods have a greater effect on bloat than others.

### 5.3 Analysis of Diversity

In this section, the evolution of the phenotypic and genotypic entropy by generation is shown for all bloat control methods for the four benchmark problems. Also, to be able to draw conclusions about the relationships between diversity, size, and fitness, the correlation between both entropies, best fitness, and mean size in the final generation has been calculated using the Spearman correlation coefficient:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n^3 - n} \tag{2}$$

where $d_i$ is the difference between the ranks of corresponding values and $n$ is the number of values in each dataset. In Equation (2), $\rho$ can take values in the range $[-1, 1]$. A value of $\rho = 1$ indicates a strong positive correlation between two sets of values, $\rho = -1$ indicates a strong negative correlation between two sets of values, and $\rho = 0$ indicates no correlation.

Figure 7 shows the evolution of the phenotypic and genotypic entropy by generation for all bloat control methods for the artificial ant problem. It can be seen that those methods that obtained a better fitness show also a high phenotypic diversity. This is in line with the findings of Burke et al. (2004), which provide strong evidence that good fitness and high phenotypic entropy are correlated for the artificial ant problem. Prune and plant maintains a high diversity along the whole execution. The general shape of the curves follows the graphical results presented in Burke et al. (2004) for the artificial ant problem: an initial peak of entropy roughly at the same time as the code growth peak and then a gradual decrease. As in the average tree size case, the lexicographic method shows a slightly different result: at around generation 15, entropy starts rising again. The lexicographic method has the highest phenotypic entropy in the last generation followed by prune and plant and double tournament. Regarding the genotypic entropy, the general trend most methods follow is that, after a sudden fall of entropy in the first two generations, the entropy value stabilizes. The curve that represents the performance of prune and plant follows this trend, but the initial drop of entropy is sharper than for any other method. The lexicographic method behaves in a different way. After the initial stabilization it starts a slow descent until it reaches the entropy levels of prune and plant. Both methods converge to the lowest genotypic entropy levels of all.

The graphical results of the evolution of entropy for the artificial ant problem agree with the conclusion in Burke et al. (2004) that good fitness correlates with high phenotypic entropy and low genotypic diversity for the artificial ant problem. This conclusion seems a bit contradictory, since one would assume that a high phenotypic entropy would be linked to a high genotypic entropy. However, the results shown here as well as those presented in Burke et al. (2004) provide strong evidence that phenotypic diversity is important for achieving good performance and it is not incompatible with structural convergence. In fact, the opposite scenario, high genotypic entropy linked to a low phenotypic entropy, could be an indicator of a bloated population: many individuals of various sizes sharing the same fitness value. The lexicographic method obtains the best results and meets both requirements: high phenotypic entropy and low genotypic diversity. On the other hand, depth limiting alone shows, at the end of the run, a low phenotypic entropy and a high genotypic entropy. Prune and plant achieves in the very early generations a low level of genotypic entropy, but this is not accompanied by high phenotypic entropy. This could mean that it is a case of premature structural convergence and it would explain why the lexicographic method obtained better results for this particular problem.

Figure 8 shows the evolution of the phenotypic entropy by generation for all bloat control methods for the 11-bit Boolean multiplexer problem. The lexicographic method and prune and plant have the highest levels of phenotypic entropy along the second half of the run. On the other hand, double tournament falls to very low levels of phenotypic entropy very early in the run. The curves for all methods are similar: a very deep descent of the phenotypic entropy level in the first generations, followed by a quick increase until generation 10, and a slow descent from then onward. Looking at the genotypic entropy results, the genotypic entropy levels drop during the initial five generations

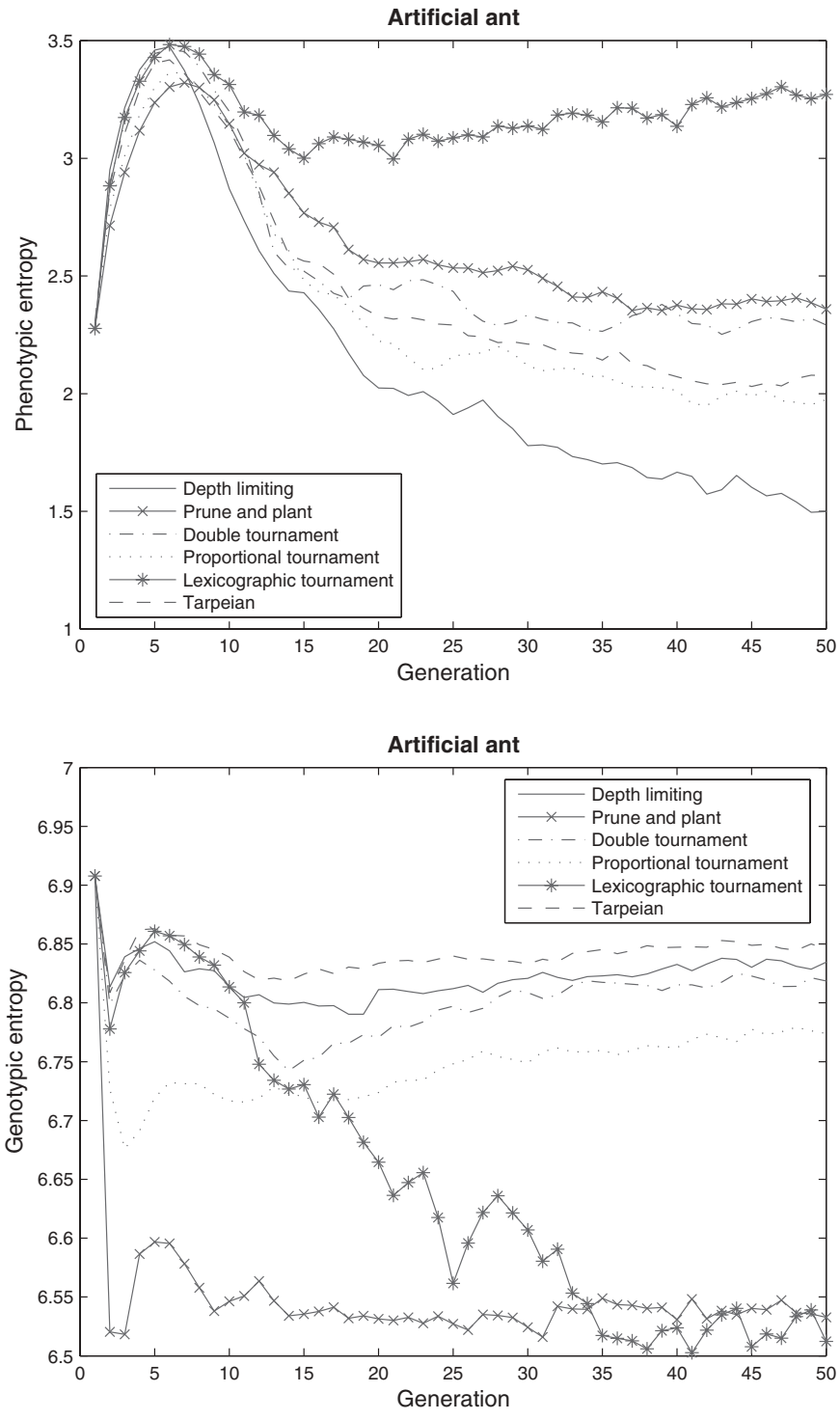


Figure 7: Evolution of phenotypic entropy (top) and the genotypic entropy (bottom) by generation for all the bloat control methods for the artificial ant problem.

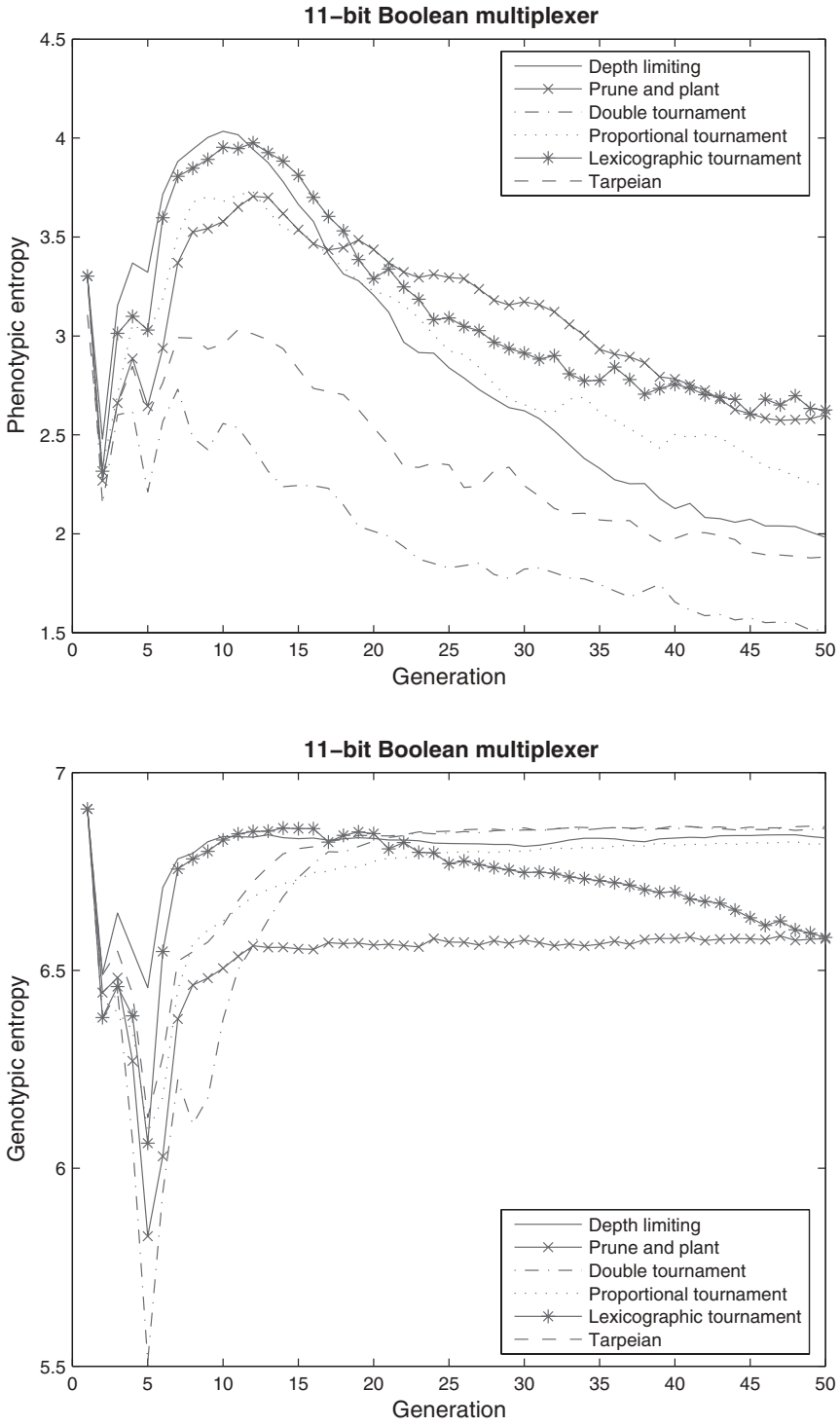


Figure 8: Evolution of phenotypic entropy (top) and the genotypic entropy (bottom) by generation for all the bloat control methods for the 11-bit Boolean multiplexer problem.

but they recover their initial value in the next generations and stabilize around it. Prune and plant and the lexicographic method are exceptions. Prune and plant stabilizes at a lower entropy level than any other method. The entropy level of the lexicographic method starts falling again after 20 generations and it reaches the same level as prune and plant at the end of the run.

Figure 9 shows the evolution of the phenotypic and genotypic entropy by generation for all bloat control methods for the 5-bit even parity problem. The plot shows a fast increase of the phenotypic entropy level in the initial generations and then it stabilizes for prune and plant, tarpeian, double tournament, and proportional tournament. In the case of the lexicographic method, the phenotypic entropy level starts rising again at around generation 15, and in the case of depth limiting, it starts falling at the same time. This is in contrast to the other problems where the phenotypic entropy levels do not decrease by generation (except for depth limiting). This is probably due to the fact that the 5-bit even parity problem is a difficult problem and in 50 generations convergence has not taken place yet. The genotypic entropy of all methods but prune and plant behave in a very similar way: the entropy levels fall very quickly in the first couple of generations and then stabilizes at a value close to the initial one. Prune and plant, on the other hand, stabilizes at a smaller entropy value than the rest, but note that the Y-scale range is smaller than for any other problem. The differences among methods are not as big as they seem in the plot.

Figure 10 shows the evolution of the phenotypic and genotypic entropy by generation for all bloat control methods for symbolic regression. We have included the results of prune and plant with a probability of 1 given the good results it achieved for this particular problem. The general shape of the phenotypic entropy curve follows the graphical results presented by Burke et al. (2004) for the symbolic regression problem: an initial descent of phenotypic entropy followed by a slower rise of the phenotypic entropy level and a steady decrease. All methods show the same behavior but at a different pace. As for the genotypic entropy, it shows an initial descent of entropy followed by a slower rise of the entropy level and a steady decrease.

Table 2 presents the $\rho$ values obtained in the correlation study. Those marked with a ✓symbol prove a significant correlation with a level of confidence of 95%. Thus, for the artificial ant problem, there is a positive correlation between genotypic entropy and size (the higher the entropy, the bigger the size) and a negative correlation between phenotypic entropy, fitness, and size, that is, the higher the phenotypic entropy, the better the fitness (lower values are better) and the smaller the size. There is not evidence of correlation between genotypic entropy and fitness, although Burke et al. (2004) reported a positive correlation between structural convergence and fitness. Summarizing, for the artificial ant problem, good fitness values and small tree sizes are associated with phenotypic diversity and structural convergence.

The correlation analysis results for the 11-bit Boolean multiplexer indicate a negative correlation between genotypic entropy and fitness (the higher the genotypic entropy, the better the fitness), and a positive correlation between size and phenotypic entropy (the highest the phenotypic entropy, the bigger the trees). The correlation coefficient between genotypic entropy and size indicates that there is a positive correlation between them (the higher the genotypic entropy, the bigger the tree size) which is significant with a confidence level of 90%. Phenotypic entropy and fitness are uncorrelated. Therefore, the conclusions for the 11-bit Boolean multiplexer problem are quite different than for the artificial ant (note that for the multiplexer problem the correlation coefficients are smaller in absolute value, which indicates less confidence in the results): phenotypic

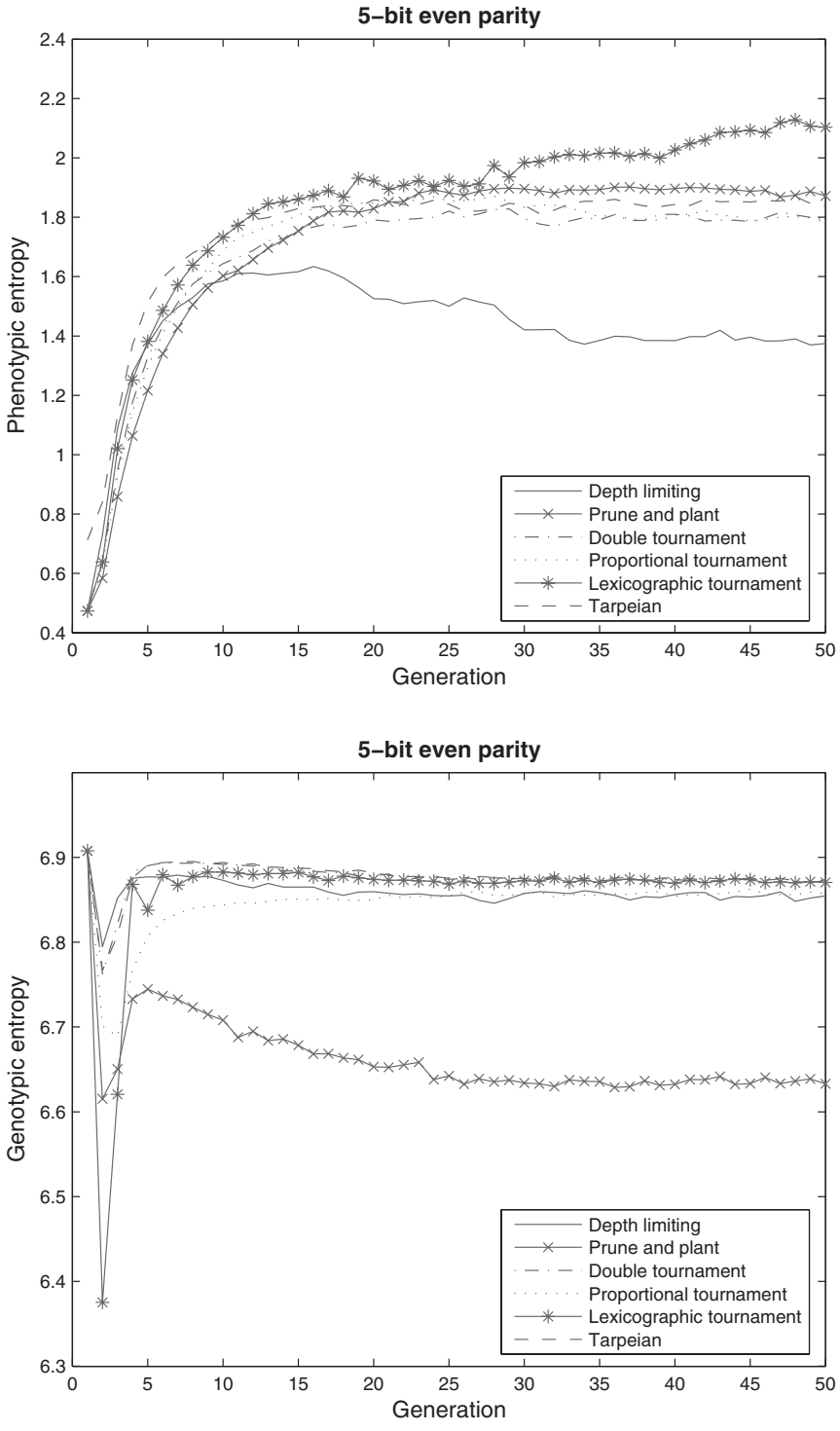


Figure 9: Evolution of phenotypic entropy (top) and the genotypic entropy (bottom) by generation for all the bloat control methods for the 5-bit even parity problem.

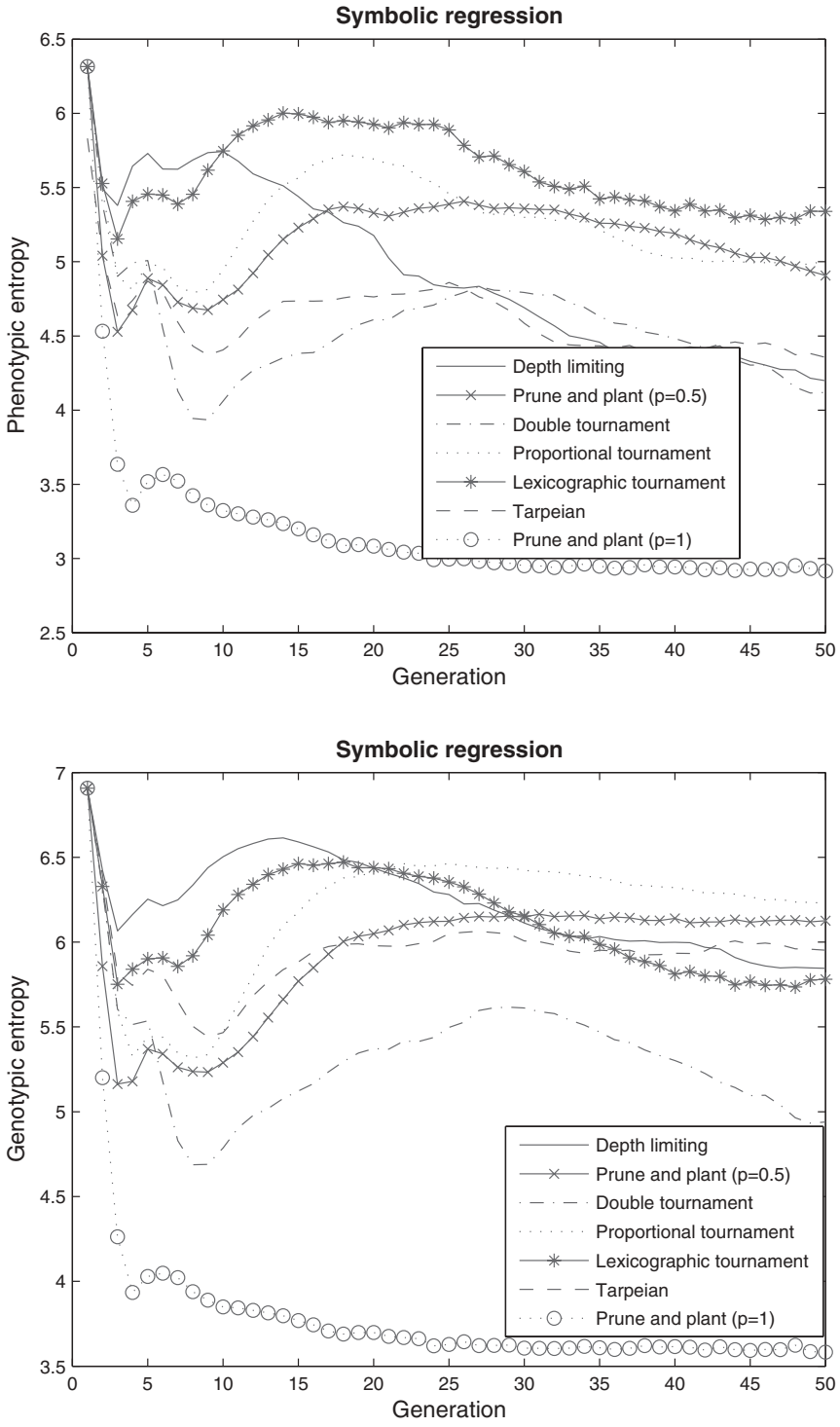


Figure 10: Evolution of phenotypic entropy (top) and the genotypic entropy (bottom) by generation for all the bloat control methods for the symbolic regression problem.

Table 2: $\rho$ values of correlation between genotypic/phenotypic entropy on one hand, and size and fitness on the other. A check indicates an observable effect.

| | | Phenotypic entropy | | Genotypic entropy | |
|---|---|---|---|---|---|
| Artificial ant | | | | | |
| | Fitness | −0.3178 | ✓ | −0.0827 | |
| | Size | −0.5338 | ✓ | 0.2767 | ✓ |
| 11-bit Boolean multiplexer | | | | | |
| | Fitness | 0.0207 | | −0.2330 | ✓ |
| | Size | 0.1769 | ✓ | 0.1300 | |
| 5-bit even parity | | | | | |
| | Fitness | −0.3439 | ✓ | −0.0680 | |
| | Size | −0.5370 | ✓ | −0.1910 | ✓ |
| Symbolic regression | | | | | |
| | Fitness | 0.0382 | | −0.1151 | |
| | Size | 0.1747 | ✓ | 0.6097 | ✓ |

diversity promotes bigger trees and structural convergence hampers the convergence to good fitness values but seems to reduce the tree size. Thus, the utility of high phenotypic entropy levels together with structural convergence for this problem is disputable. In fact, double tournament obtains very good results having a very low phenotypic entropy and a high genotypic entropy, although the lexicographic method and prune and plant obtain statistically equivalent results with high phenotypic entropy levels and structural convergence.

Regarding the correlation between diversity, size, and fitness for the 5-bit even parity problem, the results show a very strong negative correlation between phenotypic entropy, fitness, and size, that is, the higher the phenotypic entropy, the better the fitness and the smaller the size. This confirms the conclusion presented in Burke et al. (2004) that for the parity problem fitness correlates with high phenotypic diversity. Figure 9 (top) shows that lexicographic has the highest level of phenotypic entropy while depth limiting alone has the lowest, as one would expect from the correlation results. On the other hand, genotypic entropy has a negative correlation with size (the lower the genotypic entropy, the bigger the tree size).

When considering the correlation between the diversity measures, size, and fitness for the symbolic regression problem, there are significant positive correlations between genotypic/phenotypic entropy and size (the lower the genotypic/phenotypic entropy, the smaller the tree size). Phenotypic entropy and fitness are uncorrelated according to our results. This was unexpected and contradicts Burke et al.'s (2004) findings that for symbolic regression, fitness and phenotypic diversity are positively correlated. In fact, in Figure 10 (top) it can be seen that prune and plant with probability 1 (the method that achieved the best results) converges to a very low level of phenotypic entropy, which is consistent with the conclusions reported in Burke et al. (2004).

## 6 Discussion

The study of the diversity levels has provided some insight as to which bloat control method should be used for each problem. For some problems, mainly the artificial ant and the 5-bit even parity, fitness and size are negatively correlated with high phenotypic entropy. The higher the phenotypic diversity, the better the fitness and the smaller the trees. Probably this correlation is due to the reduced number of fitness cases

(Dolin et al., 2002) in these problems (90 and 33, respectively). This means that in the initial population, there are only a small number of unique fitness values. Higher fitness diversity is necessary in order to find a good result. For those types of problems, the best suited method is lexicographic tournament selection. Lexicographic selection with ratio bucketing reduces the fitness selective pressure by considering as equal from a fitness point of view all the individuals in the same bucket. Reducing the selective pressure leads to more randomness and a higher phenotypic diversity. On the other hand, the tarpeian method forces an "artificial" low phenotypic entropy by assigning a very bad fitness value to oversized trees. This could hamper its performance in problems where good fitness is associated with high phenotypic diversity.

The lexicographic method and prune and plant have shown on some problems a performance which seems counterintuitive: the case where high phenotypic entropy comes with low genotypic entropy. Although it seems an inconsistency, this situation (high phenotypic entropy and low genotypic entropy) is the exact opposite of what bloat is. Bloat is characterized by an increase in size without a return in terms of fitness, that is, individuals with various sizes achieving the same fitness value. Thus, a bloated situation could be described as high genotypic diversity associated with low phenotypic diversity. This is precisely the behavior that the depth limiting alone method used as a benchmark (the most bloated method of all) has shown for all problems: high genotypic entropy levels and low phenotypic entropy levels.

In the symbolic regression problem, there is a positive correlation between size and genotypic entropy (the lower the genotypic entropy, the smaller the trees). Some authors (e.g., Burke et al., 2004) have also provided strong evidence that there is a positive correlation between fitness and phenotypic entropy (the lower the phenotypic entropy, the better the fitness). In this case, prune and plant obtains better results because it consistently converges to low levels of genotypic entropy and its levels of phenotypic entropy are never as high as those of the lexicographic method.

The comparatively low levels of genotypic entropy of prune and plant can be explained by the selection method. All methods use tournament selection. However, all methods but prune and plant and depth limiting alone introduce a bias in the selection process toward smaller trees. For instance, in double tournament, the final tournament selects based on size; in proportional tournament a proportion $P$ of the tournaments is based on size rather than fitness; in tarpeian a fraction $W$ of individuals with above-average size is made uncompetitive regardless of their fitness; and in the lexicographic method the selection process among individuals in the same bucket is based on size. That means that sometimes a solution is selected for crossover/reproduction over others not because it is better, but because it is smaller. This effectively reduces the selective pressure. Prune and plant does not interfere in the selection process. Therefore it has a higher selective pressure than the other methods. The higher the selective pressure, the more chances the best individuals have of being cloned several times, which leads to lower levels of genotypic entropy. Besides, prune and plant creates two offspring from a single parent, which could mean that the genetic pool (in the sense of unique subtrees) is reduced by generation, since the overall number of different subtrees is reduced. On the other hand, in prune and plant after crossover/reproduction, individuals undergo prune and plant with a certain probability. That effectively reduces the reproduction rate, accordingly increasing diversity but at a different level. The number of available subtrees does not increase, but the number of unique trees does. In prune and plant, the effective reproduction rate is: $p_{\mathrm{r}}^{\mathrm{eff}} = p_{\mathrm{r}}(1 - p_{\mathrm{p\&p}})$, where $p_{\mathrm{r}} = 0.1$ is the original reproduction rate and $p_{\mathrm{p\&p}}$ is the probability of prune and plant. In the case of

$p_{p\&p} = 0.5$, the effective reproduction rate is $p_r^{eff} = 0.05$. This latter effect may counterbalance the high selective pressure and the loss of genetic material of the method and allows for good levels of phenotypic diversity.

## 7 Conclusions and Further Work

Prune and plant has demonstrated its adequacy in a wide spectrum of problems and in comparison with some of the bloat control methods that were identified as best in an extensive review in the topic. For the four problems under consideration, there has been at least one setting of prune and plant that has maintained the quality of the results in terms of fitness while significantly reducing the mean size of the trees in the final population. Moreover, there is a setting of prune and plant that achieves good results across all problem domains, namely a 0.5 prune and plant ratio. Among all the methods compared, only prune and plant, tarpeian, and the lexicographic method have been able to obtain competitive results in all four problem domains tested; proportional tournament failed to obtain satisfactory size reductions in the parity and symbolic regression problems, and double tournament failed to maintain a good fitness performance for symbolic regression. In addition, in one of the problems (symbolic regression) prune and plant outperformed the other bloat control methods in terms of fitness, final size of the solutions, and time needed to complete the execution. The results obtained with prune and plant in the symbolic regression problem are extremely good. Prune and plant improved the best fitness obtained with any other method by more than 80%; it reduced the mean size of the final generation by more than 85%, and it achieved time reductions of over 75%. This probably hints at the fact that tree size has some influence on the effectiveness of the method; prune and plant seems to be more efficient for small tree sizes (less than 100), while the lexicographic method is better over the 100 barrier. However, this hypothesis will have to be proved with further experimentation.

The study of the evolution of diversity has allowed us to come to the conclusion that the degree and even the sign of correlation between diversity, fitness, and size is problem dependent. Therefore, it is important to analyze which kind of problem we are going to tackle before deciding on the bloat control method to use. The discussion presented here can provide some clues for making this decision. If for our particular problem fitness and size are negatively correlated with high phenotypic entropy, then probably the best suited method would be lexicographic tournament selection with ratio bucketing. If the problem shows a positive correlation between size and genotypic entropy (the lower the genotypic entropy, the smaller the trees) and a positive correlation between fitness and phenotypic entropy (the lower the phenotypic entropy, the better the fitness), then prune and plant is a good option.

For our future work, we plan to look at other methods for measuring diversity, especially genotypic diversity, such as edit distances. We believe that the extra information this kind of measure provides will be very useful in our analysis of the impact of the correlation between genotypic diversity and size when controlling bloat. Also, we plan to quantify the amount of introns in the individuals to establish a relationship between genotypic and phenotypic diversity.

Other issues we want to tackle are related to how we could improve prune and plant's results in those problems where it has performed worse than the lexicographic method. Two of the ideas we are considering are to only apply prune and plant to trees over a certain size, or to test an algorithm "prune and seed" that will prune a branch

and will plant a randomly generated one of the same size, effectively reducing the loss of genotypic diversity.

Finally, we want to test the utility of a dynamic rate of prune and plant that ensures that the mean program size remains at a certain value, following the underlying idea of the covariant parsimony pressure method (Poli and McPhee, 2008).

## Acknowledgments

This work has been supported by project TIN2007-68083-C02 (Spanish Ministry of Science, NoHNES—Non-Hierarchical Network Evolutionary System Project).



## References

Alfaro-Cid, E., Cuesta-Canada, A., Sharman, K., and Esparcia-Alcázar, A. I. (2008a). Strong typing, variable reduction and bloat control for solving the bankruptcy prediction problem using genetic programming. In A. Brabazon and M. O'Neill (Eds.), *Natural computing in computational economics and finance*. *Studies in computational intelligence series*. Vol. 100. Berlin: Springer.

Alfaro-Cid, E., Esparcia-Alcázar, A. I., Sharman, K., Fernández de Vega, F., and Merelo, J. (2008b). Prune and plant: A new bloat control method for genetic programming. In *Proceedings of the 8th International Conference on Hybrid Intelligent Systems (HIS2008)*, pp. 31–35.

Alfaro-Cid, E., Sharman, K., and Esparcia-Alcázar, A. (2007). A genetic programming approach for bankruptcy prediction using a highly unbalanced database. In *Proceedings of the First European Workshop on Evolutionary Computation in Finance and Economics (EvoFIN'07)*. *Lecture notes in computer science*, Vol. 4448 (pp. 169–178). Berlin: Springer.

Angeline, P. J. (1997). Comparing subtree crossover with macromutation. In *Proceedings of the Sixth International Conference on Evolutionary Programming (EP'97)*. *Lecture notes in computer science*, Vol. 1213 (pp. 101–111). Berlin: Springer.

Araujo, L. (2004). Genetic programming for natural language parsing. In *Proceedings of the Seventh European Conference on Genetic Programming (EuroGP'04)*. *Lecture notes in computer science*, Vol. 3003 (pp. 230–239). Berlin: Springer.

Bhattacharya, M., and Nath, B. (2001). Genetic programming: A review of some concerns. In *Proceedings of the International Conference on Computational Science (ICCS'01)*. *Lecture notes in computer science*, Vol. 2074 (pp. 1031–1040). Berlin: Springer.

Burke, E. K., Gustafson, S., and Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–52.

Chellapilla, K. (1997). Evolving computer programs without using subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216.

Crane, E., and McPhee, N. (2006). The effects of size and depth limits on tree-based genetic programming. In *Genetic programming theory and practice III* (pp. 223–240). Berlin: Springer.

Crawford-Marks, R., and Spector, L. (2002). Size control via size fair genetic operators in the PushGP genetic programming system. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pp. 733–739.

da Silva, S. G. O. (2008). Controlling bloat: Individual and population based approaches in genetic programming. PhD thesis, Coimbra University, Portugal.

Dignum, S., and Poli, R. (2007). Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effect on bloat. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, Vol. 2 (pp. 1588–1595).

Dignum, S., and Poli, R. (2008). Crossover, sampling, bloat and the harmful effects of size limits. In M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcázar, I. De Falco, A. Della Cioppa, and E. Tarantino (Eds.), *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*. *Lecture notes in computer science*, Vol. 4971 (pp. 158–169). Berlin: Springer.

Dolin, B., Arenas, M., and Merelo, J. J. (2002). Opposites attract: Complementary phenotype selection for crossover in genetic programming. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*. *Lecture notes in computer science*, Vol. 2439 (pp. 142–152). Berlin: Springer.

Ekart, A. (1999). Shorter fitness preserving genetic programs. In C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, and M. Schoenauer (Eds.), *Proceedings of the 4th European Conference on Artificial Evolution (AE'99)*. *Lecture notes in computer science*, Vol. 1829 (pp. 73–83). Berlin: Springer.

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

Fernández, F., and Martín, A. (2004). Saving effort in parallel GP by means of plagues. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule (Eds.), *Proceedings of the 7th European Conference on Genetic Programming (EuroGP'04)*. *Lecture notes in computer science*, Vol. 3003 (pp. 269–278). Berlin: Springer.

Fernández de Vega, F., Rubio del Solar, M., and Fernández Martínez, A. (2005). Plantación de árboles: Una propuesta para reducir esfuerzo de cómputo en programación genética. In M. G. Arenas, F. Herrera, M. Lozano, J. J. Merelo, G. Romero, and A. M. Sánchez (Eds.), *Actas del IV Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'05)*, pp. 57–62.

Kinnear, K. E. (1993). Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, Vol. 2 (pp. 881–888). Piscataway, NJ: IEEE Press.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press.

Langdon, W. B. (2000). Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119.

Langdon, W. B., and Poli, R. (1997). Fitness causes bloat. In *Proceedings of the World Conference on Soft Computing in Engineering Design and Manufacturing*, pp. 13–22. Berlin: Springer.

Langdon, W. B., and Poli, R. (2002). *Foundations of genetic programming*. Berlin: Springer-Verlag.

Luke, S. (2000a). Code growth is not caused by introns. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO'00)*, pp. 228–235.

Luke, S. (2000b). Issues in scaling genetic programming: Breeding strategies, tree generation, and code bloat. PhD thesis, University of Maryland, College Park.

Luke, S. (2000c). Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283.

Luke, S., and Panait, L. (2002). Lexicographic parsimony pressure. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska

(Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pp. 829–836.

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

McPhee, N. F., and Miller, J. D. (1995). Accurate replication in genetic programming. In *Proceedings of the Sixth International Coference on Genetic Algorithms (ICGA'95)*, pp. 303–309.

Nordin, P., Francone, F., and Banzhaf, W. (1996). In *Advances in genetic programming 2*. Explicitly defined introns and destructive crossover in genetic programming (pp. 111–134). Cambridge, MA: The MIT Press.

Panait, L., and Luke, S. (2004). Alternative bloat control methods. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell (Eds.), *Genetic and Evolutionary Computation—GECCO-2004, Part II. Lecture notes in computer science*, Vol. 3103 (pp. 630–641). Berlin: Springer.

Poli, R. (2001). General schema theory for genetic programming with subtree-swapping crossover. In J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon (Eds.), *Proceedings of the Fourth European Conference on Genetic Programming (EuroGP'01). Lecture notes in computer science*, Vol. 2038 (pp. 143–159). Berlin: Springer.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. P. K. Tsang, R. Poli, and E. Costa (Eds.), *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP'03). Lecture notes in computer science*, Vol. 2610 (pp. 204–217). Berlin: Springer.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008a). *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

Poli, R., Langdon, W. B., and Dignum, S. (2007). On the limiting distribution of program sizes in tree-based genetic programming. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar (Eds.), *Proceedings of the 10th European Conference on Genetic Programming. Lecture notes in computer science*, Vol. 4445 (pp. 193–204). Berlin: Springer.

Poli, R., and McPhee, N. F. (2003). General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206.

Poli, R., and McPhee, N. F. (2008). Parsimony pressure made easy. In M. Keijzer, G. Antoniol, C. B. Congdon, K. Deb, B. Doerr, N. Hansen, J. H. Holmes, G. S. Hornby, D. Howard, J. Kennedy, S. Kumar, F. G. Lobo, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, and I. Wegener (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'08)* (pp. 1267–1274).

Poli, R., McPhee, N. F., and Vanneschi, L. (2008b). The impact of population size on code growth in GP: Analysis and empirical validation. In M. Keijzer, G. Antoniol, C. B. Congdon, K. Deb, B. Doerr, N. Hansen, J. H. Holmes, G. S. Hornby, D. Howard, J. Kennedy, S. Kumar, F. G. Lobo, J. F. Miller, J. Moore, F. Neumann, M. Pelikan, J. Pollack, K. Sastry, K. Stanley, A. Stoica, E.-G. Talbi, and I. Wegener (Eds.), *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 1275–1282.

Rosca, J. P. (1995). Entropy-driven adaptive representation. In J. P. Rosca (Ed.), *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pp. 23–32.

Rzhetsky, A., and Ayala, F. (1999). The enigma of intron origins. *Cellular and Molecular Life Sciences (CMLS)*, 55(1):3–6.

Soule, T. (1998). Code growth in genetic programming. PhD thesis, University of Idaho, Moscow, Idaho.

Soule, T., and Foster, J. A. (1998). Removal bias: A new cause of code growth in tree based evolutionary programming. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 781–786.

Soule, T., Foster, J. A., and Dickinson, J. (1996). Code growth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), *Proceedings of the First Annual Conference on Genetic Programming*, pp. 215–223.

Tackett, W. A. (1994). Recombination, selection and the genetic construction of genetic programs. PhD thesis, University of Southern California, Los Angeles.

Tomassini, M., Vanneschi, L., Fernández, F., and Galeano, G. (2003). Diversity in multipopulation genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO 2003). Lecture notes in computer science*, Vol. 2724 (pp. 1812–1813). Berlin: Springer.

Van Belle, T., and Ackley, D. H. (2002). Uniform subtree mutation. In *Proceedings of the Fifth European Conference on Genetic Programming (EuroGP'02). Lecture notes in computer science*, Vol. 2278 (pp. 152–161). Berlin: Springer.