Controlling Code Growth by Dynamically Shaping the Genotype Size Distribution

Marc-André Gardner Christian Gagné Marc Parizeau

marc-andre.gardner.1@ulaval.ca,
christian.gagne@gel.ulaval.ca, marc.parizeau@gel.ulaval.ca

Laboratoire de vision et systèmes numériques
Département de génie électrique et de génie informatique
Université Laval, Québec (Québec), Canada G1V 0A6

Abstract

Genetic programming is a hyperheuristic optimization approach that seeks to evolve various forms of symbolic computer programs, in order to solve a wide range of problems. However, the approach can be severely hindered by a significant computational burden and stagnation of the evolution caused by uncontrolled code growth. This paper introduces HARM-GP, a novel operator equalization method that conducts an adaptive shaping of the genotype size distribution of individuals in order to effectively control code growth. Its probabilistic nature minimizes the computational overheads on the evolutionary process while its generic formulation allows it to remain independent of both the problem and the genetic variation operators. Comparative results over twelve problems with different dynamics, and over nine other algorithms taken from the literature, show that HARM-GP is excellent at controlling code growth while maintaining good overall performance. Results also demonstrate the effectiveness of HARM-GP at limiting overfitting in real-world supervised learning problems.

1 Introduction

Genetic Programming (GP) has been successfully applied to a wide variety of problems in machine learning, engineering, physics, biology, and medical research [30]. The flexibility arising from a variable length assembly of user-defined primitives makes it a versatile and easy to use hyperheuristic that, over time, can dynamically adapt the structural complexity of its solutions to specific problem instances. However, this flexibility comes at a price that can hinder the applicability of GP, especially for real-world problems where fitness evaluation is often computationally intensive.

A particular issue is the so-called *bloat* phenomenon, where genotypes tend to systematically increase in size over time without a corresponding improvement in fitness. This behavior has been observed from the very beginning of GP [29], has been the subject of various studies [6, 13, 32, 37, 64], and has led to several approaches [1, 8, 18, 35, 46, 55, 56, 59, 72] to avoid this problem. Bloat resistant data structures were even proposed [40] to circumvent the issue. Nonetheless, bloat remains a problem that restricts the application of GP in various fields, including classification [17], heuristics search [12], genes identification and analysis [16], and several others [23].

Specifically, bloat is usually defined as "a program growth without *significant* return in terms of fitness" [49, 70]. We emphasize the word significant, because it is important to note that bloat can occur in a population even if there is a fitness improvement. From this general definition, we can now define a *bloated population* as a population in which the individuals grow without significant return in terms of fitness. The same way, we generalize this at the individual level by characterizing an individual as bloated when it contains many parts which do not *significantly* affect its semantic – in other terms, its output is not significantly modified given an input, and so is its fitness.

Genotype growth by itself can become problematic as it increases the computational effort required to allow GP to reach convergence. Again, some solutions have been proposed to address this issue, such as surrogate models and introns removal [11], but the problem roots remain: how to permit code growth while keeping it under control in a generic fashion.

Another less explored aspect of GP is its generalization ability. It has not received much attention until recently [4, 68], and remains an open issue in general [42], and for supervised learning applications in particular.

In this paper, we present a new code growth control method named *Histogram-based Accept-Reject Method for GP* (HARM-GP). It is part of the broader operator equalization (OpEq) family [15], but innovates in its use of an accept-reject method to control the genotype size distribution of a population. Its design is consistent with most common bloat theories, and it aims at facilitating the use of GP on practical problems. However, it does not depend on feature-specific GP and can be used on virtually any variable-length representations.

The remainder of the paper is structured as follows. First, Sec. 2 presents a review of the existing generic bloat control methods. Next, Sec. 3 provides a detailed description of HARM-GP, complete with its theoretical foundations. Exhaustive experimental results are then presented and analyzed in Sec. 4, with ten methods compared over twelve problems. Sec. 5 follows with a general discussion of results, and conclusions are drawn in Sec. 6.

2 Bloat Control Methods

This section presents a succinct review of some common bloat control methods. For a comprehensive review, the reader is referred to [2, 37, 57]. We restrained ourselves to the current generic bloat control methods. For instance, algebraic

simplification [76] or numerical simplification [27] can be quite useful in a regression context, but cannot be applied to a planning problem. In the same way, one could use different well-known algorithms to simplify boolean expressions, but these algorithms are not suitable for classification or regression problems. While we do not dispute the usefulness of these methods in specific contexts, they do not constitute a generic solution and have therefore not been included in this review.

2.1 Static Depth Limit

Certainly the simplest, this bloat control method is relatively effective. It assumes that any offspring whose tree depth (or size) is above a threshold is too bloated to be useful. It is thus rejected and replaced by one of its parents. Proposed by Koza in his seminal book on GP [29], it remains as one of the most commonly used approaches to control bloat. Moreover, combining a static limit with other bloat control methods has been shown to improve efficiency for some operators [37]. However, setting the limit for non trivial problems without prior knowledge of the problem's dynamics may be challenging. It has also been shown that a fixed limit can be detrimental in some conditions [14].

For our experiments, we will use a static depth limit of 17, as defined by Koza [29]. Even though it has been demonstrated that it can be both too large for simple GP problems and too small for more complicated ones [7], it remains a good reference, and performs relatively well over a wide range of problems, including more difficult ones. It should also be noted that this setting was also previously used by most other papers on bloat control [37, 54, 55, 56].

2.2 Dynamic Depth Limit

Proposed by Silva and Costa [55], this method is intended to boost the efficiency of a static limit by allowing it to change during the evolution, depending on the current population. The limit is initially fixed to a relatively low value (for instance the depth or the size of the largest individual in the initial population). Any individual generated under this limit is unconditionally accepted. However, an individual not respecting this limit must prove that its larger size is worthwhile. Usually, this is done by comparing its fitness to the fitness of the best individual found so far. The individual is then accepted only if it is recognized as the new best-so-far individual. In this case, the limit is also increased to the size of this new individual, allowing the evolution to move the search space towards a promising area (that is, the area where the current best known solution is). As for the static limit, it has been shown by Silva and Costa [55] that a depth limit is often better than a size limit, and at least not worse.

For our experiments, we will use a non-heavy (non-decreasing) dynamic depth limit, without an upper static limit, which are essentially the settings that Silva and Costa recommend as being effective for a broad range of problems [55]. For each problem, the initial maximum depth was fixed to the height of the deepest individual generated during the first generation.

2.3 Parsimony Pressure and Double Tournament

Instead of using a specific limit (dynamic or not), another family of methods uses the selection process to restrain bloat. First suggested by Koza [29], one such approach attempts to apply a penalty on the fitness of the largest individuals. The difficulty, however, is to parametrize this penalty in a problem-independent way, and to ensure that the penalty does not promote the small individuals too much, resulting in poor performance [60]. Other similar methods such as Lexicographic Parsimony Pressure [36] or Biased Multi-objective Parsimony Pressure [44] have since then been proposed, using the selection process and adding the size as a second objective against the fitness. But it has been shown that these methods can be outperformed by a simple static limit in many situations [37].

Another interesting algorithm is the so-called Double Tournament, which, as its name suggests, uses two successive tournaments for selection: typically the first acts on tree sizes, selecting the participants of the second tournament based on fitness [35].

Because size tournaments with two participants tend to apply considerable size pressure, stochastic tournaments with a smaller average number of participants were finally suggested. For instance, a tournament with $D = 1.4$ participants will lead to tournaments with two participants and return the smaller individual with a probability of $D/2 = 0.7$, otherwise returning the larger one.

For our experiments, we will use a tournament size of $D = 1.4$, associated with a static depth limit of 17, which are the settings recommended by Luke and Panait [37]. As for the number of participants in the fitness tournaments, this was adapted to each problem in order to match the selection operator of other bloat control methods used.

2.4 Tarpeian

The Tarpeian method [46] also makes use of the selection process to control bloat, but in a different way. It considers that any individual beyond the average tree size might be undesirable, and thus assigns a very poor fitness to those individuals with a probability W. For instance, if $W = 0.3$, each individual greater than the average size will receive the worst possible fitness with a 30% chance, effectively reducing the amount of large individuals in the population.

One of the interesting characteristics of this operator is that this fitness degradation is actually conducted *before* the evaluation step, therefore leading to a substantial decrease in computational effort, as the fitness of some of the large individuals does not have to be evaluated at all. As the evaluation function usually has a runtime proportional to the size of the individual evaluated, this greatly reduces the computational effort needed to perform the evolution.

A variant able to automatically set W to follow a given average size target has been proposed recently [47]. While interesting, because it basically allows the user to select the mean tree size he wants over the evolution, it does not offer a complete bloat control method, since the user must still provide the optimal mean tree size for the problem (which is not trivial in most cases).

Algorithm 1 Generic Operator Equalization algorithm.

1: Initialize the population
2: **while** stop criterion not reached **do**
3: **Estimate** the current population size distribution;
4: **Determine** a target size distribution, possibly using the current distribution;
5: **Generate** a new population using the target size distribution, by accepting or rejecting individuals stemming from standard selection and variation operators.
6: **end while**

For our experiments, we will use the value $W = 0.3$ as this is the value that was used in previous comparisons [2, 37], and was shown to perform well in many situations. Like Double Tournament, it was also augmented with a static depth limit of 17.

2.5 Prune and Plant

The Prune and Plant method [1] is another recent bloat control operator, which defines a new offspring generation operator. Basically, the Prune and Plant operator selects a random non-terminal point in an individual and splits the individual at this point, effectively producing two new offspring from one parent. This is somewhat similar to some mutation operators (such as Hoist [26]), but according to the Prune and Plant authors, it performs better at controlling bloat than other existing mutation operators.

For our experiments, we will use the operator rate recommended by Alfaro-Cid *et al.* [1], that is a rate of 0.5, as it has been shown to provide a good performance over a wide range of problems.

2.6 Operator Equalization and its Variants

Dignum and Poli first developed a method referred to as Operator Equalization (OpEq), which is able to reach a given distribution of sizes (flat, triangular, etc.), by discretizing the distribution over bins of a given width and assigning an acceptance probability to each of these bins [15]. A very high level description of the OpEq algorithm is given as Algorithm 1. In the case of the seminal OpEq, the estimation of the current size distribution is simply the population produced at generation $t-1$, the target distribution is a fixed, user-defined size distribution, and the production of the target is achieved by assigning an acceptance probability to each bin. This acceptance probability is proportional to the ratio of the estimated current size distribution to the target size distribution.

A natural evolution of this operator, called DynOpEq [56], allows a dynamic adaptation of the target histogram. This operator brings two major changes to OpEq. First, the target is no longer static, but rather based on fitness, that is, line 4 in Algorithm 1 is replaced by the computation of a dynamic

size distribution. A bin containing better individuals will receive more space in the next generation, and conversely for weaker individuals. This is designed to quickly bias the search towards the most promising areas (i.e., size where fitness is better), and allow the evolutionary process to tune itself to the desired size distribution [57]. Second, acceptance (line 5) is no longer probabilistic: where OpEq makes use of an acceptance probability, DynOpEq remembers how many individuals were accepted in each bin so far in the offspring generation process. A new offspring can then only be accepted if its corresponding bin is not already full, or if its fitness is better than the current best individual of this bin. Besides, an offspring can create a new bin only if its fitness is better than the best-so-far individual, otherwise it is automatically rejected.

This last characteristic can induce many extraneous fitness evaluations per generation, as every individual, accepted or not, must have their fitness evaluated. This generates important computational overheads, especially in the early stages of evolution. To avoid this drawback, an incremental version called MutOpEq was developed [58]. As its name suggests, it makes use of a specific light mutation operator to modify the rejected individuals so that they can fit the target: the algorithm searches for the nearest free bin, and mutates the rejected individual in order to force it to fit into this bin.

During the experiments, it was discovered that the fitness-based target distribution was not biasing the search as much as expected. A recent paper [54] indicated that in the specific case of a real-world regression problem, all other parameters being the same, a flat distribution can outperform a dynamic distribution. This variation is called FlatOpEq, and simply consists in the generation of a flat target over all tree sizes currently present in the population.

For our experiments, we will use a bin width equal to 2 for all OpEq operators (DynOpEq, MutOpEq, and FlatOpEq), as a lower value such as 1 can lead to difficult situations for some problems – for instance, when the primitive set can only produce odd length genotypes and half of the bins thus remain empty [59], which is the case when all branches have an arity of 2.

2.7 Spatial Population Structure and Elitism (SS+E)

This recent technique uses an indirect means to control bloat, by mapping the population onto a 2D torus and using neighborhood-dependent elitist crossover operations, as defined in [72]. At each generation, an individual is first mated with one of its neighbors, and one of the generated offspring replaces the individual only if it improves fitness; otherwise the individual is left unchanged into the next generation. By performing this operation for every individual of the torus, the algorithm effectively controls bloat through spatially local crossovers.

For our experiments, we will use a square number as population size, namely 1024. The spatial structure and the neighborhood computation are the same as the configuration used in the relevant paper [72].

3 HARM-GP

To introduce HARM-GP, we first present some of the most common bloat theories. Then, we build on this foundation and on the seminal OpEq idea [15] to provide detailed explanations concerning the behavior of HARM-GP.

3.1 Measuring Bloat

The usual definition of bloat, as presented in the introduction, does not precisely specify the frontier between bloated and non-bloated, nor to which extent a given population is "bloated". Although admittedly sufficient for the sake of textual explanations, it is not sufficient to accurately compare two different evolutions. A more formal definition of bloat has been established in [70]: the *bloat level*. Basically, it measures the ratio of the size increase to the training fitness improvement. The smaller the value obtained, the less a population is considered bloated:

$$b(t) = \frac{(\bar{s}(t) - \bar{s}(0))/\bar{s}(0)}{(\bar{f}(0) - \bar{f}(t))/\bar{f}(0)}, \tag{1}$$

where $\bar{s}(t)$ and $\bar{f}(t)$ are respectively the mean size and the mean training fitness value at generation t, assuming a minimizing fitness. A bloat level of 0 is technically a bloat-free population, while any positive value indicates a given level of bloat.

3.2 Bloat Theories and Analysis

In [51] and [52], Poli and McPhee demonstrated that the expected change in the average individual size may be expressed as:

$$E[\bar{s}(t+1) - \bar{s}(t)] = \sum_{l} l(p(l,t) - \Phi(l,t)), \tag{2}$$

where $\bar{s}(t)$ and $\bar{s}(t+1)$ are the average size at generation t and $t+1$, l a given size, p the probability to choose individuals of size l among the population and Φ the current proportion of individuals of size l. In other words, the average size will change if and only if there is a disparity between the current size distribution and the probabilistic selection density. The bloat phenomenon can therefore be described as a situation where the large individuals have a higher selection probability than their current proportion in the population [49]. While useful, this equation in itself does not rationalize the bloat causes. We thus have to look at some specific bloat emergence theories.

3.2.1 Search Spaces Nature and Varieties of Bloat

The search spaces nature theory [31] states that the fitness selection is responsible for the disparity between p and Φ, at least after the first few generations. Indeed, reaching a better fitness becomes more and more difficult for an offspring as far as the evolution progresses, while producing larger offspring with

the same fitness as their parents remains relatively easy, because the search space is enlarged by the tree size increase. On the contrary, mutations and crossovers resulting in a tree shrink are often destructive and *reduce* the offspring's fitness, as shown in [61]. Some types of mutation are often neutral, but crossover is destructive most of the time while being the main genetic operator used in most studies. In this way, larger offspring have a proportionally higher chance of being selected in comparison to smaller offspring which, on average, exhibit poorer fitness. This results in a gradual drift to larger sizes over generations. In support to this explanation, it was shown that a GP evolution without selection pressure (using only random selection) did not exhibit bloat emergence [22, 33]. Moreover, recent preliminary research gave rise to promising results in bloat control by using a novelty search instead of a direct fitness measurement [67].

3.2.2 Structural vs. Functional Bloat

The search space nature theory originally presented in [31] can be extended to take into account other search space properties, especially the optimum localization. This was actually done in distinct but related work by Amil *et al.* [4], in order to support a theoretical analysis of genetic programming. In this paper, a new distinction was introduced to more accurately characterize the bloat: the difference between structural and functional bloat. Basically, *structural bloat* can be defined as "the code bloat that necessarily takes place when no optimal solution can be approximated by a set of programs with bounded length". *Functional bloat* is defined by the growth of the program size where the current size already suffices to reach optimal solutions. For instance, if, for a given problem, an optimal solution is known to require 25 nodes, then an optimal GP individual with 160 nodes will be characterized as functionally bloated.

There is an interesting aspect to this distinction. Structural bloat can be seen as an adaptation of the GP evolution process in response to the problem's difficulty. In this way, it is hardly avoidable, and even sometimes a desirable phenomenon. The program growth (including useless subtrees) can simply be mandatory to better explore the search space. In this context, the higher selection probability granted to the larger offspring is probably desirable. For instance, the fact that a subtree of an individual is useless at generation t, taking little or no part in the fitness score of its parent tree, does not mean that this subtree will remain useless at generation $t + 1$. A crossover may occur, placing this subtree in another tree where it will play a major role in its new individual performance. It was also shown early in the GP history that bloated trees exhibit a better resistance to the destructive effects of crossovers and mutations [9, 24, 41]. Therefore, a bloat control method which is too restrictive may actually decrease the performance of the best final solution.

3.2.3 Crossover Bias

The crossover bias theory [13] is a more recent theory which does not contradict, but completes the search space nature theory. It explains bloat emergence by the crossover operation itself. In GP, by exchanging two random subtrees, the standard crossover operation does not change the mean tree size, since no genetic material is actually added or discarded, but it tends to reshape the tree size distribution. Poli *et al.* [48] showed that this distribution can be approximated by a *Lagrange distribution of the second kind*, which is biased towards either smaller or larger individuals. But since very small individuals tend to be unfit, they are also discarded more often by the selection process, which in turn promotes code growth over generations.

3.3 Generic Bloat Control Model

At this point, it can be useful to derive a generic model for bloat control techniques from Eq. 2. In a bloat control context, $\Phi(l,t)$ can be considered as fixed, since it only depends on the current population. However, it is possible to impede bloat emergence by acting on $p(l,t)$, which will in turn change $\Phi(l,t+1)$. This can be seen by the addition of another factor in the equation:

$$E[\bar{s}(t+1) - \bar{s}(t)] = \sum_{l} l(\lambda(l,t)p(l,t) - \Phi(l,t)), \tag{3}$$

where $\lambda(l,t)$ is a distribution able to constrain $p(l,t)$ to a certain shape. For instance, the static size limit method can be seen as a simple cutoff point on the selection probabilities (e.g., $\lambda(l,t) = 0,\ \forall\ l > L$ where L is a fixed threshold). The parsimony pressure boils down to the use of an arbitrary decreasing distribution as λ, meaning that a small individual sees its selection probability increased while a large one has its own decreased. The covariant Tarpeian method also directly derives from this equation [47], with the difference that λ is adjusted at each generation to fit a given mean size target.

However, this ability to change the selection probabilities is not enough to create a complete bloat control method, since an important question remains: which optimal distribution should be followed? Some studies have been conducted on that point [15, 47], and conclude that there is simply no general answer. A given distribution may work well only for some problems, or even for some *instances* of a problem. Moreover, even when considering a single instance, the optimal distribution changes over the evolution. Therefore, the distribution selection method is crucial to achieving good performance.

3.4 Distribution Alteration by Histogram

While $\lambda(l,t)$ can be determined analytically under certain conditions, many methods do not use an analytical solution, but rather implicitly affect the population. Indeed, the discreteness of l naturally leads to the use of a histogram. Besides providing the possibility of setting the mean tree size like the analytic

models, a histogram method has the advantage of also being able to shape the complete size distribution. In this aspect, OpEq [15] was one of the first successful attempts of altering the size distribution by acting only on discrete bins of the size histogram. Nevertheless, this original paper did not define a way of computing the optimal size distribution for a given population.

Silva's OpEq variants [54, 56] attempt to solve this problem, but often at the expense of a tremendous computational effort, especially when the population has to be fitted into a size distribution far from its natural distribution (e.g., FlatOpEq). Thus, while effective in many ways, those variants cannot always be used in practice because of the high computational effort they imply [68].

On the other hand, using acceptance probabilities, HARM-GP operates by dynamically controlling the size distribution of a population over generations. As the other OpEq variants, it builds on the seminal OpEq idea [15] presented in Algorithm 1 but uses more efficient mechanisms to ensure that the population follows a given target. Moreover, it offers a new, original way of computing this optimal target. Contrary to Dyn/Mut/FlatOpEq, HARM-GP is a probabilistic method, and works without requiring any extra fitness evaluations. This is an important characteristic for many real-world problems where an evolution may require hours or even days to complete [65, 75], because execution time is mostly dominated by fitness evaluation. The following subsections present the different elements of the HARM-GP algorithm, first beginning with the refinements to OpEq, and followed by a complete explanation of the target estimation method.

3.5 Modifications to OpEq

As mentioned before, Operator Equalization was shown to be quite successful in forcing the population to adopt a given size distribution. However, we identified some areas where its efficiency and accuracy could be improved. In particular, we focused on the acceptation procedure and the distribution averaging.

3.5.1 Accept-Reject Method

The original OpEq algorithm makes use of a probability vector, one acceptance probability being associated with each length. These probabilities are changed according to the current size distribution. While relatively effective, it suffers from two related problems. The first one is the forced creation of a "rate" parameter characterizing the algorithm response time when facing a disturbance in the histogram. In their paper, Dignum and Poli empirically set this parameter to a value of 0.1 [15], but this was not motivated and may not be the best choice for some problems. The second problem is that the speed of convergence to the desired distribution largely depends on the current one. If the latter is very different from the former (such as at the beginning of the run), then the convergence will require many generations, even though it would be possible to obtain the desired distribution more quickly through rejections.

HARM-GP makes use of a generalized accept-reject method [53] (a.k.a. rejection sampling), which solves both of the problems mentioned above. The

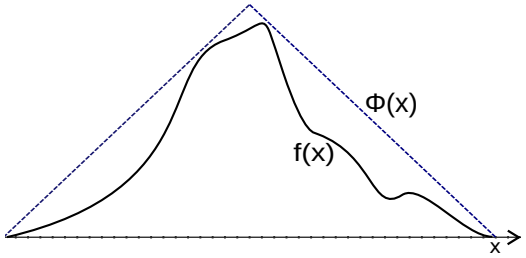


Figure 1: Envelope and target distributions for the accept-reject method.

accept-reject method does not have any parameter, and ensures that the produced distribution *immediately* follows the target.

This Monte Carlo method, first proposed by John von Neumann in the early 1950s, generates values following an arbitrary distribution (the target distribution) from values following another known distribution (the envelope distribution). The approach consists in accepting a value x generated according to the envelope distribution $\phi(x)$ with a probability $\mathrm{p(x)} = \mathrm{f}(x)/\phi(x)$, that depends on the ratio of the envelope to the target distributions $\mathrm{f}(x)$. It assumes that the envelope distribution $\phi(x)$ is greater or equal to the target distribution $\mathrm{f}(x)$ over the entire domain of x. Formally, the method is defined as follows:

1. Generate random value $X \sim \phi(x)$
2. Generate uniform random number $Y \sim \mathcal{U}(0, 1)$
3. If $Y \leq \mathrm{f}(X)/\phi(X)$ accept X by returning $Z = X$, otherwise return to Step 1.

It is easy to show that the distribution of numbers Z will follow the target distribution $\mathrm{f}(x)$.

Fig. 1 illustrates the method by presenting the envelope distribution $\phi(x)$ and the target distribution $\mathrm{f}(x)$, which should be less than or equal to the envelope for all x. The method is commonly used to generate pseudo-random numbers following arbitrary target distributions, using as an envelope a transformed uniform distribution (generally by a constant scaling or a linear transformation). A key element for an efficient exploitation of this method is to keep the envelope as close as possible to the target distribution. Otherwise, the rejection rate can become quite high, requiring many additional computations to fit the target distribution.

3.5.2 Histogram Smoothing

Another common problem with size histograms in GP comes from the specificities of the primitive set. In particular, it is common to have only primitives

with an even arity (typically two), leading to a histogram where all the even sizes are unoccupied, simply because it is impossible to generate them. On other problems, the primitive arities can influence the size distribution, and thus some sizes are less likely to be obtained than others.

This problem was often addressed through a bin width adjustment [15, 56]. However, this approach requires selecting a bin width parameter that can have considerable impact on the quality of the estimation. Extremely narrow bins lead to high noise sensitivity, while excessively large bins lead to possible information loss. Moreover, histograms are sensitive to the influence of data close to frontiers between adjacent bins, which can distort the distribution in random ways.

Some common methods are better than standard histograms for estimating a density over a given domain. One of them is the so-called *kernel density estimation* (a.k.a. Parzen window or Parzen-Rosenblatt window), which applies a kernel function to each distribution value in order to smooth off the distribution. The mathematical formulation for estimating the distribution $\hat{\mathrm{f}}(x)$ from n points $\{x_i\}_{i=1}^{n}$ over a real-valued domain is given by the following equation:

$$\hat{\mathrm{f}}(x) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{K}(x - x_i), \tag{4}$$

where x is the value at which we want to determine the distribution $\hat{\mathrm{f}}(x)$ and $\mathrm{K}(\cdot)$ is the kernel function. Kernel functions are symmetric functions that integrate to 1; commonly used functions include uniform, triangular, or Gaussian kernels.

In HARM-GP, we used a discrete triangular kernel of width 5, which considerably reduces the noise over the size distribution. Moreover, this setting was found to be able to adequately smooth the distribution for every problem tested. The exact mathematical formulation of the normalized kernel is given by the following equation:

$$\mathrm{K}(x) = \begin{cases} 0.4/2^{|x|} & \text{if } x \in \{-2, -1, 0, 1, 2\} \\ 0 & \text{otherwise} \end{cases} . \tag{5}$$

3.6 Target Size Distribution Computation

The production of a given target size distribution in GP leads us to its application towards bloat control. The target can greatly affect the performance of the operator, so it has to be chosen with care. HARM-GP defines its target as a decaying exponential starting from a dynamically defined *cutoff size*.

The cutoff size corresponds to the size of the smallest individual which reaches a certain percentage of the best fitness found so far. Though individuals smaller than this size can still be bloated, their bloat is more likely to be structural than functional, according to the definitions given in Sec. 3.2.2. In other words, finding good solutions which are smaller than the solution selected for the cutoff size is not likely. If it is possible to find such a solution, it is likely to adjust the cutoff size at the next generation accordingly.

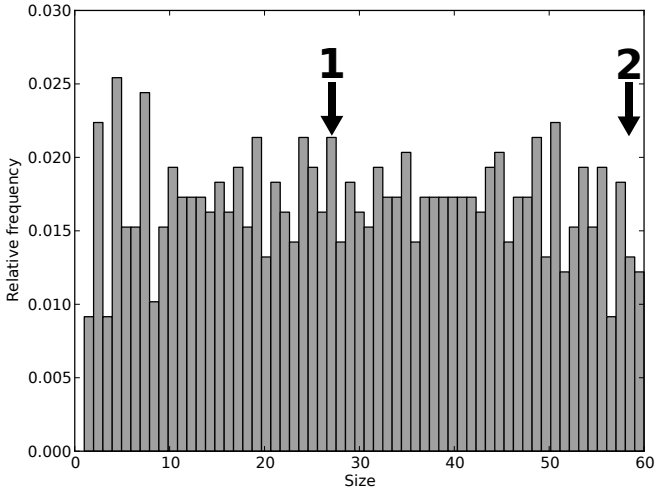


Figure 2: Size distribution of a population illustrating two possible cases: 1) the best individual has an average size compared to the population and resides in the bin designated by arrow #1; and 2) the best individual is large in comparison with the population size distribution, residing in the bin designated by arrow #2.

Let us consider the two alternatives presented in Fig. 2 where the best individual is either of average size (case #1), or amongst the larger individuals (case #2). In the first case, the individuals larger than the best one are likely to contain *functional* bloat, since the best individual achieves its fitness score with a substantially lower number of nodes. On the contrary, in the second case, the size distribution suggests that if there is bloat occurring in this population, it is likely to be *structural*, since the evolution was unable to find any optimum with a smaller number of nodes. Again, if there actually *is* a smaller solution, then we are likely to find it in the next generations, which would bring us back to case #1.

That being said, using only the absolute best individual can be misleading in real-world problems. First, because the difference in fitness between the best and the second best can be negligible, or even simply due to rounding errors. Second, given that bloat can still have a *non-significant* effect on an individual fitness, the best individual could actually be more bloated than a slightly worse individual with a significant smaller size. Third, an individual with a slightly better fitness is likely to have memorized one or several samples of the training set. This memorization does not usually generalize well, and increases the chances of overfitting. While improving the generalization capability of GP is not our first objective, it is still an important target to improve its performance.

Typically, we define the cutoff bin as the bin containing the *smallest individual that reaches the top 10 fitness percentile*. This is configurable, however, for example when dealing with a discrete objective function, where we may want to directly set the cutoff point to the size of the best individual so far.

For sizes which are smaller than the cutoff point, no restriction is enforced at all: the population simply follows its natural distribution. Even if the crossover bias theory points out that small unfit individuals may lead to bloat emergence [14], these individuals themselves are *not* bloated, and the algorithm has no reason to reject them (we distinguish *causing* bloat from *being* bloated). We then want to restrain individuals larger than the cutoff size from entering the population. This is achieved by setting an *exponential decay* from this cutoff size. This choice was made considering that the accept-reject method reaches better efficiency when the target approaches the envelope. Now, the schema theorems applied to GP show that crossovers produce a *gamma distribution* [51]. Consequently, the choice of a decaying exponential is motivated by the fact that the right hand side of the gamma distribution can be approximated by a distribution of this type. This choice also has the benefit of minimally impairing the evolution process, by reducing the number of rejected individuals. Finally, the decaying exponential does not enforce a too sharp cutoff point in the distribution, ensuring that individuals larger than the cutoff can still be accepted. This helps to avoid the degenerated case where the only individuals allowed would be too small to improve performance, as pointed out by Soule and Foster [60].

Another interesting property of the decaying exponential is that it can be characterized by two parameters: its half-life and the area under the curve (i.e., the total number of individuals larger than the cutoff size allowed). These

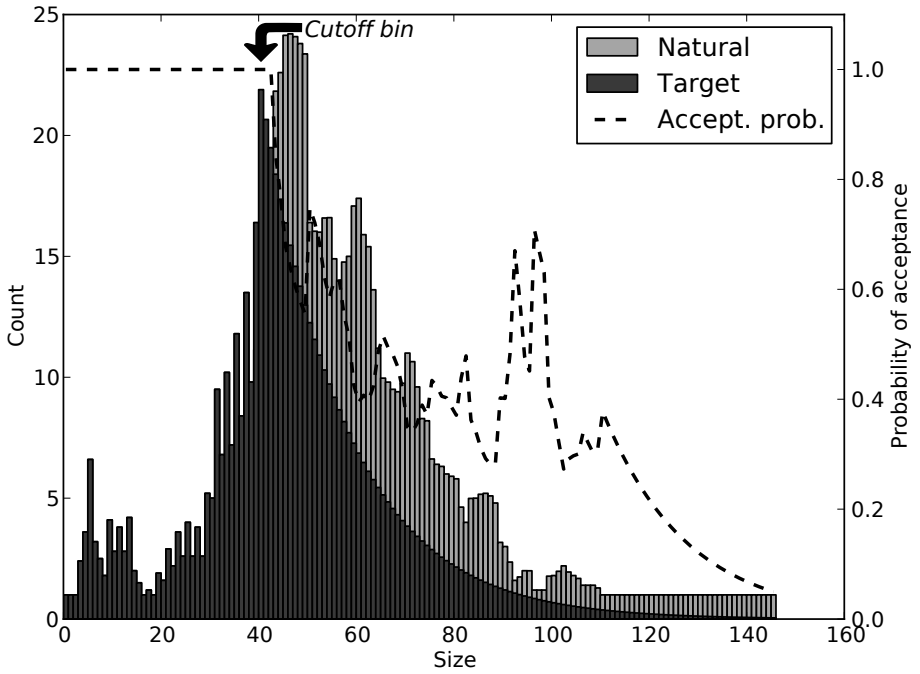


Figure 3: Typical HARM-GP distribution. The acceptance probability is the ratio of the target to the natural distribution bins. To avoid large variations between consecutive bins, which would render the graph illegible, the acceptance probability is only plotted for the odd bins.

parameters can be set to generic values, but can also be used to tune the code growth pressure according to a specific problem.

Globally, the target generation is given as:

$$H_{\text{tar}}(i) = \begin{cases} H_{\text{nat}}(i) & \text{if } i \leq c \\ \Gamma(M) \exp\left[-\frac{\ln(2)}{\tau} \cdot (x_i - x_c)\right] & \text{otherwise} \end{cases}, \tag{6}$$

where x_c is the size value of the cutoff bin, x_i is the size value of the i-th bin, τ the half-life of the exponential, $\Gamma(M) = \gamma M \ln(2)/\tau$ is a function of the population size M to obtain the desired fraction of population beyond the cutoff bin (γM individuals, where $0 < \gamma < 1$), $H_{\text{nat}}(i)$ is the count (height) of the i-th bin of the natural distribution (envelope), and $H_{\text{tar}}(i)$ is the count for the target distribution of the i-th bin.

3.7 Overview of HARM-GP

At this point, we have defined all of the HARM-GP building blocks. The envelope distribution of the accept-reject method corresponds to the natural dis-

tribution of the population resulting from the selection, crossover, mutation, and reproduction operators without bloat control and after being filtered by the kernel density estimator. This corresponds to line 3 of Algorithm 1, and can easily be estimated by generating many individuals with these operators, keeping in mind that the generated offspring are not wasted, that they can be reused thereafter. The representation for this distribution is a tree size histogram built using the kernel density estimation of Eq. 4.

The target distribution (Algorithm 1, line 4) is simply the desired tree size histogram we wish to obtain, as explained in the previous section. An acceptance probability is then computed for each bin (Algorithm 1, line 5). This probability corresponds to the ratio of the target to the natural distributions. For each individual, the algorithm chooses to accept this individual or not, using the probability corresponding to its size. Overall, the generated distribution closely matches the target distribution.

Because of the probabilistic nature of HARM-GP, it is important that the natural size distribution be estimated using a relatively large number of offspring. Since GP already requires large populations to be successful with hard problems [29], this is not a serious issue in practice. Moreover, there is no need to evaluate fitness before deciding to accept or reject individuals, decision being solely based on size. Thus, assuming that fitness evaluation is much more computationally expensive than offspring generation, the induced overhead compared with standard GP is low, if not negligible.

Of course, being stochastic, HARM-GP may sometimes produce a slightly different population than the required target. In practice, this is not an issue either as the deviation remains very small – we asserted this by using standard histogram distance metrics (i.e., Kullback-Leibler divergence and Bhattacharyya distance).

Fig. 3 illustrates typical HARM-GP behavior. Notice the shape of the envelope distribution, close to the predicted gamma function [50]. As can be seen by the acceptance probability graph, the algorithm accepts all individuals with 42 nodes or less, but more or less severely restrains the emergence of larger individuals. For instance, an individual of size 60 has about a 40% chance of being accepted, while an individual of size 85 has less than a 30% chance, as the natural distribution tends to promote size 85 more often than others. However, the acceptance probability increases around size 90-100, where a very small number of individuals of these sizes are created naturally by the genetic operators.

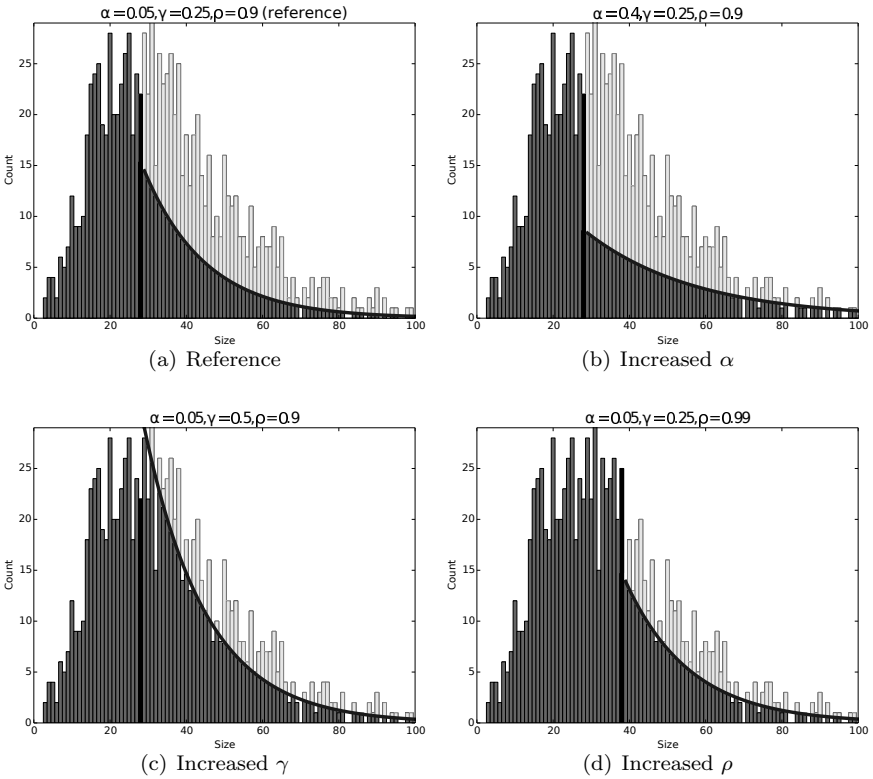
3.8 Evolutionary Loop

Algorithm 2 presents the high-level evolutionary loop of the HARM-GP procedure. In lines 1 and 2, a population of size M is first initialized in any desired way. In line 3, the evolutionary loop is run over $t_{\max}$ generations, or until some other (unspecified here) stopping criteria. In line 4, a temporary population P' of size M' is generated using any combination of variation operators (selection, crossover, mutation, reproduction, etc.) applied to the previous generation P^{t-1}. Value M' is not necessarily related to M, but must be large enough to

Algorithm 2 High-level evolutionary loop of HARM-GP.

1: $P^0 \leftarrow$ initialize_population(M)
2: Estimate fitness of all individuals in P^0
3: **for** $t = 1 \rightarrow t_{\max}$ **do**
4: $\quad P' \leftarrow$ generate_offspring(P^{t-1}, M')
5: $\quad H_{\text{nat}} \leftarrow$ estimate_histogram(P')
6: $\quad H_{\text{tar}} \leftarrow$ create_target(H_{nat}, P^{t-1})
7: $\quad P^t \leftarrow \emptyset$
8: $\quad$ **repeat**
9: $\quad\quad$ **if** P' not empty **then**
10: $\quad\quad\quad j \leftarrow$ choose_next(P')
11: $\quad\quad\quad P' \leftarrow P' \backslash \{j\}$
12: $\quad\quad$ **else**
13: $\quad\quad\quad j \leftarrow$ generate_offspring($P^{t-1}, 1$)
14: $\quad\quad$ **end if**
15: $\quad\quad p_{\text{accept}} = H_{\text{tar}}(\text{size}(j)) \,/\, H_{\text{nat}}(\text{size}(j))$
16: $\quad\quad$ **if** rand$(0,1) < p_{\text{accept}}$ **then**
17: $\quad\quad\quad P^t \leftarrow P^t \cup \{j\}$
18: $\quad\quad$ **end if**
19: $\quad$ **until** $|P^t| = M$
20: $\quad$ Estimate fitness of all individuals in P^t
21: **end for**

achieve a good estimation of the natural population distribution. A value of $M' \geq \max(2000, M)$ is recommended, that is to say that a minimum value of 2000 individuals should be used in order to obtain a sufficiently accurate estimation. If the population size is greater than this value, then HARM-GP can benefit from a more precise estimation with these supplementary individuals, as these would be generated anyway. In line 5, a kernel density estimation of the size distribution is conducted to determine the natural distribution histogram H_{nat} (see Sec. 3.5.2). Line 6 involves the computation of the target distribution histogram H_{tar} (see Sec. 3.6). Then, the loop between lines 8 to 19 consists in first selecting an individual j from population P' (lines 9 to 14), and adding this individual to the output population P^t if it passes the acceptance test (lines 15 to 18; see Sec. 3.5.1 and 3.6 for details). In the special case where population P' becomes empty (lines 12 and 13), a new offspring must be generated using the same algorithm as for line 4. Note that this process is assumed not to be computationally expensive as fitness is not evaluated at this step. As soon as population P^g is complete (line 19), the loop ends and the fitness of all successful individuals is evaluated before completing the current generation (line 20). Any residual offspring in P' are discarded as they are no longer useful. This may only happen when $M' > M$, but it can be safely assumed that any wasted computational resources are low, if not negligible for real-world problems, since these offspring never had their fitness evaluated.



(a) Reference (b) Increased α

(c) Increased γ (d) Increased ρ

Figure 4: Examples of different parametrization with the same input population: (a) recommended parameters used as reference; (b) effect of an increased α; (c) effect of an increased γ; and (d) effect of an increased ρ.

3.9 Parameters

HARM-GP has three main parameters: α, γ, and ρ. Parameter α defines the half-life $\tau = \alpha x_c + \beta$ of the exponential. The half-life is linearly proportional to cutoff point x_c with a minimal value set to β, so that reasonable size growth is always possible, even during the first few generations where sizes can be small. The exact value of β is thus not very important for the algorithm performance. Parameter γ is used to define the percentage of individuals that are allowed after the cutoff point. Finally, ρ partly controls the position of the cutoff point, by setting the range of fitness in which to search for the smallest individual. For instance, if $\rho = 0.9$, then the cutoff point will be the size of the smallest individual of the top 10 fitness percentile. Similarly, a $\rho = 1$ would always select the best individual as the cutoff point (if multiple individuals share the same best fitness, then the one with smallest size is chosen). Fig. 4 presents four configuration examples, in order to illustrate the effect of each parameter.

Table 1: List of used benchmark problems.

Problem	Domain
Artificial Ant [29]	Planning
Even Parity 8 bits [29]	Combinational Logic
Keijzer-6 [25]	Symbolic Regression (univariate)
Nguyen-7 [69]	Symbolic Regression (univariate)
Pagie-1 [43]	Symbolic Regression (2 var.)
Vladislavleva-4 [71]	Symbolic Regression (5 var.)
Bioavailability [58]	Real-life Symbolic Regression (241 var.)
Dow Chemical [28]	Real-life Symbolic Regression (57 var.)
Adult Dataset [74]	Real-life Classification (123 features)
Magic Telescope Dataset [10]	Real-life Classification (10 features)
Spambase Dataset [34]	Real-life Classification (57 features)
PSP100 [5]	Real-life Classification (100 features)

3.9.1 Recommended Parameter Values

In all of our experiments, unless stated otherwise, we have used the stable parametrization $\alpha = 0.05$, $\beta = 10$, $\gamma = 0.25$, and $\rho = 0.9$ that performs well over all tested problems. On average, this parametrization generates a population with 25% of its individuals larger than the cutoff point. Recall that when it comes to tuning the algorithm, α and γ are the most important parameters. The first sets the legroom for growth according to the current cutoff point: the greater the half-life, the higher the chance that larger individuals are accepted. The second sets the proportion of the population that is allowed to grow in size. With $\gamma = 0$, the number of individuals allowed after the cutoff bin tends to zero, and size growth becomes impossible. Finally, higher values of ρ can be used to search more aggressively for slightly better solutions, with the potential downside of increasing the risk of overfitting. Note that a sensitivity analysis of these parameters is conducted in Sec. 5.1.

4 Experimentations

To compare the strengths and weaknesses of the different considered bloat control methods, we conducted experiments on an extensive set of 12 problems, each with its specific dynamics. The detailed list of problems is given in Tab. 1. Each method is applied 100 times on every problem, using a common parametrization summarized in Tab. 2. For each independent run, the same initial population is provided to each method.

The specific parameters used for the different bloat control algorithms are those presented in Sec. 2. They are either the default parameters prescribed by the algorithm's authors, or values used in previously published papers. They are the same across the whole problem set.

In the following analysis, a result is said to be *significant* only if it is asserted

Table 2: General parameters used for all experiments.

Parameter	Value
Number of independent runs	100
Population size	1024
Crossover distribution	Biased towards non-terminals with prob. 0.9
Crossover probability	0.8
Subtree mutation probability	0.1
Replication probability	0.1
Tournament size for selection	8
Initialization method	Ramped half-and-half over depth $[2, 5]$

by a Wilcoxon signed-rank test at $p < 0.01$, with a continuity correction.

4.1 Comparison Methodology

Controlling bloat, and more generally code growth, is by essence a multiobjective problem that seeks to maximize performance (in terms of fitness) while minimizing solution size and computational effort. Performance is obviously the main objective, but at equivalent performance, smaller solutions obtained through less computational effort are highly desirable.

To compare the behaviors of the different bloat control methods over the selected problems, we first allocated a fixed budget of 150 000 fitness evaluations, corresponding to about 150 times the population size. This budget results in runs of about 150 generations for most methods, which is more than enough to both allow convergence and bloat emergence for all problem types. However, in the cases of DynOpEq and FlatOpEq, because these methods tend to consume a very large number of evaluations during their first few generations [54, 56], their budgets were doubled (300 000 evaluations), effectively allowing them to remain competitive with respect to performance. Otherwise, using the default budget, they were often observed to perform badly for lack of convergence. Hence, the reader should keep in mind that, while hindering their computational efficiency, this special measure also favors their performance.

In the next subsections, results will be analyzed according to the following criteria, in decreasing order of importance.

1. The performance of the best found solution, named the *best-of-run* individual, as measured by a problem specific minimizing objective function. Whenever possible, this performance will be measured using a testing data set independent of the training set that was used to guide the emergence of this best-of-run solution, and to select it. This is important given that the fitness computed using the training set is a biased estimator of performance that can be quite misleading [3]. Indeed, the evolutionary process can evolve solutions that learn by rote the samples of the training set without being able to generalize.

In the following, let $f^*_{x,i}$ designate the fitness on the *training set* of the best-of-run individual for method x during run i, and let $\bar{f}^*_x$ denote either the median or the average of $f^*_{x,i}$ over all runs. Similarly, let $g^*_{x,i}$ designate the performance on the *testing set* of the best-of-run individual for method x during run i, and let $\bar{g}^*_x$ denote either the median or the average of $g^*_{x,i}$ over all runs[1]. For most problems, we prefer the median because it is less sensitive to outliers. Indeed, even a single non converging run can greatly affect the average performance. The exceptions are for the Artificial Ant and Even Parity problems, where we instead prefer the average, because methods often succeed in producing optimal performance for these problems, and the median becomes unsuitable for discriminating between methods, especially when the success rate approaches or exceeds the 50% mark.

2. The size of the best-of-run solution, as measured by its number of tree nodes (primitives). The size metric is directly related to the level of bloat in an individual, which is characterized by unnecessary large individuals. A smaller solution will tend to overfit less. It will also be faster to compute and easier to interpret or to integrate into a broader system.

   In the following, let $s^*_{x,i}$ designate the size of the best-of-run individual for method x during run i, and let $\bar{s}^*_x$ denote the average of $s^*_{x,i}$ over all runs.

3. The computational effort[2] necessary to complete the evolutionary process, as measured by the cumulative total number of evaluated tree nodes. In practice, the time required to perform an evolution may prove to be one of the most limiting factors for real world problems. This metric was used in previous works to compare the efficiency of different GP variants [66, 69]. It assumes that the fitness evaluation step is the time dominating task in the evolutionary algorithm, and that this time is proportional to tree size.

   In the following, let $c_{x,i}$ be the cumulative total number of evaluated tree nodes for method x during run i, and let $\bar{c}_x$ denote the average of $c_{x,i}$ over all runs.

4. The level of population bloat, as measured by the *bloat level* metric (see Sec. 3.1 and [70]). This metric seeks to quantify the actual bloat control capacity by comparing relative size increases with relative fitness improvements. The smaller the value, the less a population is considered bloated. Note that this metric ignores absolute sizes and fitnesses, so should not be used alone.

   In the following, let $b_{x,i}$ designate the final population bloat for method x during run i, and let $\bar{b}_x$ denote the average of $b_{x,i}$ over all runs.

[1] For two of our twelve problems, namely Artificial Ant and Even Parity, the training and testing sets are the same, since the task is rather a planning problem (Artificial Ant) or the training set already samples the full problem space (Even Parity). Considering this, $f^*_{x,i}$ and $g^*_{x,i}$ refer to the same values for these specific problems.

[2] We do not refer to the original concept of *computational effort* developed by Koza, but rather to the actual processing effort needed to perform an evolution.

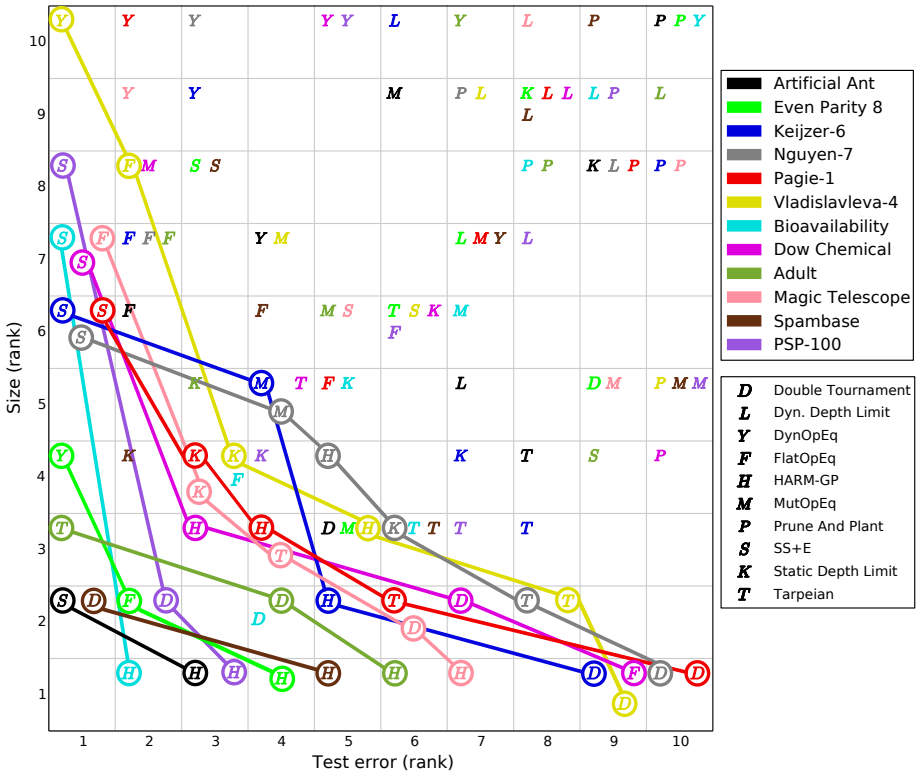


Figure 5: Best-of-run performance $\bar{g}_x^*$ and size $\bar{s}_x^*$ for each method x applied to all problems. This graph uses the rank of each method as a discrete measurement to position method specific markers using problem specific colors. The Pareto fronts are outlined for each problem.

4.2 Results Overview

We first present in this section an overview of results, aggregating the data from all problems in order to provide an overall picture of the relative performance of the different methods. Detailed results will follow in Section 4.3.

4.2.1 Scatter Graph

Fig. 5 presents the performance and size ranks of the best-of-run individuals for each method and each problem. Methods have different markers, and problems are identified by different colors. All markers inside a given square represent the same rank position. For this graph, the ideal position is square $(1, 1)$ where a method achieves both the best fitness and the smallest tree size. Every other position can be considered as a trade-off between size and performance. Markers

that are outlined by a circle and linked by line segments designate those methods that lie on the Pareto front for a given problem. Note that ranks are discrete and have no absolute meaning; they should be interpreted with care. Moreover, even though one method may have a better rank than another, it does not imply that the difference in performance or size is statistically significant. Detailed results with statistical significance are presented later.

Interesting patterns emerge from Fig. 5. Some methods generally obtain very good fitness ranks, for instance SS+E and DynOpEq, but at the cost of larger sizes. Likewise, other methods often exhibit the smallest sizes, but at the expense of some fitness, for instance HARM-GP, Double Tournament or Tarpeian. Overall, the most interesting methods are those that lie on the Pareto fronts, namely HARM-GP (12/12), Double Tournament (9/12), SS+E (7/12), Tarpeian (5/12), FlatOpEq (4/12), and Static Limit (4/12). Some methods are never on the Pareto front, namely Dyn. Depth Limit (0/12) and Prune And Plant (0/12). Finally, it should be observed that the lengths of the Pareto fronts vary from 2 methods for Artificial Ant, Bioavailability, and Spambase, to 6 methods for Nguyen-7 and Vladislavleva-4.

4.2.2 Relative Histogram

To better illustrate the magnitude of differences between methods, the double histogram of Fig. 6 plots relative performances and sizes for each method and each problem.

For a given problem, the relative performance improvement G_x of method x is defined by:

$$G_x = \frac{\bar{r}^* - \bar{g}_x^*}{\bar{r}^* - \min\limits_{y \in Y} \bar{g}_y^*} \tag{7}$$

where $\bar{r}^*$ represents the median (or average for Artificial Ant and Even Parity) of the best individuals found in the initial (random) populations used to bootstrap the different runs, and Y is the set of methods. Note that the value of $\bar{r}^*$ is independent of x, because the same 100 initial populations are used to bootstrap all methods for a given problem. It should also be noted that we assume a minimizing objective function. A method x with a $G_x = 0.95$ is thus able to reach 95% of the maximum observed improvement.

Similarly, the relative size S_x of method x is defined as:

$$S_x = \frac{\bar{s}_x^*}{\min\limits_{y \in Y} \bar{s}_y^*} \tag{8}$$

Thus, the method that achieves the best median (or average) performance will always obtain a relative improvement of 1, and the method producing the smallest average solution size will always obtain a relative size of 1. All other values indicate a performance decrease or a size increase. For instance, a score $S_x = 7$ indicates that for a given problem, a method x produces solutions that are on average seven times larger than the best method for size.

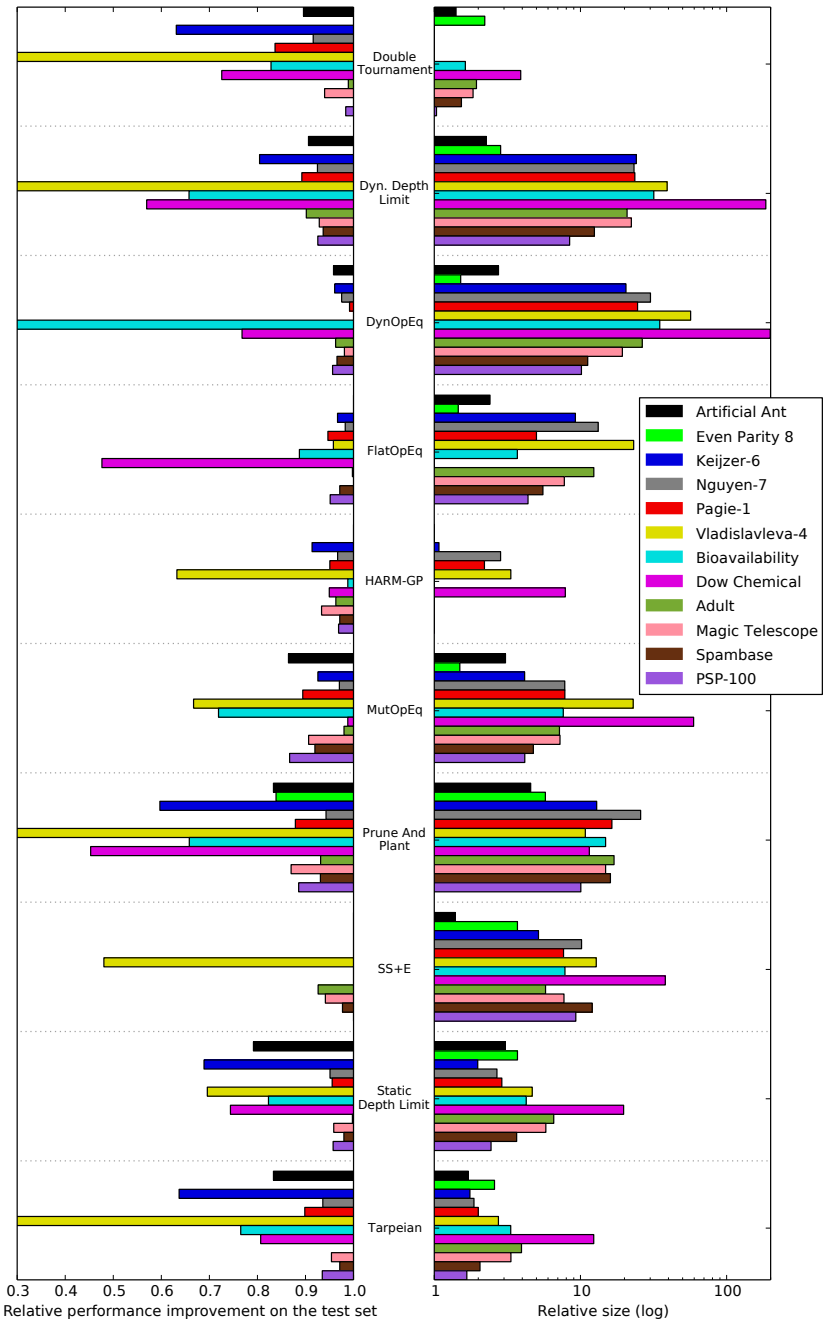


Figure 6: Per method relative performances and sizes. For the performance bar-plot (left), a value of $G_x = 0.95$ indicates that method x reaches 95% of the best median (or average) observed performance improvement. Similarly, for the size graph (right), a value of $S_x = 7$ indicates that method x produces solutions that are on average seven times larger than the best method for size.

It is important to note that these relative results cannot be used to compare methods *across* problems, since each problem has its own maximum performance improvement and its own baseline size. These relative metrics should rather be seen as a convenient way to place side-by-side on a common graph the performance and size results.

These results show that the rankings illustrated in Fig. 5 should indeed be interpreted with care, because the differences in performance can be rather small, no matter their statistical significance. For instance, DynOpEq has a performance loss of 5% or less compared to the best performing method for all problems except Bioavailability and Dow Chemical. Likewise, HARM-GP never induces a performance loss of more than 10%, apart for Vladislavleva-4, and is often under 5%. On the contrary, the size differences can be quite large, even by orders of magnitude in some cases (notice the log scale for size).

It should finally be noted that the unexpected behavior of most methods on the Vladislavleva-4 problem can be explained by its high level of difficulty, the RMSE improvement over the whole evolution being around 0.04 for the best performing method. Since the best improvement is quite small, even a minimal loss in performance is seen as a huge relative difference on the bar chart.

4.2.3 Summary Tables

Table 3 allows a global comparison of methods using pairwise statistical Wilcoxon signed-rank test with $p < 0.01$. For each criterion (performance, size and effort), a method that significantly outperforms another obtains $+1$, while the other receives -1 when it is significantly worse. When the difference is not significant, the score given is 0.

This process produces a 10x10 matrix for each problem. We then sum these matrices to obtain an overview of the general performance. For instance, a score of $+4$ at position (i, j) indicates that the method on row i provides better performance than the one on column j at least four times – it may also perform better six times, but exhibit a worse performance on two other problems, and so on. We also provide the sum of each line as an indicator of the method's performance against that of every other method. While this sum has to be interpreted with care, since a positive score for a method necessarily involves a negative one for the opponent method, it remains a good indicator of the general performance on each criterion.

Results show that SS+E is the best performing method when considering only the performance criterion, and HARM-GP obtains the best results in size and computational effort. However, HARM-GP still obtains good performance results – in particular, it was only outperformed three times by SS+E, and its total score gives it a third rank for overall performance – while SS+E exhibits poor results in terms of both size and effort. Double Tournament and Tarpeian also reach interesting results in these aspects, but clearly score lower in terms of performance. Overall the results presented in Table 3 support the analysis of Fig. 5, but with an added statistical significance.

Table 3: Per method pairwise comparisons using a Wilcoxon signed-rank test with $p < 0.01$; a significant result counts for either $+1$ or -1.

Performance	1	2	3	4	5	6	7	8	9	10	Total
(1) Double Tournament	0	2	0	-4	-4	-1	5	-6	-3	0	-11
(2) Dyn. Depth Limit	-2	0	-7	-10	-7	-4	4	-11	-3	-4	-44
(3) DynOpEq	0	7	0	-2	2	3	8	-4	2	2	18
(4) FlatOpEq	4	10	2	0	2	7	10	-2	6	5	44
(5) HARM-GP	4	7	-2	-2	0	2	11	-3	1	2	20
(6) MutOpEq	1	4	-3	-7	-2	0	7	-7	-3	-1	-11
(7) Prune And Plant	-5	-4	-8	-10	-11	-7	0	-11	-7	-7	-70
(8) SS+E	6	11	4	2	3	7	11	0	6	8	58
(9) Static Depth Limit	3	3	-2	-6	-1	3	7	-6	0	1	2
(10) Tarpeian	0	4	-2	-5	-2	1	7	-8	-1	0	-6
Size	1	2	3	4	5	6	7	8	9	10	Total
(1) Double Tournament	0	10	10	8	-2	10	11	11	12	10	80
(2) Dyn. Depth Limit	-10	0	2	-10	-11	-9	-2	-8	-7	-10	-65
(3) DynOpEq	-10	-2	0	-10	-12	-10	-5	-8	-9	-10	-76
(4) FlatOpEq	-8	10	10	0	-9	2	8	2	-3	-7	5
(5) HARM-GP	2	11	12	9	0	12	11	11	11	8	87
(6) MutOpEq	-10	9	10	-2	-12	0	7	-1	-7	-10	-16
(7) Prune And Plant	-11	2	5	-8	-11	-7	0	-6	-7	-8	-51
(8) SS+E	-11	8	8	-2	-11	1	6	0	-7	-10	-18
(9) Static Depth Limit	-12	7	9	3	-11	7	7	7	0	-11	6
(10) Tarpeian	-10	10	10	7	-8	10	8	10	11	0	48
Effort	1	2	3	4	5	6	7	8	9	10	Total
(1) Double Tournament	0	12	12	12	-3	10	11	12	12	12	90
(2) Dyn. Depth Limit	-12	0	11	-5	-11	-8	-3	-6	-7	-11	-52
(3) DynOpEq	-12	-11	0	-9	-12	-12	-8	-10	-10	-12	-96
(4) FlatOpEq	-12	5	9	0	-12	-9	3	-5	-8	-11	-40
(5) HARM-GP	3	11	12	12	0	12	11	12	12	8	93
(6) MutOpEq	-10	8	12	9	-12	0	8	1	-4	-10	2
(7) Prune And Plant	-11	3	8	-3	-11	-8	0	-7	-9	-9	-47
(8) SS+E	-12	6	10	5	-12	-1	7	0	-3	-12	-12
(9) Static Depth Limit	-12	7	10	8	-12	4	9	3	0	-12	5
(10) Tarpeian	-12	11	12	11	-8	10	9	12	12	0	57

4.2.4 Results with Equivalent Computational Effort

It is often argued that unlike the absolute fitness performance, the reduction of computational effort required to complete an evolution is not a priority. Albeit the ever increasing power of computers can compensate up to a certain point, an abusive computational effort can still be an issue with prototyping, real-time applications, large datasets or complex simulations, continuous learning, or simply when a very large number of runs is needed.

In any case, the comparison of a method requiring ten times the computational effort of another is unfair, at least in practical terms. We argue that if a method requires ten times the effort of another, then a run of the former should be compared to the best among ten runs of the latter. We thus apply this reasoning and produce new results corrected for *equivalent effort*.

The methodology used is the following: for each problem, we first select the best performing method on the testing set. Then we compare its computational effort to other methods. If another method exhibits an effort n times smaller than the reference, we produce a total of $(n-1) \times 100$ additional runs. Subsequently, we gather the results in $100 \times (n - \lfloor n \rfloor)$ groups of $\lceil n \rceil$ runs and the remaining results in groups of $\lfloor n \rfloor$ runs, from which we pick the run that produced the best individual according to the training fitness. As a result, we still obtain a total of 100 runs (or meta-runs) for each method, using the same computational effort. No special treatment is applied to the methods exhibiting a greater effort.

The results are presented in Fig. 7 and in Table 4. Fig. 7 shows that the new Pareto fronts, corrected for equivalent effort, are closer to the origin and contain less methods. The methods appearing most often on the Pareto fronts are now limited to HARM-GP (12/12), Double Tournament (9/12), and SS+E (4/12). All other methods are either not present or appear only once. Table 4 also shows a drastic improvement from the previous section. As can be seen, with equivalent effort, HARM-GP and Double Tournament see their relative performance increase notably. This is expected, since these methods were also the ones using the least effort in Table 3. Interestingly, this performance improvement does not come with a size increase. On the contrary, they also increase their relative position with respect to size. This is reflected in Fig. 7, where they own most of the Pareto fronts.

4.3 Results by Problem Type

Following the overall picture reported in the previous section, we now present more detailed results. For each problem, we produce a summary table, and, for selected problems, we also provide a graphical view of the performance and size evolution. More specifically, we provide the median (or average) best-of-run performance on both the training and testing sets, the average accumulated size (computational effort), the average best-of-run solution size, and the average population bloat level at the last generation.

We draw performance vs. size plots to visually present results in a compact

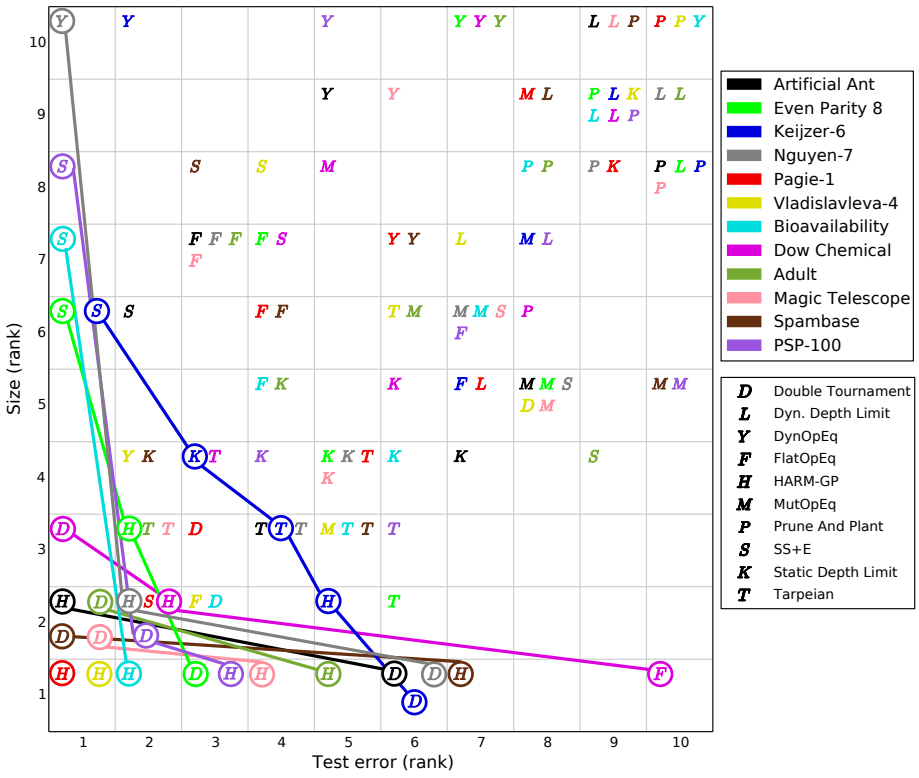


Figure 7: Global results corrected for equivalent effort on performance and size, for all problems and methods. The graph uses the rank of each method as a discrete measurement to position markers. The Pareto fronts are outlined for each problem.

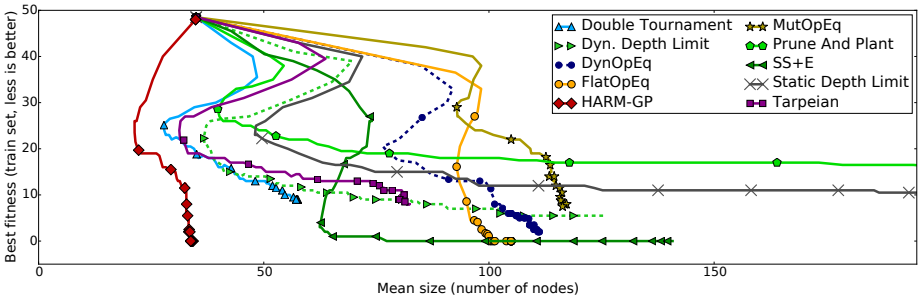
Table 4: Per method pairwise comparisons, corrected for equivalent effort, using a Wilcoxon signed-rank test with $p < 0.01$; a significant result counts for either $+1$ or -1.

Performance	1	2	3	4	5	6	7	8	9	10	Total
(1) Double Tournament	0	11	5	1	-2	9	12	0	2	3	41
(2) Dyn. Depth Limit	-11	0	-7	-9	-11	-4	2	-11	-8	-8	-67
(3) DynOpEq	-5	7	0	-2	-8	2	8	-4	-1	-3	-6
(4) FlatOpEq	-1	9	2	0	-4	6	9	-4	3	-1	19
(5) HARM-GP	2	11	8	4	0	10	12	5	6	6	64
(6) MutOpEq	-9	4	-2	-6	-10	0	6	-5	-6	-6	-34
(7) Prune And Plant	-12	-2	-8	-9	-12	-6	0	-10	-9	-11	-79
(8) SS+E	0	11	4	4	-5	5	10	0	4	2	35
(9) Static Depth Limit	-2	8	1	-3	-6	6	9	-4	0	-2	7
(10) Tarpeian	-3	8	3	1	-6	6	11	-2	2	0	20
Size	1	2	3	4	5	6	7	8	9	10	Total
(1) Double Tournament	0	11	10	8	-3	10	12	11	12	10	81
(2) Dyn. Depth Limit	-11	0	2	-10	-11	-9	-2	-7	-7	-10	-65
(3) DynOpEq	-10	-2	0	-10	-12	-10	-4	-8	-9	-10	-75
(4) FlatOpEq	-8	10	10	0	-9	2	10	2	-6	-6	5
(5) HARM-GP	3	11	12	9	0	12	12	12	11	9	91
(6) MutOpEq	-10	9	10	-2	-12	0	8	-1	-7	-10	-15
(7) Prune And Plant	-12	2	4	-10	-12	-8	0	-8	-10	-11	-65
(8) SS+E	-11	7	8	-2	-12	1	8	0	-7	-11	-19
(9) Static Depth Limit	-12	7	9	6	-11	7	10	7	0	-10	13
(10) Tarpeian	-10	10	10	6	-9	10	11	11	10	0	49

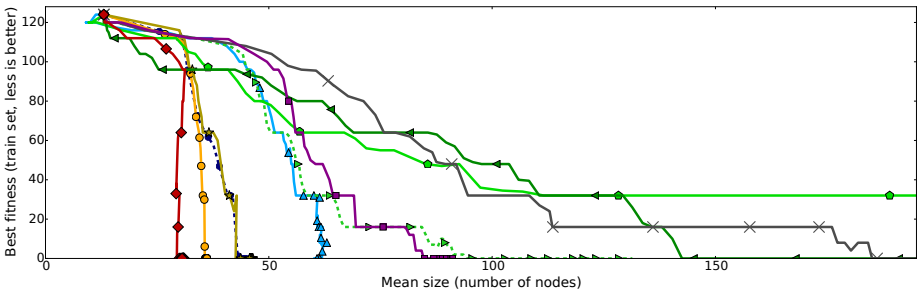
format. While quite different from the usual performance against generation or size against generation plots, these graphs are more informative since they combine two of our criteria into a single plot. Thus, at each generation, we begin by extracting the median (or average) best performance and the average mean solution size, and plotting this as a point on the graph. By connecting these points together, we can provide a visual cue of each method's behavior. For instance, a method that produces a line going straight down is able to improve performance without adding size. On the contrary, a left to right horizontal line indicates a lack of control over bloat.

Figures of this type have been used in previous works on bloat control [46, 58, 70]. However, in our plots, we add a third dimension by positioning the markers at intervals of 10 000 fitness evaluations. Thus, a method achieving a given performance improvement using a fewer number of markers can be seen as making more efficient progress. It should be noted that some markers may be superimposed, especially at the end of runs where the performance tends not to improve much, or may extend past the right edge of the graph in some cases.

Detailed results for all problems are also given in Appendix B (see B), along with a comprehensive description and analysis for each of them.



(a) Average best fitness vs. average mean solution size (Artificial Ant)



(b) Average best fitness vs. average mean solution size (Even Parity)

Figure 8: Fitness vs size results for Artificial Ant and Even Parity problems. Over 100 independent runs, graphs plot the average of the best fitness against the corresponding average mean solution size. Each marker accounts for 10 000 evaluations; some markers may overlap one another, or extend past the right edge of the graph.

4.3.1 Artificial Problems

Fig. 8 gives performance vs. size curves for the two classical Artificial Ant (planning) and Even Parity 8 (combinatorial logic) problems. The corresponding numerical results are also given in Table 5.

Regarding the Artificial Ant problem, SS+E obtains the best performance, with the highest success rate (percentage of runs achieving the best possible fitness), followed by FlatOpEq and HARM-GP where the success rate is about 30% lower. Even though the size of SS+E solutions are quite reasonable, its effort and bloat level are noticeably higher, mostly because of higher average individual size (not only the best-of-runs). This can be seen in Fig. 8(a) where even after reaching a near-perfect solution, SS+E seems to let the mean solution size grow considerably without much performance return. Among the other methods, HARM-GP exhibits a balanced behavior, with a third rank for fitness, a top rank for the size and computational effort, and a very low bloat level.

Results for Even Parity show the relative simplicity of this problem. However, it can be observed that HARM-GP is amongst the methods with the best

Table 5: Detailed results for artificial problems. Bold numbers indicate per column best results. When the best is not significantly better than others, according to a Wilcoxon signed-rank test at $p < 0.01$, these other results are also set in bold. Column *Fitness* ($\bar{g}^*_x$) is the average best-of-run fitness (less is better), column *Success Rate* is the percentage of runs that reached perfect fitness, column *Size* is the average best-of-run tree size, column *Effort* ($\bar{c}_x$) is the average computational effort, and column *Bloat* is the average population bloat level. The values in columns *Effort* and *Size* are relative to the smallest effort and size, respectively. The reference values needed to determine actual effort and size are given on the *Reference* line.

	Method	Fitness ($\bar{g}^*_x$)	Success Rate	Size ($\bar{s}^*_x$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Artificial Ant	Double Tournament	7.020	43	1.4	1.6	0.7
	Dyn. Depth Limit	7.550	43	2.3	2.6	1.8
	DynOpEq	6.150	47	2.8	6.9	2.8
	FlatOpEq	4.240	59	2.4	7.4	2.5
	HARM-GP	5.710	53	**1.0**	**1.0**	**-0.3**
	MutOpEq	7.530	41	3.1	3.7	3.5
	Prune And Plant	9.530	35	4.6	9.4	26.4
	SS+E	**1.190**	88	**1.4**	3.1	2.6
	Static Depth Limit	8.750	41	3.1	5.6	7.7
	Tarpeian	7.830	37	1.7	2.1	1.3
	Reference	-	-	*36*	*4 858 363*	-
Even Parity 8	Double Tournament	21.610	55	2.2	1.9	6.4
	Dyn. Depth Limit	10.220	63	2.8	3.3	10.6
	DynOpEq	**2.640**	90	1.5	3.2	**4.0**
	FlatOpEq	**3.520**	90	1.5	3.3	5.1
	HARM-GP	**6.400**	78	**1.0**	**1.0**	**3.7**
	MutOpEq	7.840	77	1.5	1.6	4.9
	Prune And Plant	25.960	40	5.8	11.9	94.4
	SS+E	**3.600**	85	3.7	4.4	14.9
	Static Depth Limit	13.680	55	3.7	5.9	21.9
	Tarpeian	9.530	66	2.6	2.6	5.2
	Reference	-	-	*30*	*4 454 749*	-

performance (not significantly different from DynOpEq and FlatOpEq), while also producing the smallest solutions with the least effort, as can be seen on Fig. 8(b).

4.3.2 Symbolic Regression

Table 6 presents results for the four artificial symbolic regression problems used in this analysis, namely Keijzer-6, Nguyen-7, Pagie-1, and Vladislavleva-4. Fig. 9 also shows performance vs. size plots for these problems.

Performance results show that SS+E is the top performer three times out of four. However, other methods such as DynOpEq, FlatOpEq, and HARM-GP are generally quite close to this top result, when not statistically equivalent (as with Pagie-1). On the size aspect, however, HARM-GP outperforms all other methods. For instance, in Vladislavleva-4, DynOpEq obtains a better RMSE on the test set than HARM-GP (0.149 vs. 0.165), but at the expense of a solution *17 times larger*. In Keijzer-6, HARM-GP obtains a RMSE of 0.021, whereas SS+E obtains a RMSE of 0.011 but with individuals about five times larger.

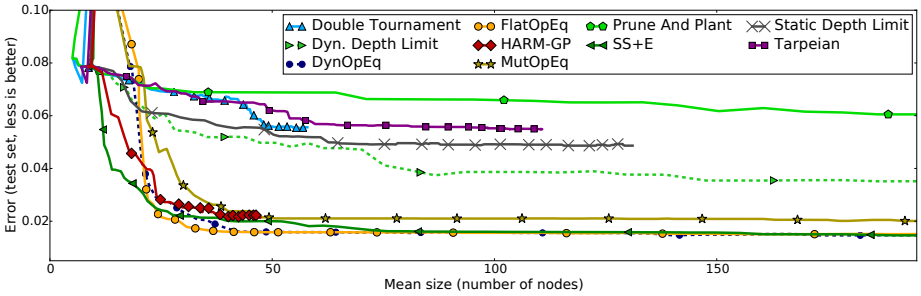
On the other hand, some methods produce very small but highly unfit solutions. For instance, Double Tournament seems to have difficulties with these problems. Indeed, while always very small, the solutions it produces are also consistently among the worst in terms of fitness. Part of the answer may be found in the Vladislavleva-4 results. In this problem, Double Tournament completely crushes the population, so that only very small individuals (a few nodes) remain, perforce leading to poor performance. As one can see, the bloat level obtained is *negative*, because the final mean size is smaller than the initial mean size. In this particular problem, there is simply no improvement over the whole evolution! Although to a lesser extent, the same issue arises with the other regression problems.

Regarding the computational effort, one can see that, as expected, DynOpEq and FlatOpEq are quite demanding. A similar ascertainment can be made, to a lesser extent, about SS+E and Dynamic Limits. It is interesting to see that many of the bloat control methods actually require *more* computational power than a standard GP run (with a static limit). Among the top five methods in performance, HARM-GP is always the method using the less effort, often by an order of magnitude.

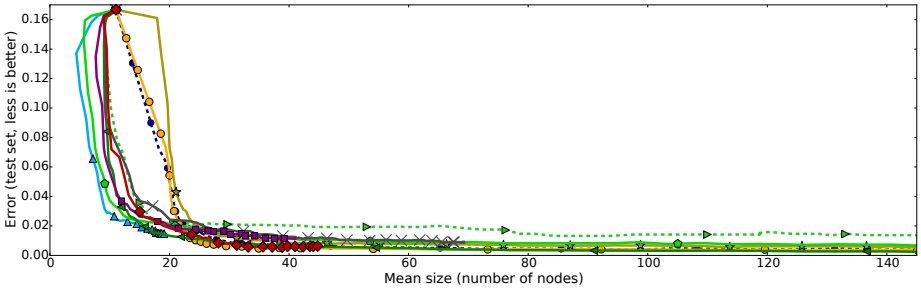
Overall, HARM-GP is always in the top five for performance, along with the other *OpEq and SS+E. It is also among the methods producing the smallest solutions, especially when we consider the performance of Double Tournament and Tarpeian, which are technically superior to HARM-GP regarding the size aspect, but with a poor fitness value. In all cases, HARM-GP clearly offers one of the best trade-off between size and performance. This is confirmed by the bloat level indicator, where HARM-GP obtains the best results (putting aside the methods with performance issues), except in Pagie-1 where it is slightly outperformed in this aspect by FlatOpEq.

Table 6: Detailed results for artificial symbolic regression problems. Bold numbers indicate per column best results. When the best is not significantly better than others, according to a Wilcoxon signed-rank test at $p < 0.01$, these other results are also set in bold. Columns *Train Error* ($\bar{f}_x^*$) and *Test Error* ($\bar{g}_x^*$) are the RMSE performances as measured on the training and testing set (less is better), respectively. See Table 5 for a description of the other columns.

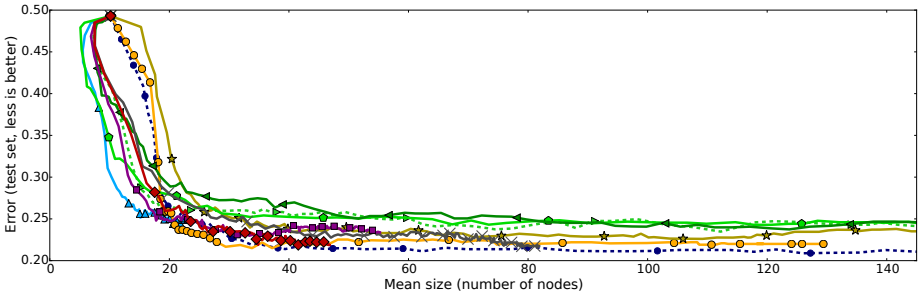
	Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Keijzer-6	Double Tournament	0.032	0.056	**1.0**	**1.1**	3.9
	Dyn. Depth Limit	0.009	0.035	24.1	18.6	124.1
	DynOpEq	**0.004**	0.016	20.5	29.6	109.3
	FlatOpEq	**0.004**	0.015	9.2	11.5	38.3
	HARM-GP	0.008	0.021	**1.1**	**1.0**	**3.2**
	MutOpEq	0.008	0.020	4.2	3.5	16.7
	Prune And Plant	0.035	0.060	12.9	10.9	59.8
	SS+E	**0.004**	**0.011**	5.2	5.4	45.9
	Static Depth Limit	0.019	0.049	2.0	2.5	9.3
	Tarpeian	0.024	0.055	1.8	2.0	8.2
	Reference	-	-	*69*	*5 439 003*	-
Nguyen-7	Double Tournament	0.010	0.014	**1.0**	**1.0**	**0.7**
	Dyn. Depth Limit	0.008	0.013	23.3	12.7	2.9
	DynOpEq	0.001	0.005	30.1	32.9	51.7
	FlatOpEq	0.002	0.003	13.2	13.0	4.9
	HARM-GP	0.003	0.006	2.8	2.2	3.2
	MutOpEq	0.003	0.005	7.8	5.6	8.2
	Prune And Plant	0.003	0.010	25.8	15.8	32.2
	SS+E	**0.000**	**0.001**	10.2	8.4	27.9
	Static Depth Limit	0.005	0.009	2.7	3.2	3.8
	Tarpeian	0.007	0.011	1.9	1.7	1.7
	Reference	-	-	*24*	*2 342 961*	-
Pagie-1	Double Tournament	0.064	0.252	**1.0**	**1.0**	**0.2**
	Dyn. Depth Limit	0.051	0.236	23.6	14.8	59.1
	DynOpEq	**0.040**	**0.207**	24.6	28.7	68.4
	FlatOpEq	**0.047**	**0.220**	5.0	6.6	3.1
	HARM-GP	0.047	**0.219**	2.2	1.9	3.6
	MutOpEq	0.050	0.235	7.8	5.5	11.4
	Prune And Plant	0.060	0.239	16.4	10.6	35.5
	SS+E	**0.048**	**0.205**	7.7	6.5	29.8
	Static Depth Limit	**0.042**	**0.217**	2.9	3.3	6.4
	Tarpeian	0.051	0.234	2.0	1.9	4.2
	Reference	-	-	*30*	*2 741 243*	-
Vladislavleva-4	Double Tournament	0.194	0.199	**1.0**	**1.0**	**-0.7**
	Dyn. Depth Limit	0.165	0.180	39.2	30.0	58.2
	DynOpEq	**0.116**	**0.149**	56.7	87.0	113.3
	FlatOpEq	**0.118**	**0.150**	23.1	34.1	40.8
	HARM-GP	0.137	0.165	3.3	3.2	**4.1**
	MutOpEq	0.138	0.163	23.0	19.3	42.2
	Prune And Plant	0.194	0.200	**10.8**	**7.8**	**-0.8**
	SS+E	0.156	0.171	12.8	14.0	35.4
	Static Depth Limit	0.133	0.162	4.7	6.2	**9.6**
	Tarpeian	0.174	0.192	2.7	3.4	**2.9**
	Reference	-	-	*22*	*1 565 337*	-



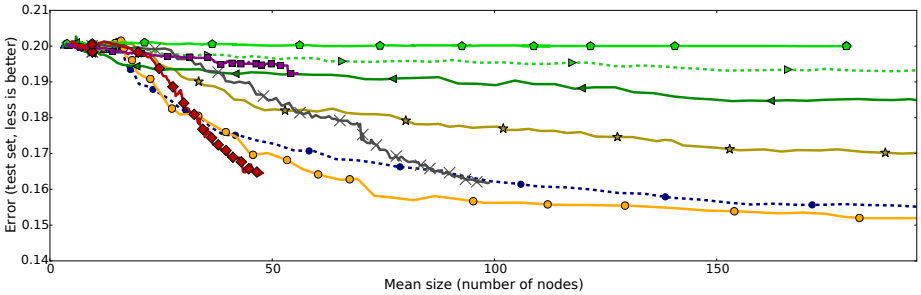
(a) Median best RMSE performance vs. average mean solution size (Keijzer-6)



(b) Median best RMSE performance vs. average mean solution size (Nguyen-7)



(c) Median best RMSE performance vs. average mean solution size (Pagie-1)



(d) Median best RMSE performance vs. average mean solution size (Vladislavleva-4)

Figure 9: Performance vs size results for the artificial symbolic regression problems: Keijzer-6, Nguyen-7, Pagie-1, and Vladislavleva-4. Over 100 independent runs, these graphs plot the median of the best Root Mean Square Error (RMSE) performance, against the corresponding average mean solution size. Each marker accounts for 10 000 evaluations; some markers may overlap one another, or extend past the right edge of the graph.

Table 7: Detailed results of real-life symbolic regression problems. Bold numbers indicate per column best results. When the best is not significantly better than others according to a Wilcoxon signed-rank test at $p < 0.01$, these other best results are also set in bold. See Table 6 for a description of columns.

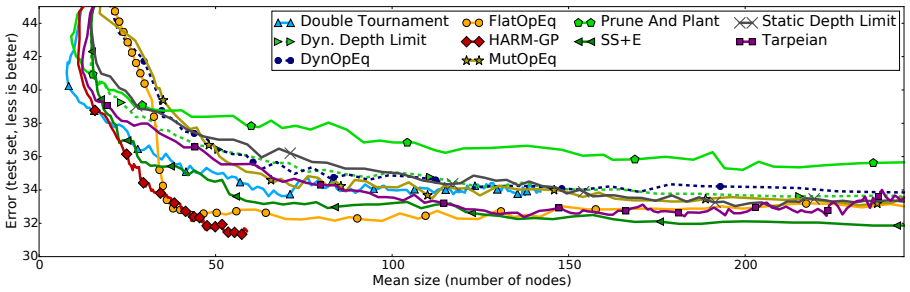
	Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Bioavailability	Double Tournament	27.874	33.903	1.6	2.0	6.0
	Dyn. Depth Limit	27.076	36.457	31.8	27.7	143.5
	DynOpEq	**24.581**	46.796	34.9	58.9	148.9
	FlatOpEq	27.829	**33.019**	3.7	4.7	5.0
	HARM-GP	28.691	**31.511**	**1.0**	**1.0**	**1.8**
	MutOpEq	27.273	35.539	7.6	6.6	23.4
	Prune And Plant	28.333	36.449	14.9	10.1	41.7
	SS+E	28.707	**31.332**	7.8	6.1	44.7
	Static Depth Limit	25.690	33.982	4.3	5.8	20.8
	Tarpeian	26.305	34.847	3.3	4.2	26.8
	Reference	-	-	*100*	*6 280 874*	-
Dow Chemical	Double Tournament	0.221	0.234	3.9	**1.0**	-0.6
	Dyn. Depth Limit	0.182	0.257	185.4	27.7	63.4
	DynOpEq	**0.159**	0.228	198.6	52.6	65.8
	FlatOpEq	0.268	0.271	**1.0**	17.9	0.6
	HARM-GP	0.181	**0.201**	7.9	1.7	1.0
	MutOpEq	0.175	**0.195**	59.5	10.2	15.9
	Prune And Plant	0.284	0.274	**11.5**	1.7	**-0.8**
	SS+E	0.180	**0.193**	38.1	5.6	18.7
	Static Depth Limit	**0.164**	0.231	19.7	5.3	5.6
	Tarpeian	0.184	0.222	12.3	2.9	2.7
	Reference	-	-	*7*	*2 199 072*	-

4.3.3 Real-life Symbolic Regression

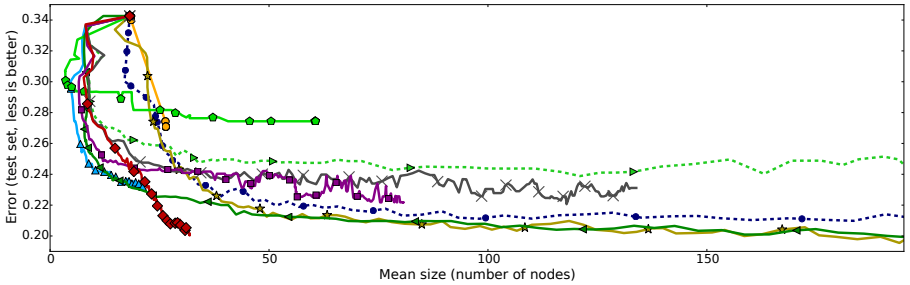
The detailed results for the two real-life regression problems, namely Bioavailability and Dow Chemical, are summarized in Table 7 and illustrated in Fig. 10. In this figure, along with median performance vs. average size plots, we produce two other graphs for Bioavailability. The first (Fig. 10(c)) provides a boxplot of the best-of-run individual size, while the last one (Fig. 10(d)) is a boxplot for the final bloat level.

When we consider these results, we can note that HARM-GP is not the best performing method in terms of training error. However, the error on the test set depicts an entirely different situation, where HARM-GP is second – actually not significantly different from the best method. In other words, as discussed in Sec. 4.1, the good performance of other methods on the training set is due to overfitting and does not represent the actual performance of these methods on real-world applications. Besides, HARM-GP again produced individuals up to seven times smaller than all other methods with a non-significant difference in test fitness, which is interesting in the case of a real-world problem, where we would like to analyze the solution obtained. Among all methods, HARM-GP also uses the smallest effort, and obtains the best score regarding the bloat level.

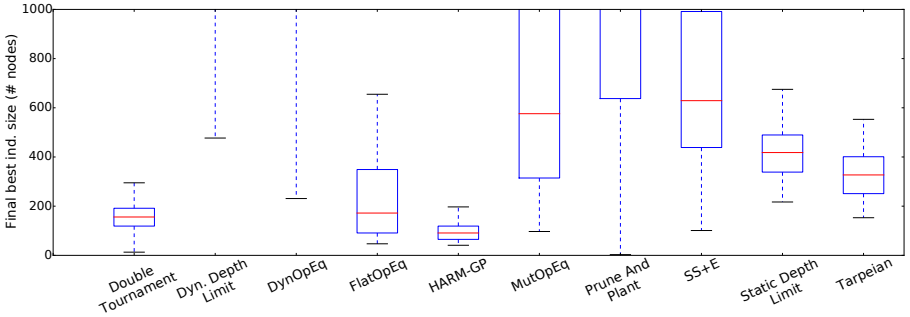
These observations also apply to the Dow Chemical problem, where despite being surpassed by other techniques such as Static Depth Limit in terms of training RMSE, HARM-GP scores among the best performers according to the



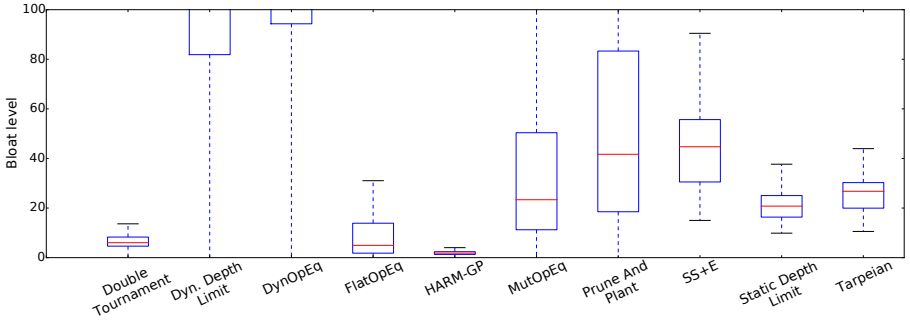
(a) Median RMSE performance vs. average mean solution size (Bioavailability)



(b) Median RMSE performance vs. average mean solution size (Dow Chemical)

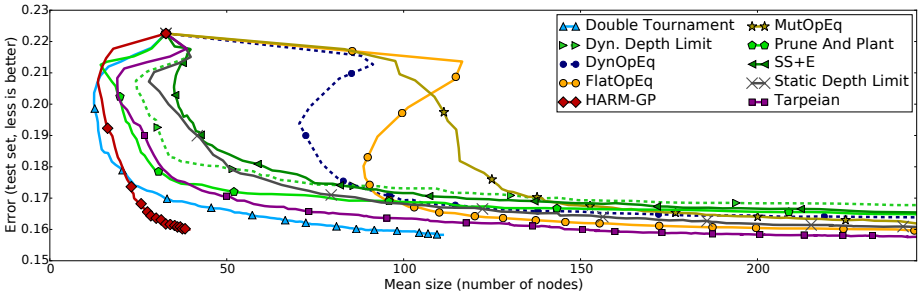


(c) Best-of-run size boxplot (Bioavailability)

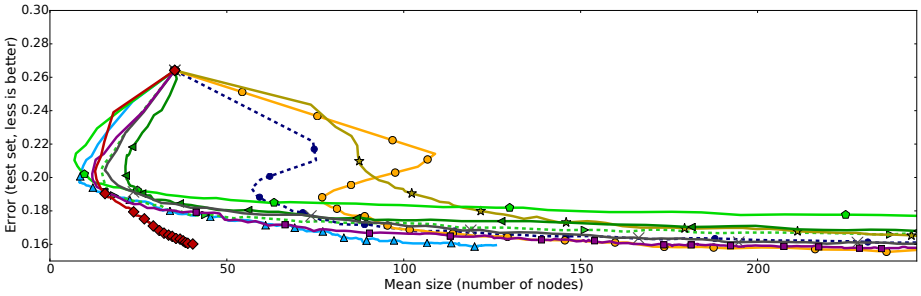


(d) Bloat level boxplot, according to Eq. 1, less is better (Bioavailability)

Figure 10: Results for the Bioavailability and Dow Chemical symbolic regressions: (a) and (b) present the median over 100 runs of the RMSE on the testing set of the individual achieving the best performance on the training set, according to the mean size of the population, each marker accounting for 10 000 evaluations; (c) boxplot of the best-of-run individual size for Bioavailability; (d) boxplot of the bloat level at the last generation, again for Bioavailability.



(a) Median best classification error performance vs. mean solution size (Adult)



(b) Median best classification error performance vs. mean solution size (Magic Telescope)

Figure 11: Performance vs size results for the Adult and Magic classification problems. Over 100 independent runs, these graphs plot the median of the best classification error performance, against the corresponding average mean solution size. Each marker accounts for 10 000 evaluations; some markers may overlap one another, or extend past the right edge of the graph.

RMSE on the testing set (again not significantly worse than the best method). The only two methods able to reach a similar RMSE on the testing set also produce solutions four and six times larger, and use at least three times the HARM-GP computational effort to do so. In this problem, one could also note the poor performance of FlatOpEq, caused by the tremendous number of evaluations it needs to start an evolution. Even when given twice the number of evaluations of other techniques, FlatOpEq is barely able to complete more than a few generations, and thus accordingly obtained a relatively high RMSE.

4.3.4 Classification

Finally, we present detailed results for the four classification problems used. The required binary classification is obtained by simply using a fixed threshold (at 0) on a real-valued, regression output. Table 8 shows numerical results for all problems, and Fig. 11 the evolution of size and performance for the Adult and Magic Telescope datasets.

In this category, the performance gap between the methods substantially

Table 8: Detailed results of real-world classification problems conducted with the different bloat control methods. Bold numbers indicate the best result for each column. If the best result is not significantly better than others according to a Wilcoxon signed-rank test at $p < 0.01$, these other best results are also set in bold. Columns *Train Error* ($\bar{f}_x^*$) and *Test Error* ($\bar{g}_x^*$) are the classification error rates as measured on the training and testing set (less is better), respectively. See Table 5 for a description of the other columns.

	Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Adult	Double Tournament	0.158	**0.158**	1.9	2.3	2.9
	Dyn. Depth Limit	0.164	0.164	20.8	17.0	40.0
	DynOpEq	0.159	0.160	26.5	41.1	52.4
	FlatOpEq	0.157	**0.158**	12.3	19.4	13.6
	HARM-GP	0.161	0.160	**1.0**	**1.0**	**0.2**
	MutOpEq	0.159	0.159	7.2	8.3	14.1
	Prune And Plant	0.162	0.162	17.0	11.1	36.2
	SS+E	0.164	0.162	5.8	6.4	18.0
	Static Depth Limit	**0.156**	**0.158**	6.6	7.9	13.8
	Tarpeian	**0.156**	**0.158**	4.0	4.5	7.2
	Reference	-	-	*65*	*4 630 486*	-
Magic Telescope	Double Tournament	0.149	0.159	1.8	2.1	4.0
	Dyn. Depth Limit	0.144	0.161	22.3	20.0	55.9
	DynOpEq	**0.137**	**0.155**	19.4	40.2	52.4
	FlatOpEq	0.141	**0.153**	7.8	15.3	14.7
	HARM-GP	0.153	0.160	**1.0**	**1.0**	**0.3**
	MutOpEq	0.147	0.163	7.2	8.8	16.2
	Prune And Plant	0.156	0.167	14.9	13.3	46.8
	SS+E	0.153	0.159	7.7	7.4	26.9
	Static Depth Limit	**0.139**	**0.157**	5.8	7.7	15.5
	Tarpeian	0.140	**0.158**	3.3	4.4	7.8
	Reference	-	-	*78*	*4 858 638*	-
Spambase	Double Tournament	0.064	**0.077**	1.5	2.6	3.2
	Dyn. Depth Limit	0.068	0.085	12.5	16.0	37.3
	DynOpEq	**0.058**	0.081	11.2	31.3	27.5
	FlatOpEq	0.063	0.080	5.5	12.1	10.4
	HARM-GP	0.071	0.080	**1.0**	**1.0**	**0.3**
	MutOpEq	0.069	0.087	4.8	7.3	12.6
	Prune And Plant	0.072	0.085	16.0	19.1	45.1
	SS+E	0.068	0.080	12.0	11.0	41.2
	Static Depth Limit	**0.059**	**0.079**	3.7	6.6	10.8
	Tarpeian	0.064	0.080	2.1	3.6	5.2
	Reference	-	-	*98*	*5 210 493*	-
PSP100	Double Tournament	0.246	**0.267**	**1.0**	1.7	2.7
	Dyn. Depth Limit	0.251	0.272	8.4	10.5	40.7
	DynOpEq	**0.238**	0.269	10.2	26.4	42.0
	FlatOpEq	0.243	0.270	4.4	10.2	12.7
	HARM-GP	0.248	0.268	**1.0**	**1.0**	**0.8**
	MutOpEq	0.249	0.278	4.2	6.6	16.6
	Prune And Plant	0.253	0.276	10.1	11.0	52.2
	SS+E	0.248	**0.265**	9.3	7.8	53.1
	Static Depth Limit	0.242	0.269	2.4	4.3	11.0
	Tarpeian	0.248	0.271	1.7	2.7	5.7
	Reference	-	-	*98*	*5 461 516*	-

shrinks compared to the previously analyzed problems. Double Tournament obtains good results and is usually among the best performing methods, while HARM-GP does not score in the first place when considering the classification error rate. However, this must be put into context. The accuracy difference between HARM-GP and the best method is around 0.002 for Adult, and 0.003 for Spambase. Considering the size of these datasets, it thus implies that HARM-GP misses about 32 samples more than the best method over the 16 281 Adult test set instances. Similarly, the median difference between HARM and the best performing alternative in Spambase represents *four* missed test instances over 1380. This is a very minimal loss, especially considering that the solution size is reduced by a factor of 2 and 1.5, respectively.

Regarding the size of the produced solutions and the computational effort, HARM-GP consistently produces the smallest solutions with the lesser effort. The bloat level also reflects this good behavior, with HARM-GP runs being essentially bloat-less according to this indicator.

The trade-off between performance, size and effort is clearly illustrated in Fig. 11. The figure indicates that for almost every size, HARM-GP generates a solution better than any other method. This can also be seen backwards: at almost every performance level, the solutions produced by HARM-GP are the smallest among all methods. The large space between markers at the beginning of the HARM-GP curves is also interesting. Because these markers are evenly spaced regarding the number of evaluations, it implies that HARM-GP is able to reach a given performance faster than most of the other methods.

5 Discussion

Results show that HARM-GP is consistently among the best methods regarding the size of the produced individuals. Moreover, when it is surpassed in this aspect, it is usually by other methods obtaining a very poor fitness rank (such as Double Tournament on Keijzer-6, Pagie-1 or Nguyen-7, or FlatOpEq on Dow Chemical).

Of course, producing small individuals is not enough, and one could note that it is not a difficult task if they are all unfit. However, in the light of the results presented herein, this is clearly not the case for HARM-GP. Fig. 6 shows that the performance loss relative to the best method is always less than 10%, apart for the Vladislavleva-4 regression, which is explained by the metric used. Actually, when taking into account statistical significance, HARM-GP is able to reach the first performance rank (tied with other methods) four times over the twelve problems.

While this is already a good result, it still means that other methods are often able to obtain better solutions. Indeed, one could argue that SS+E exhibits a better performance, reaching the first rank six times on the testing set. However, our results point out that the fundamental reason for this performance difference should be sought in the computational effort used. When we assign the same computing resources to each method, as described in Sec. 4.2.4, results

change drastically: HARM-GP then holds (alone or tied with others) the first performance rank *nine* times out of twelve, and reaches another second place, without affecting its ability to produce small solutions. Table 4 shows that with equivalent effort, HARM-GP is able to surpass on average *all other methods* on *both* performance and size, and this despite the fact that the equivalent effort has actually *degraded* its rank on the Spambase problem[3]. Moreover, as we did not penalize methods using more computational effort than the best performing one, the effort used by HARM-GP is still often smaller than that of other methods. While this set of results has to be treated with caution, since it partly depends on the problem set used, it remains an interesting ascertainment about HARM-GP's level of performance.

These results clearly demonstrate the importance of taking into account the computational effort when comparing two or more GP heuristics such as bloat control methods. The equivalent computational effort gives a different picture of the situation, and not only for HARM-GP. Indeed, Double Tournament and Tarpeian also achieve interesting performance gains in this context. This result is not surprising: as Silva already noted [54], the greater the effort allocated to a method, the more it can explore the search space and perform well.

5.1 Sensitivity Analysis of Parameters

The set of parameters given in Sec. 3.9.1 was used for all of the aforementioned problems, with good results. However, it is important to see the effect of each parameter, especially to ensure that there is no breakpoint in the parameter space, where the performance would suddenly drop after a slight change in the settings. Detailed results on sensitivity analysis made on the Bioavailability problem are given in Appendix A (see A).

Results show that while a different parametrization does induce some changes in the output results, it does not modify the typical behavior of HARM-GP. The size curve simply increases to a different level than the reference configuration, where it stabilizes itself. Similarly, the train RMSE shows a different final result for each parametrization, but nothing fundamentally changes. This is an important characteristic, because it indicates that HARM-GP can be tuned step-by-step, without the risk of a small change having a serious negative impact on the method's performance.

6 Conclusion

Overall, results show that HARM-GP is able to strongly restrain bloat emergence, producing smaller solutions on a broad range of problems, while mostly preserving performance, especially when allocated equivalent computing resources to those required by the method reaching the best performance. In

[3] As we select the runs only based on their fitness on the training set, we actually increase the likelihood of overfitting, and thus impact test error. Using a validation set would prevent this and allow better generalization results.

particular, amongst all tested methods, HARM-GP was the only one that systematically appeared on every fitness-size Pareto front, regardless of the problem. Furthermore, for all tested problems, HARM-GP always performed well, albeit without necessarily being the best, contrary to other methods that exhibited their limitations on at least one problem, either in terms of performance or size, or both.

The probabilistic nature of HARM-GP plays a very important role in these results: it allows sampling of a non-bloated target distribution from the arbitrary natural distribution that stems from the application of genetic operators. The fact that fitness is only used to define the target (choose the right distribution to attain), not to reach it (generate the wanted distribution with standard variation operators applied to the current population), is another key characteristic of the algorithm. It ensures the independence of HARM-GP with respect to fitness space, including the case of multi-objective fitness, which is essential to avoid inducing an unwanted elitist bias in the reproduction process. Considering these key characteristics, HARM-GP and its underlying concept could be applied to other evolutionary algorithms that utilize variable length genotypes.

In addition to introducing HARM-GP, this paper also features a comprehensive review of many of the most well known bloat control methods, highlighting both their strengths and weaknesses. The full results are given in an online appendix (see B) that compares the ten methods over twelve problems. It is suggested that this review could constitute a future reference base for various GP benchmark problems which have not yet been thoroughly tested, because of their recent introduction in [73].

In this paper, we have focused on the bloat control property of HARM-GP. A future work of interest would be to focus more on its ability to limit overfitting (using classification and regression problems). We think that by slightly changing the internal mechanisms of the algorithm, for instance by changing the cutoff point definition to the individual that generalizes the most, regardless of its size, one could explore a promising avenue to tackle the important, currently open issue of overfitting in genetic programming.

Open source implementations of HARM-GP are available in the DEAP[4] Python library [19] and in the Open BEAGLE[5] C++ framework [21].

acknowledgements

We acknowledge the financial support of the NSERC (Canada) and FRQ-NT (Québec), and the access to supercomputing facilities of Calcul Québec / Compute Canada. We also thank Annette Schwerdtfeger for proofreading the manuscript.

[4] `http://deap.gel.ulaval.ca`
[5] `http://beagle.gel.ulaval.ca`

References

[1] E. Alfaro-Cid, A. Esparcia-Alcázar, K. Sharman, F. Fernandez de Vega, and J. J. Merelo. Prune and plant: a new bloat control method for genetic programming. In *Proc. of the international conference on Hybrid Intelligent Systems (HIS)*, pages 31–35, 2008.

[2] E. Alfaro-Cid, J. J. Merelo, F. Fernandez de Vega, A. Esparcia-Alcazar, and K. Sharman. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation*, 18(2):305–332, 2010.

[3] E. Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[4] N. Amil, N. Bredeche, C. Gagné, S. Gelly, M. Schoenauer, and O. Teytaud. A statistical learning perspective of genetic programming. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 327–338, 2009.

[5] J. Bacardit, M. Stout, N. Krasnogor, J. D. Hirst, and J. Blazewicz. Coordination number prediction using learning classifier systems: performance and interpretability. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 247–254, 2006.

[6] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1997.

[8] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: reducing bloat using SPEA2. In *Proc. of the Congress on Evolutionary Computation (CEC)*, volume 1, pages 536–543, 2001.

[9] T. Blickle and L. Thiele. Genetic programming and redundancy. In *Genetic Algorithms within the Framework of Computation (Workshop at KI-94)*, 1994.

[10] R. Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jiřina, J. Klaschka, E. Kotrč, P. Savickỳ, S. Towers, et al. Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 516(2):511–528, 2004.

[11] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001.

[12] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, 2010.

[13] S. Dignum and R. Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1588–1595, 2007.

[14] S. Dignum and R. Poli. Crossover, sampling, bloat and the harmful effects of size limits. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 158–169, 2008.

[15] S. Dignum and R. Poli. Operator equalisation and bloat free GP. In *Proc. of the European conference on Genetic Programming (EuroGP)*, pages 110–121, 2008.

[16] K. Dimitrios, K. Aigli, T. Konstantinos, L. Spiros, T. Athanasios, and M. Seferina. Where we stand, where we are moving: Surveying computational techniques for identifying miRNA genes and uncovering their regulatory role. *Journal of Biomedical Informatics*, 46(3):563–573, 2013.

[17] P. G. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144, 2010.

[18] J. Fitzgerald, R. Azad, and C. Ryan. Bootstrapping to reduce bloat and improve generalisation in genetic programming. In *Companion Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 141–142, 2013.

[19] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 2171-2175(13), 7 2012.

[20] M. A. Franco, N. Krasnogor, and J. Bacardit. Post-processing operators for decision lists. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, GECCO '12, pages 847–854, New York, NY, USA, 2012.

[21] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 4 2006.

[22] S. Gelly, O. Teytaud, N. Bredeche, and M. Schoenauer. A statistical learning theory approach of bloat. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1783–1784, 2005.

[23] L. Guo, D. Rivero, J. Dorado, C. R. Munteanu, and A. Pazos. Automatic feature extraction using genetic programming: An application to epileptic EEG classification. *Expert Systems with Applications*, 38(8):10425–10436, 2011.

[24] K. Harries and P. Smith. Code growth, explicitly defined introns and alternative selection schemes. *Evolutionary Computation*, 6(4):346–364, 1998.

[25] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 70–82, 2003.

[26] K. Kinnear Jr. Evolving a sort: Lessons in genetic programming. In *Proc. of the IEEE International Conference on Neural Networks (ICNN)*, pages 881–888, 1993.

[27] D. Kinzett, M. Johnston, and M. Zhang. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence*, 2(4):151–168, 2009.

[28] A. Kordon, G. Smits, E. Jordaan, and E. Rightor. Robust soft sensors based on integration of genetic programming, analytical neural networks, and support vector machines. In *Proc. of the IEEE International Conference on E-Commerce Technology*, volume 1, 2002.

[29] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992.

[30] J. R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251, 2010.

[31] W. Langdon and R. Poli. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer London, 1998.

[32] W. Langdon, T. Soule, R. Poli, and J. Foster. The evolution of size and shape. In *Advances in Genetic Programming III*, chapter 8, pages 163–190. MIT Press, 1999.

[33] W. B. Langdon and R. Poli. *Foundations of genetic programming*. Springer, 2002.

[34] S. M. Lee, D. S. Kim, J. H. Kim, and J. S. Park. Spam detection using feature selection and parameters optimization. In *Proc. of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 883–888, Washington, DC, USA, 2010.

[35] S. Luke and L. Panait. Fighting bloat with nonparametric parsimony pressure. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, pages 411–421, 2002.

[36] S. Luke and L. Panait. Lexicographic parsimony pressure. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 829–836, 2002.

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

[38] M. G. Madden. On the classification performance of TAN and general Bayesian networks. *Knowledge-Based Systems*, 22(7):489–495, 2009.

[39] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, et al. Genetic programming needs better benchmarks. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 791–798, 2012.

[40] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 121–132, 2000.

[41] P. Nordin, W. Banzhaf, et al. Complexity compression and evolution. In *Proc. of the International Conference on Genetic Algorithms (ICGA)*, pages 310–317, 1995.

[42] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[43] L. Pagie and P. Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary computation*, 5(4):401–418, 1997.

[44] L. Panait and S. Luke. Alternative bloat control methods. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 630–641, 2004.

[45] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[46] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 204–217, 2003.

[47] R. Poli. Covariant tarpeian method for bloat control in genetic programming. In *Genetic Programming Theory and Practice VIII*, pages 71–89. Springer, 2011.

[48] R. Poli, W. B. Langdon, and S. Dignum. On the limiting distribution of program sizes in tree-based genetic programming. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 193–204, 2007.

[49] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. freely available at `http://www.gp-field-guide.org.uk`, 2008.

[50] R. Poli and N. F. McPhee. Exact schema theorems for gp with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, 2001.

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

[52] R. Poli and N. F. McPhee. Parsimony pressure made easy. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1267–1274, 2008.

[53] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo method*. Wiley-Interscience, 2008.

[54] S. Silva. Reassembling operator equalisation: a secret revealed. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1395–1402, 2011.

[55] S. Silva and E. Costa. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009.

[56] S. Silva and S. Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 159–170, 2009.

[57] S. Silva, S. Dignum, and L. Vanneschi. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines*, 13(2):197–238, 2011.

[58] S. Silva and L. Vanneschi. Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1115–1122, 2009.

[59] S. Silva and L. Vanneschi. The importance of being flat – studying the program length distributions of operator equalisation. In *Genetic Programming Theory and Practice IX*, pages 211–233. Springer, 2011.

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

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

[62] L. Spector and T. Helmuth. Uniform linear transformation with repair and alternation in genetic programming. In *Genetic Programming Theory and Practice XI*. Springer, 2013.

[63] M. Streeter and L. A. Becker. Automated discovery of numerical approximation formulae via genetic programming. *Genetic Programming and Evolvable Machines*, 4(3):255, 2003.

[64] W. Tackett. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, University of Southern California, 1994.

[65] A. Teller and M. Veloso. Program evolution for data mining. *International Journal of Expert Systems Research and Applications*, 8:213–236, 1995.

[66] M. Tomassini, L. Vanneschi, J. Cuendet, and F. Fernández. A new technique for dynamic size populations in genetic programming. In *Proc. of the Congress on Evolutionary Computation (CEC)*, volume 1, pages 486–493, 2004.

[67] L. Trujillo, E. Naredo, and Y. Martínez. Preliminary study of bloat in genetic programming with behavior-based search. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, pages 293–305. Springer, 2013.

[68] L. Trujillo, S. Silva, P. Legrand, and L. Vanneschi. An empirical study of functional complexity as an indicator of overfitting in genetic programming. In *Proc. of the European Conference on Genetic Programming (EuroGP)*, pages 262–273, 2011.

[69] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

[70] L. Vanneschi, M. Castelli, and S. Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 877–884, 2010.

[71] E. J. Vladislavleva, G. F. Smits, and D. Den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[72] P. A. Whigham and G. Dick. Implicitly controlling bloat in genetic programming. *IEEE Transactions on Evolutionary Computation*, 14(2):173–190, 2010.

[73] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.

[74] L. Wilkinson, A. Anand, and D. N. Tuan. CHIRP: a new classifier based on composite hypercubes on iterated random projections. In *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 6–14, 2011.

[75] J. Yu, J. Yu, A. A. Almal, S. M. Dhanasekaran, D. Ghosh, W. P. Worzel, and A. M. Chinnaiyan. Feature selection and molecular classification of cancer using genetic programming. *Neoplasia*, 9(4):292, 2007.

[76] M. Zhang and P. Wong. Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines*, 9(3):229–255, 2008.

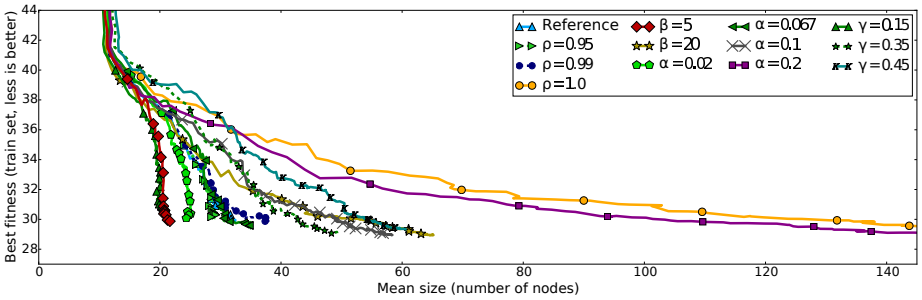
A Sensitivity Analysis of HARM-GP Parameters

This appendix conducts a parameter sensitivity analysis of HARM-GP. A total of twelve configurations were tested, in addition to the reference (using the recommended parameters), each configuration being run 30 times. Among these configurations, α was varied four times, β two times, and γ and ρ three times. This analysis was performed over the Bioavailability regression problem. The fitness and size results are presented in Fig. 12 and Table 9.

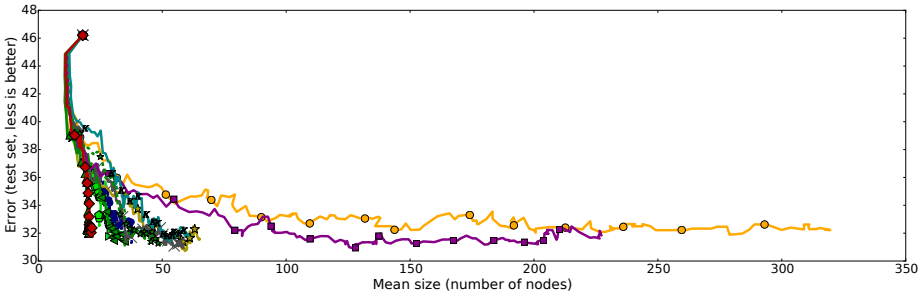
By looking at each parameter individually, we can first observe that the fitness percentage of the cutoff point (ρ) does not have much effect unless it is set to 100% – meaning that we always use the best individual as the cutoff point. In this case, the mean size considerably increases, and the training RMSE (Root Mean Squared Error) improves significantly (out of graph). This performance increase is due to an overfitting of the training set, however, and the test RMSE remains about the same as the reference. Varying β slightly changes the size distribution, but not significantly. Sweeping α from $\frac{1}{20}$ to $\frac{1}{5}$ can considerably change the size of the produced solutions, and allow more individuals beyond the cutoff point. Tuning γ produces slightly larger individuals with a small gain in fitness.

Table 9: Detailed results of experiments conducted with the different bloat control methods for the Bioavailability problemBold numbers indicate per column best results. When the best is not significantly better than others according to a Wilcoxon signed-rank test at $p < 0.01$, these other best results are also set in bold. See Table 6 for a description of columns. Note that in this specific case, no configuration was found significantly superior on the test set.

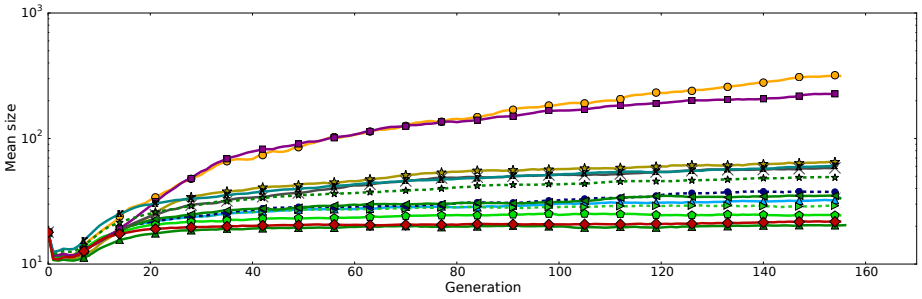
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Reference (3.9.1)	29.643	31.879	1.8	1.4	0.6
$\rho = 0.95$	29.956	31.608	1.7	1.4	0.5
$\rho = 0.99$	29.604	31.795	2.0	1.6	0.9
$\rho = 1.0$	**27.368**	32.243	8.6	7.8	13.7
$\beta = 5$	29.603	31.757	**1.0**	1.1	0.2
$\beta = 20$	28.801	31.567	2.4	2.5	2.3
$\alpha = 0.02$	29.867	32.536	**1.4**	1.2	0.3
$\alpha = 0.067$	29.285	31.520	2.2	1.5	0.8
$\alpha = 0.1$	28.860	31.193	2.4	2.2	1.9
$\alpha = 0.2$	**27.611**	31.850	6.5	6.7	10.5
$\gamma = 0.15$	30.483	32.008	**1.3**	**1.0**	**0.1**
$\gamma = 0.35$	28.905	31.210	2.2	2.0	1.4
$\gamma = 0.45$	29.201	31.809	2.2	2.3	2.3
Reference	-	-	*44*	*2837242*	-



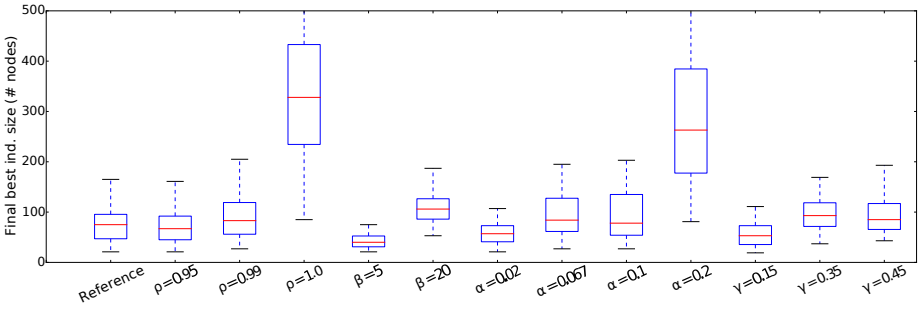
(a) Median best training RMSE vs. average mean solution size



(b) Median best testing RMSE vs. average mean solution size



(c) Average mean solution size vs. generation



(d) Best-of-run size boxplot

Figure 12: HARM-GP sensitivity analysis on Bioavailability. Results are for 30 independent runs. Each marker account for 10 000 evaluations.

B Detailed Results

This appendix presents detailed results for each of the twelve studied problems. The next section begins with a description of our general methodology, followed by an enumeration of per problem raw results.

B.1 General Methodology

For each problem, we proceed in the following three steps: 1) problem definition and dataset description (when applicable); 2) short survey of relevant literature; and 3) presentation of results in the form of tables and figures.

B.1.1 Result Tables

Most of the columns are simply the comparison metrics defined in Sec. 4.1. In these cases, we will refer to them using the notation previously defined.

The first column of each table is a fitness metric, measured on the *training* set ($\bar{f}_x^*$). The second column presents the same metric, but measured using an independent *testing* set ($\bar{g}_x^*$). For the problems where the training and testing sets are the same, namely Artificial Ant and Even Parity, these columns contain the fitness and the success rate, respectively. All fitness functions were designed to be *minimized*[6]. In the case of symbolic regression, the fitness is defined by Root Mean Squared Error (RMSE), while the classification error rate is used for classification problems. The presented fitnesses are for best-of-run individuals, as measured against the training set. The fitnesses for these individuals are also computed on the testing set in order to better evaluate their true performance. These columns present median results over all runs, except for Artificial Ant and Even Parity where an average is preferred over a median.

The third column is the size of the best-of-run individual ($\bar{s}_x^*$). Note that this is not necessarily the final mean size of the population, since the best-of-run can be a very large individual. This is intended as an indicator of the *solution complexity*. A tree of 800 nodes is far more difficult to study and analyze than a 20-node tree. From the perspective of a real-world application, this metric thus becomes very important. To avoid too large, hardly comparable numbers, we present relative results, the best result of each problem being used as a reference. This reference value is shown in the last row.

The fourth column contains the *accumulated size* ($\bar{c}_x$) over the whole evolution. The accumulated size is defined by the total number of nodes processed, thus effectively becoming a measurement of the total accumulated size since the beginning of the run. Because the evaluation step is the most computationally intensive part of a genetic programming evolution (at least on real-world problems), the number of nodes processed is a rough estimation of the computational

[6]Some bloat control methods explicitly require a *maximized* fitness. For these methods, a transformation was adequately applied to the fitness value before feeding it to the algorithm, but in order to keep results comparable, we present them using a minimized metric.

effort required to complete an evolution. The results are presented in a relative manner, as for the previous column.

The fifth and last column is a measurement of the population *bloat level* ($\bar{b}_x$) at the end of the run (see Sec. 4.3). It is also a minimized column (less is better).

Each method is run 100 times on each problem. Therefore, this table presents a condensed view of all runs. For all columns except the fitness-related ones, the data were averaged. For instance, the size column presents the average value of the best-of-run solution sizes. As for fitness, a median was preferred to avoid the outlier effect on the average value, which was particularly apparent for problems with RMSE metrics (since this metric is not upper-bounded). A mean was still used on artificial problems such as Even Parity, to be consistent with other results in the literature, and because they did not exhibit problematic behavior.

To assert the significance of results, we used a Wilcoxon signed-rank test, with a threshold at $p = 0.01$. This non-parametric test makes no assumptions about the underlying distribution of results. The best result of each column (e.g., the absolute minimum) is in bold. But when another method isn't deemed to be significantly different according to Wilcoxon signed-rank test, it is also in bold. All *non-bold* results are thus significantly worse than the best one.

The first table presents all of these results using the same evaluation budget for all methods (except for DynOpEq and FlatOpEq, see Sec. 4). However, as shown in the effort column, not all methods require the same computational effort. In order to be entirely fair over all methods, the second table presents the same metrics with the *equivalent computational effort* of the best method according to the test fitness. For instance, if the best method uses four times the effort of another, then this other method also obtains four times the number of runs (since, basically, it will require the same CPU time). Then, these runs are grouped by four, and, for each group, only the best run according to the training fitness is retained. This process produces a new set of 100 runs (or meta-runs), which are used to produce the second table.

B.1.2 Result Graphs

Different graphs present different views on performance, size, and bloat level. There are a total of eight graphs per problem, except for Artificial Ant and Even Parity that have only six, essentially because their training and testing sets are the same. These graphs are organized in two figures.

The first figure presents the following four graphs:

1. A plot of best fitness on the *training* set against mean solution size. As for the tables, a median is used to reduce the results of 100 runs into a single value, except for Artificial Ant and Even Parity where the average is used instead. As for the mean solution size, it is always averaged over the 100 runs. As this figure is generation based, different methods may obtain different curve lengths: we stopped the curve when less than 50%

of the runs remain active. Finally, the markers on the curves are evenly spaced according to the number of evaluations performed (each 10 000 evaluations), so they can be seen as a rough time line.

2. A boxplot of the best-of-run fitnesses, as measured on the training set for each run. In this figure, as in all boxplots, the ends of the whiskers represent the position of the lowest and highest samples within $1.5 \times \text{IQR}$, where IQR is the interquartile range. Outliers are not plotted.

3. A graph similar to the first, but where the best-of-run performance is measured on the testing set instead of the training set. This graph is not present for Artificial Ant and Even Parity, since the testing set is equal to the training set for these problems.

4. A boxplot of the best-of-run performances, as measured on the testing set for each run.

Note that performance metrics are always minimized. Also, the curves are often clipped because a full representation would stretch the graph too much, while a logarithmic scale on the size axis would render accurate comparisons difficult (this explains the discrepancies between the final value in the line plots and the value reported in tables and boxplots).

The second figure presents the following four graphs:

5. A plot of the mean individual size at the end of each generation. Notice the logarithmic scale on the size axis. As the number of generations is not the stopping criterion, different methods may generate lines of different length, depending on how they spent their evaluation budget.

6. A boxplot of best-of-run individual size.

7. A boxplot of the final population bloat level, according to the metric of Vanneschi *et al.* [70] (see Sec. 3.1).

8. A plot of the best training fitness against the accumulated size. This graph basically shows the evolution of fitness improvement over *time*, the accumulated size being roughly proportional to the elapsed time in most problems.

B.2 Artificial Ant

B.2.1 Problem Description and Parameters Used

This is the classical Artificial Ant problem, on the Santa Fe trail. This is a planning problem, where the programs control a simulated ant in order to harvest as much food as possible (with a maximum of 89 units of food). The primitive set includes an *if-food-ahead* clause, two serialization primitives (*prog2* and *prog3*), and three terminals, namely *move_forward*, *turn_left* and *turn_right*.

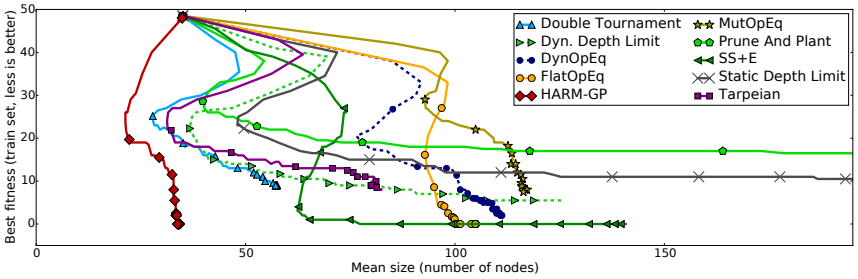
B.2.2 Previous Results and References

This problem was initially proposed by Koza in the early 1990s [29], and has been widely used in GP since then. In [73], it is part of the benchmark problems blacklist. However, no definitive argument is given about its improper use, and alternatives all involve real-time simulation (Physical TSP, Mario gameplay) or a complex simulated environment (TORCS). Considering that Artificial Ant is a problem of a distinct nature (i.e., robotic/agent planning and control) from all other problems presented in this study, we still present its results here.

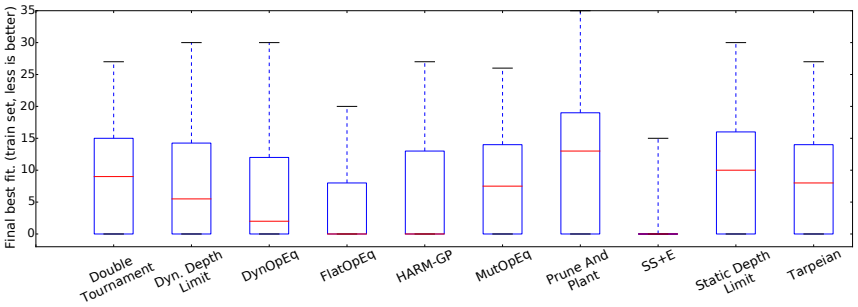
B.2.3 Results

Table 10: Detailed results of experiments conducted with the different bloat control methods for the Artificial Ant.

Method	Fitness ($\bar{g}^*_x$)	Success Rate	Size ($\bar{s}^*_x$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	7.020	43	1.4	1.6	0.7
Dyn. Depth Limit	7.550	43	2.3	2.6	1.8
DynOpEq	6.150	47	2.8	6.9	2.8
FlatOpEq	4.240	59	2.4	7.4	2.5
HARM-GP	5.710	53	**1.0**	**1.0**	**-0.3**
MutOpEq	7.530	41	3.1	3.7	3.5
Prune And Plant	9.530	35	4.6	9.4	26.4
SS+E	**1.190**	88	**1.4**	3.1	2.6
Static Depth Limit	8.750	41	3.1	5.6	7.7
Tarpeian	7.830	37	1.7	2.1	1.3
Reference	-	-	*36*	*4 858 363*	-

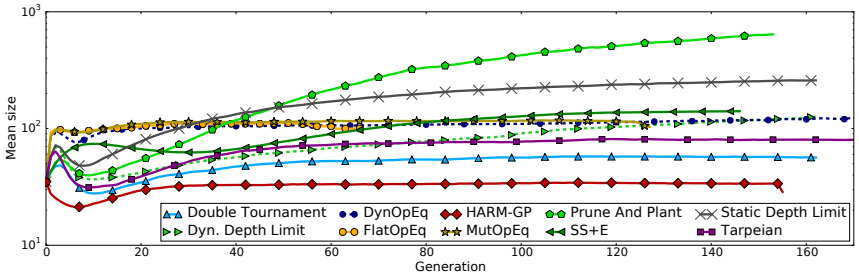


(a) Best fitness vs. mean size of the population

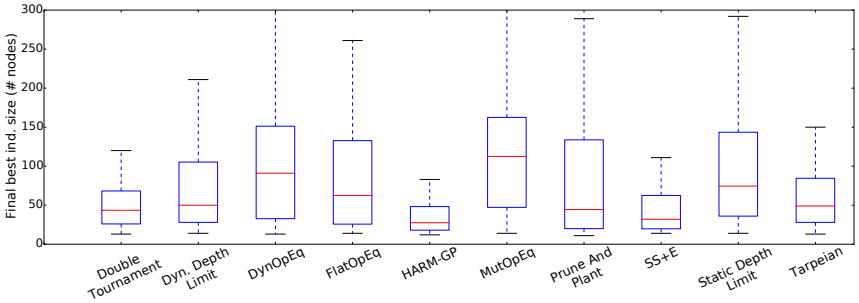


(b) Boxplot of the best-of-run fitness

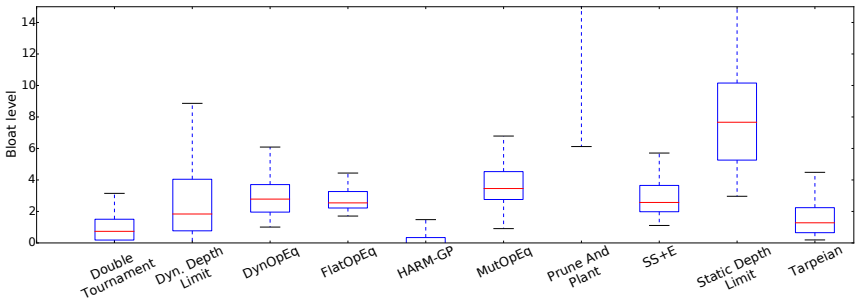
Figure 13: Fitness results for the Artificial Ant: (a) median over 100 runs of the best fitness achieved according to the mean size of the population, each marker accounting for 10 000 evaluations, some markers may overlap one over another or extend to the right of the graph; and (b) boxplot of the best-of-run individual fitness.



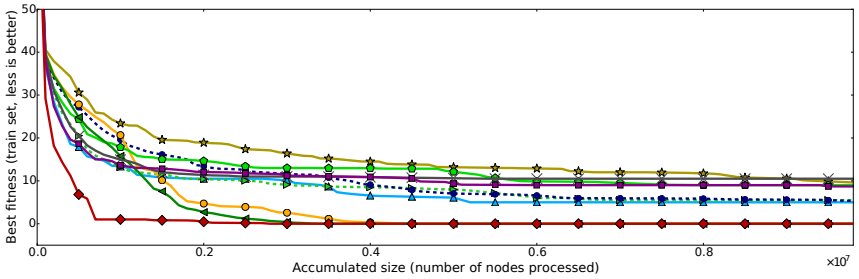
(a) Mean individual size vs. generation



(b) Boxplot of the best-of-run size



(c) Boxplot of the bloat level



(d) Best fitness vs. accumulated size

Figure 14: Size and bloat results for the Artificial Ant: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best fitness achieved according to the accumulated size.

Table 11: Detailed results of experiments conducted with the different bloat control methods for the Artificial Ant, with equivalent computational effort.

Method	Fitness ($\bar{g}^*_x$)	Success Rate	Size ($\bar{s}^*_x$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	3.290	62	1.9	1.8	0.4
Dyn. Depth Limit	6.420	51	2.9	3.0	1.7
DynOpEq	6.150	47	3.8	8.3	2.8
FlatOpEq	4.240	59	3.3	9.0	2.5
HARM-GP	**0.720**	89	**1.0**	**1.0**	**-0.5**
MutOpEq	7.530	41	4.2	4.5	3.5
Prune And Plant	9.530	35	6.3	11.3	26.4
SS+E	**1.190**	88	1.9	3.8	2.6
Static Depth Limit	8.750	41	4.2	6.7	7.7
Tarpeian	5.250	53	2.2	2.4	0.9
Reference	-	-	*26*	*4 020 047*	-

B.3 Even Parity 8

B.3.1 Problem Description and Parameters Used

This is the classic Even Parity 8 problem, where the program has to find the parity value of a given number of bits. In our case, we use an 8 bits parity, which translates into a total number of samples of 256. The primitive set was composed of $\{\text{AND}, \text{OR}, \text{XOR}, \text{NOT}\}$ and the terminals 0 and 1. The fitness is defined by the number of wrong parity bits over the whole training set.

B.3.2 Previous Results and References

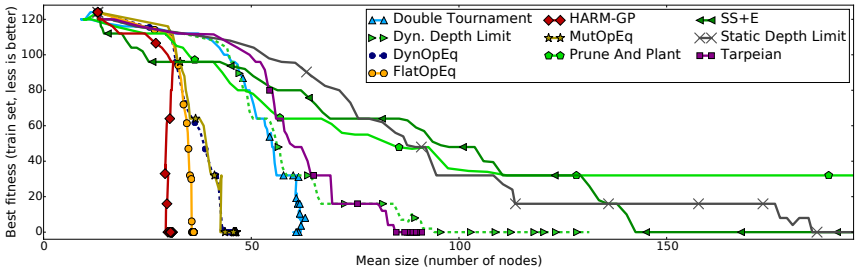
This problem has been proposed by Koza in the early 1990s [29]. While it does not offer a serious challenge in terms of performance, it remains an interesting comparison problem considering the lack of good and straightforward Boolean benchmarks in [73][7], especially considering that we are not only interested by the final fitness value, but also by the solution size and the computational effort required to achieve it.

B.3.3 Results

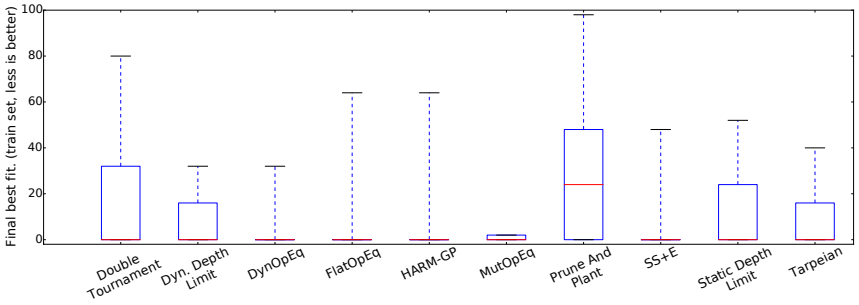
Table 12: Detailed results of experiments conducted with the different bloat control methods for Even Parity 8.

Method	Fitness ($\bar{g}^*_x$)	Success Rate	Size ($\bar{s}^*_x$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	21.610	55	2.2	1.9	6.4
Dyn. Depth Limit	10.220	63	2.8	3.3	10.6
DynOpEq	**2.640**	90	1.5	3.2	**4.0**
FlatOpEq	**3.520**	90	1.5	3.3	5.1
HARM-GP	**6.400**	78	**1.0**	**1.0**	**3.7**
MutOpEq	7.840	77	1.5	1.6	4.9
Prune And Plant	25.960	40	5.8	11.9	94.4
SS+E	**3.600**	85	3.7	4.4	14.9
Static Depth Limit	13.680	55	3.7	5.9	21.9
Tarpeian	9.530	66	2.6	2.6	5.2
Reference	-	-	*30*	*4 454 749*	-

[7] For instance, this paper proposed a multiple output parallel multiplier. While interesting, this problem was created for Cartesian GP, and leads to a series of design decision which can have a significant affect on the final performance, the most important one being how to create a tree with multiple outputs using standard GP.

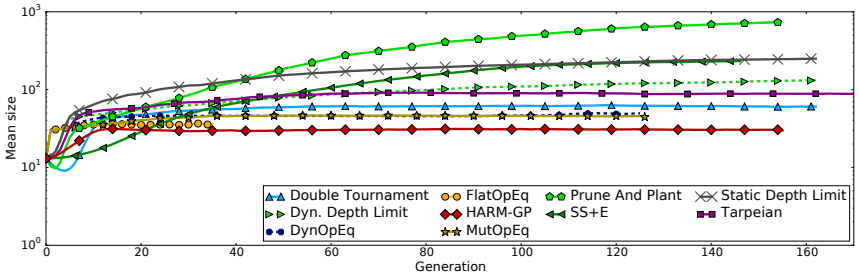


(a) Best fitness vs. number of fitness evaluations

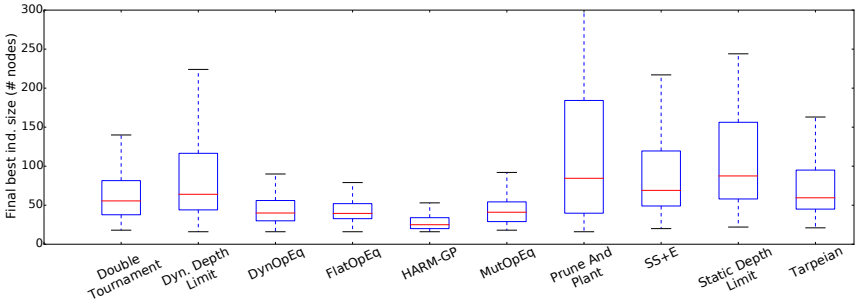


(b) Best-of-run fitness boxplot

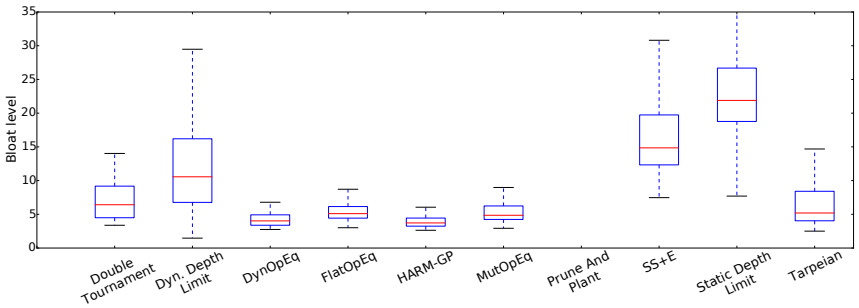
Figure 15: Fitness results for the Even Parity 8: (a) median over 100 runs of the best fitness achieved according to the mean size of the population, each marker accounting for 10 000 evaluations, some markers may overlap one over another or extend to the right of the graph; and (b) boxplot of the best-of-run individual fitness.



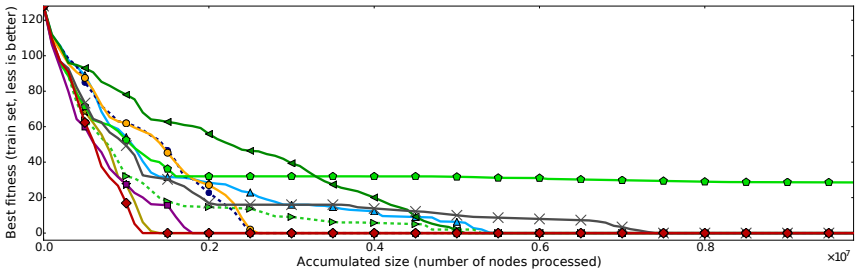
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best fitness vs. accumulated size

Figure 16: Size and bloat results for Even Parity 8: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best fitness achieved according to the accumulated size.

Table 13: Detailed results of experiments conducted with the different bloat control methods for Even Parity 8, with equivalent computational effort.

Method	Fitness ($\bar{g}_x^*$)	Success Rate	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	11.670	68	2.3	2.0	6.2
Dyn. Depth Limit	10.220	63	3.1	3.3	10.6
DynOpEq	2.640	90	1.7	3.2	4.0
FlatOpEq	3.520	90	1.6	3.3	5.1
HARM-GP	**0.160**	99	**1.0**	**1.0**	**3.5**
MutOpEq	7.840	77	1.6	1.6	4.9
Prune And Plant	25.960	40	6.3	11.9	94.4
SS+E	3.600	85	4.1	4.4	14.9
Static Depth Limit	13.680	55	4.1	5.9	21.9
Tarpeian	9.340	69	2.6	2.5	5.1
Reference	-	-	*27*	*4 457 008*	-

B.4 Symbolic Regression: Keijzer-6

B.4.1 Problem Description and Parameters Used

This problem consists in a one variable symbolic regression over the following equation (which is actually the harmonic series):

$$\mathrm{f}(x) = \sum_{i=1}^{x} \frac{1}{i}.$$

It uses the first 50 natural numbers (except 0) as a training set, and $\{1, 2, ...120\}$ as a test set in order to emphasize extrapolation. The primitive set used is a standard symbolic regression set: $\{+, -, *, /, \mathrm{neg}, \cos, \sin, \ln\}$, with the division and the natural logarithmic operator protected. The initialization is ramped half-and-half in the $[2 - 5]$ range for maximum depth. Reported errors are RMSE.

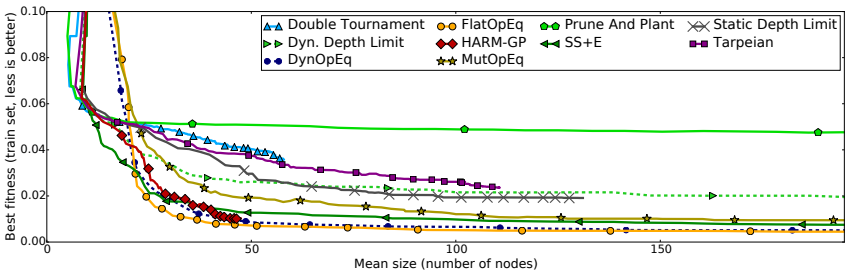
B.4.2 Previous Results and References

The problem has originally been proposed by Keijzer in [25] to show the potential of linear scaling in GP. It is part of the recommended GP benchmarks in [73] and was also used as a benchmark in [63], with an error on the training set of about 5×10^{-4} at best (no results were given on a test set).

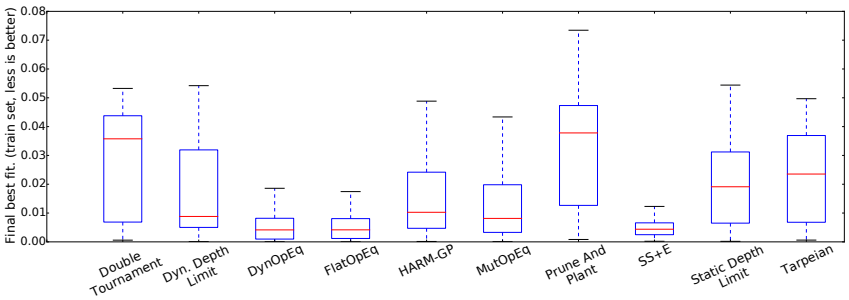
B.4.3 Results

Table 14: Detailed results of experiments conducted with the different bloat control methods for Keijzer-6.

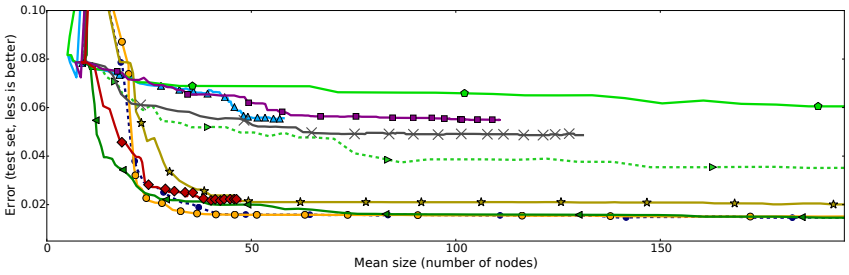
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.032	0.056	**1.0**	**1.1**	3.9
Dyn. Depth Limit	0.009	0.035	24.1	18.6	124.1
DynOpEq	**0.004**	0.016	20.5	29.6	109.3
FlatOpEq	**0.004**	0.015	9.2	11.5	38.3
HARM-GP	0.008	0.021	**1.1**	**1.0**	**3.2**
MutOpEq	0.008	0.020	4.2	3.5	16.7
Prune And Plant	0.035	0.060	12.9	10.9	59.8
SS+E	**0.004**	**0.011**	5.2	5.4	45.9
Static Depth Limit	0.019	0.049	2.0	2.5	9.3
Tarpeian	0.024	0.055	1.8	2.0	8.2
Reference	-	-	*69*	*5 439 003*	-



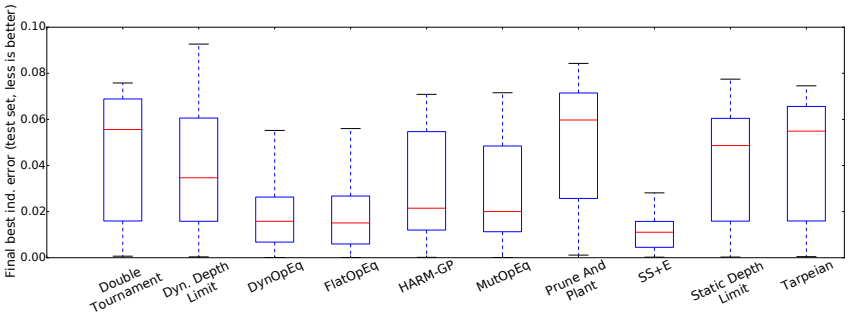
(a) Best RMSE on the training set vs. mean size of the population



(b) Boxplot of the best-of-run RMSE on the training set

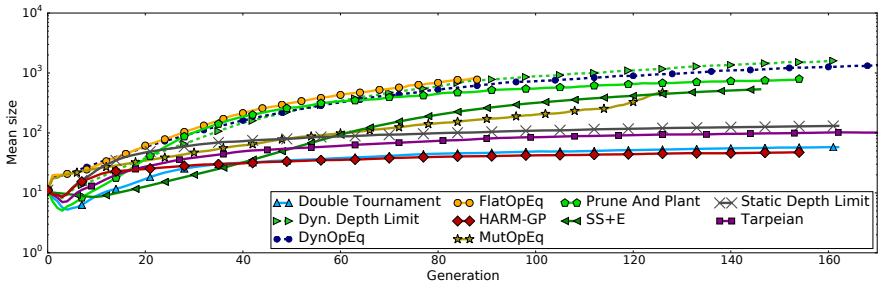


(c) Best RMSE on the testing set vs. mean size of the population

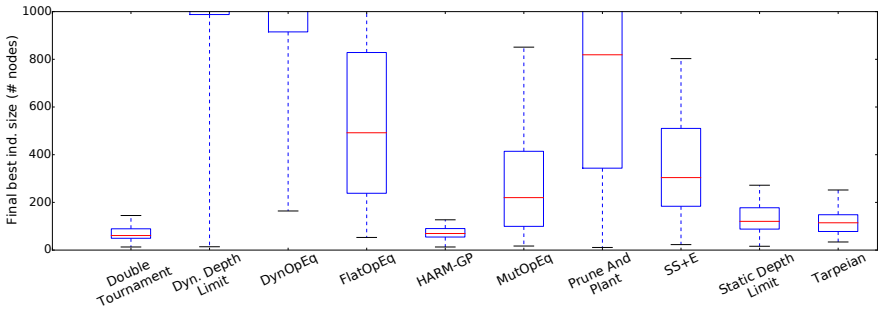


(d) Boxplot of the best-of-run RMSE on the testing set

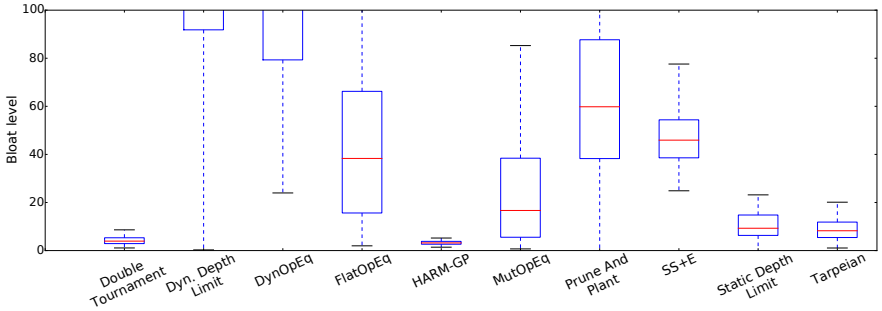
Figure 17: Performance and size results for Keijzer-6, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



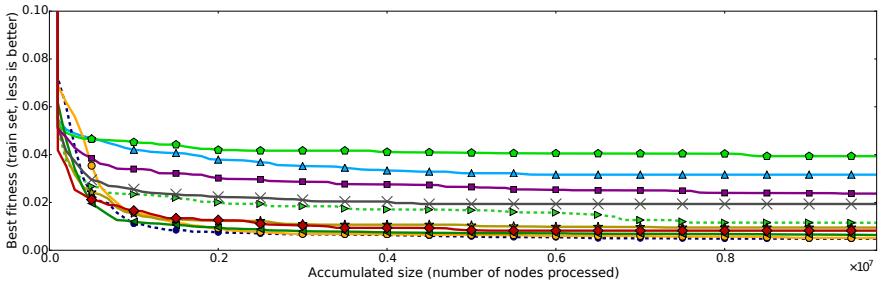
(a) Mean individual size vs. generation



(b) Boxplot of the best-of-run size



(c) Boxplot of the bloat level



(d) Best RMSE on the training set vs. accumulated size

Figure 18: Size and bloat results for Keijzer-6: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 15: Detailed results of experiments conducted with the different bloat control methods for Keijzer-6, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.007	0.016	**1.0**	**1.0**	**2.7**
Dyn. Depth Limit	0.009	0.035	33.1	20.7	124.1
DynOpEq	0.004	0.016	28.1	33.0	109.3
FlatOpEq	0.004	0.015	12.7	12.8	38.3
HARM-GP	**0.002**	**0.007**	1.6	1.2	3.5
MutOpEq	0.008	0.020	5.7	3.9	16.7
Prune And Plant	0.035	0.060	17.7	12.1	59.8
SS+E	0.004	0.011	7.1	6.0	45.9
Static Depth Limit	0.007	0.016	2.7	2.7	9.0
Tarpeian	0.007	0.016	2.0	1.9	6.7
Reference	-	-	*50*	*4 887 484*	-

B.5 Symbolic Regression: Nguyen-7

B.5.1 Problem Description and Parameters Used

This problem involves a one variable symbolic regression over the equation:

$$f(x) = \ln(x+1) + \ln(x^2+1) \tag{9}$$

It uses 20 training points from a uniform distribution over the range $[0, 2]$. The test set was not mentioned in previous work, but in this paper we use 1000 points uniformly distributed in the same range. The primitive set used is a standard symbolic regression set: $\{+, -, *, /, \text{neg}, \cos, \sin, \ln\}$, with the division and the natural logarithmic operator protected. The initialization is conducted with a ramped half-and-half method, with a $[2-5]$ range for the maximum initial depth. The reported errors are the computed RMSE.

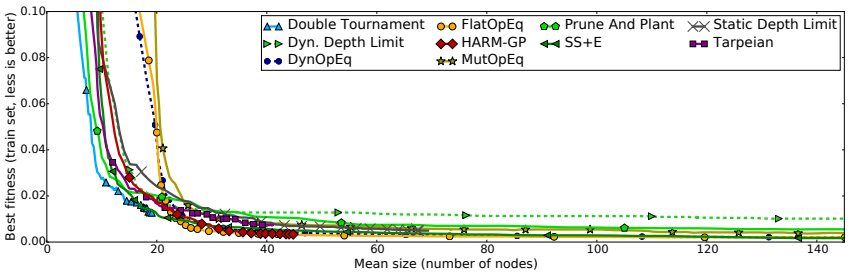
B.5.2 Previous Results and References

This problem was proposed and used by Nguyen *et al.* in [69], where they report a final sum of errors of about 0.13, which translates to an average error of 6.5×10^{-3} on the training set. While the aim of [69] was to show the benefits of a semantically aware crossover, the problem can still be of some use in other areas of GP, and is among the GP benchmarks recommended in [73].

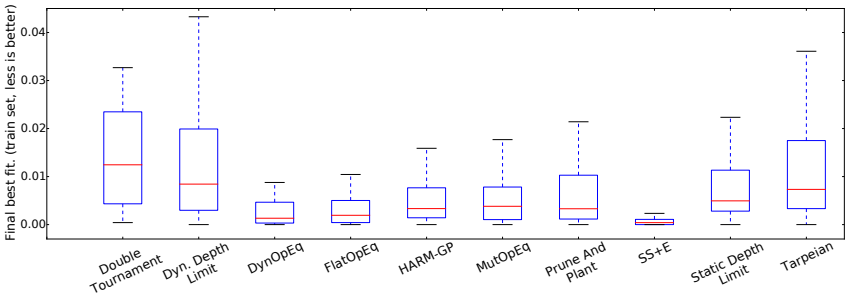
B.5.3 Results

Table 16: Detailed results of experiments conducted with the different bloat control methods for Nguyen-7.

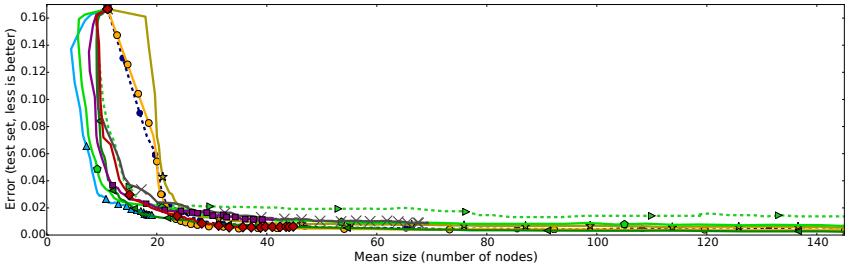
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.010	0.014	**1.0**	**1.0**	**0.7**
Dyn. Depth Limit	0.008	0.013	23.3	12.7	2.9
DynOpEq	0.001	0.005	30.1	32.9	51.7
FlatOpEq	0.002	0.003	13.2	13.0	4.9
HARM-GP	0.003	0.006	2.8	2.2	3.2
MutOpEq	0.003	0.005	7.8	5.6	8.2
Prune And Plant	0.003	0.010	25.8	15.8	32.2
SS+E	**0.000**	**0.001**	10.2	8.4	27.9
Static Depth Limit	0.005	0.009	2.7	3.2	3.8
Tarpeian	0.007	0.011	1.9	1.7	1.7
Reference	-	-	*24*	*2 342 961*	-



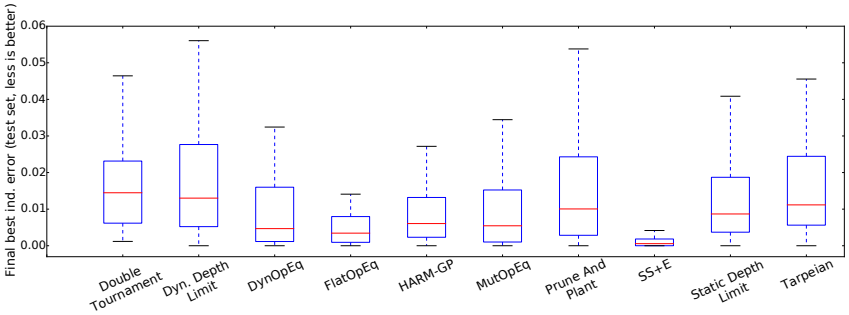
(a) Best RMSE on the training set vs. mean size of the population



(b) Best-of-run training RMSE boxplot

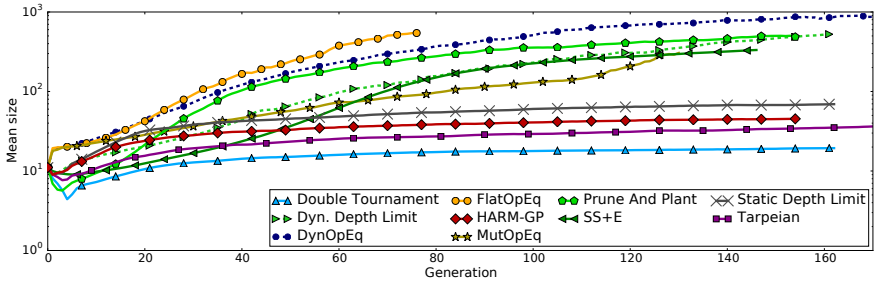


(c) Best RMSE on the test set vs. mean size of the population

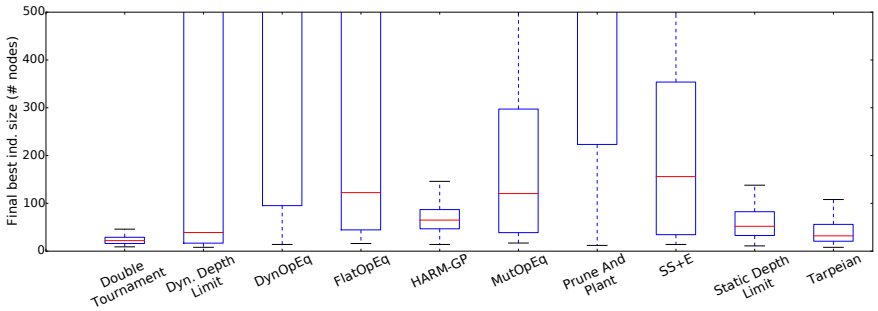


(d) Best-of-run test RMSE boxplot

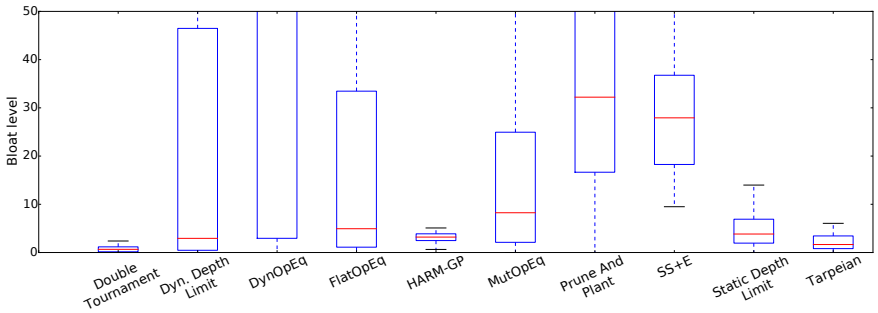
Figure 19: Performance and size results for Nguyen-7, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



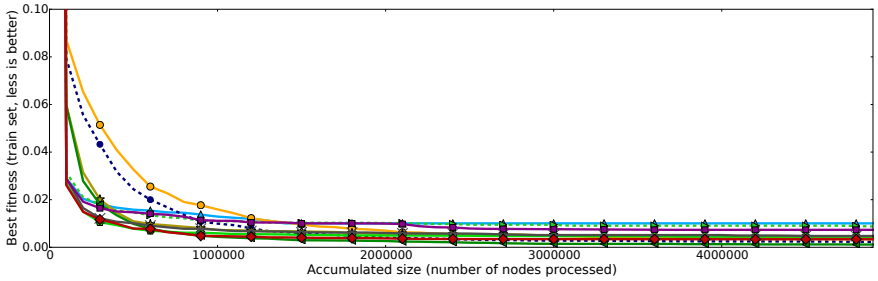
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best RMSE on the training set vs. accumulated size

Figure 20: Size and bloat results for Nguyen-7: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 17: Detailed results of experiments conducted with the different bloat control methods for Nguyen-7, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.002	0.003	**1.0**	**1.0**	**1.3**
Dyn. Depth Limit	0.008	0.013	17.2	8.9	2.9
DynOpEq	0.001	0.005	22.3	23.2	51.7
FlatOpEq	0.002	0.003	9.8	9.1	4.9
HARM-GP	0.001	0.002	2.0	1.6	3.4
MutOpEq	0.003	0.005	5.8	3.9	8.2
Prune And Plant	0.003	0.010	19.1	11.1	32.2
SS+E	**0.000**	**0.001**	7.5	5.9	27.9
Static Depth Limit	0.002	0.003	2.0	2.4	4.3
Tarpeian	0.003	0.004	1.7	1.5	3.0
Reference	-	-	*33*	*3 331 869*	-

B.6 Symbolic Regression: Pagie-1

B.6.1 Problem Description and Parameters Used

This problem consists in a two-variable symbolic regression over the equation:

$$\mathrm{f}(x, y) = \frac{1}{1 + x^{-4}} + \frac{1}{1 + y^{-4}} \tag{10}$$

It uses 25 training points, evenly distributed over the range $[-5, 5]$ on each dimension. The test set was not mentioned in previous work. We used 50 points uniformly distributed in the same range. The primitive set used is a standard symbolic regression set: $\{+, -, *, /, \mathrm{neg}, \cos, \sin,$
$\ln\}$, with the division and the natural logarithmic operator protected, plus an ephemeral constant in the range $[-1, 1]$. The initialization is carried out with a ramped half-and-half method, with a $[2 - 5]$ range for the maximum initial depth. The reported errors are the RMSE.

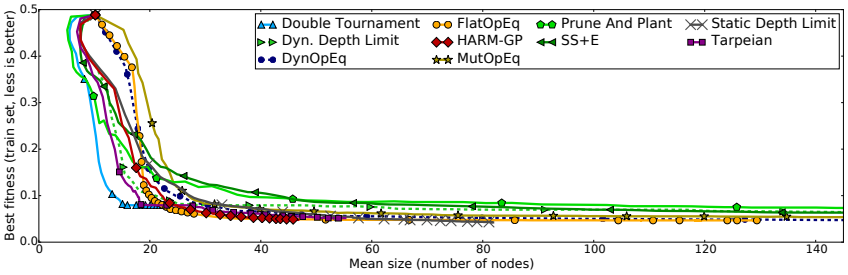
B.6.2 Previous Results and References

Introduced by Pagie and Hogeweg in [43], this problem has been used in the literature on different aspects of GP. In [62], Spector and Helmuth reported a mean error of 0.32 on the test set for standard GP, over 100 runs. This problem is also part of the recommended GP benchmarks listed in [73].

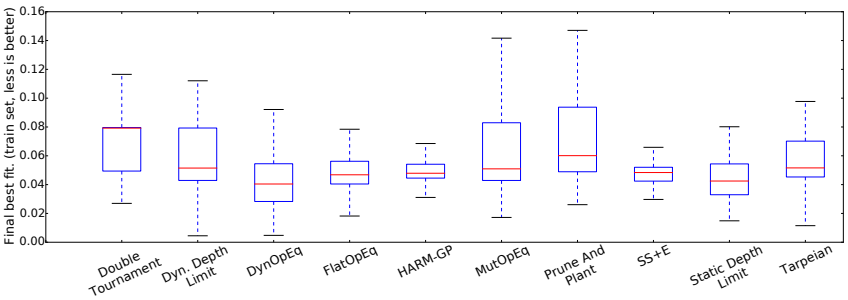
B.6.3 Results

Table 18: Detailed results of experiments conducted with the different bloat control methods for Pagie-1.

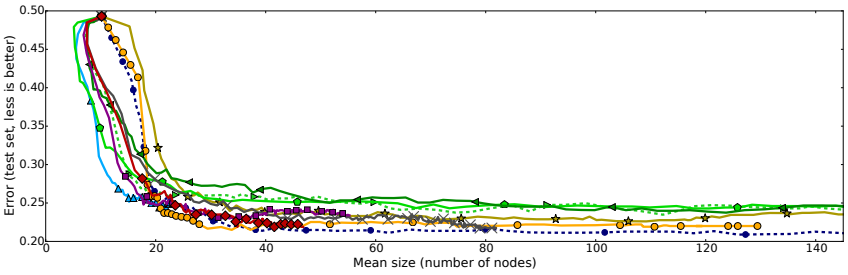
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.064	0.252	**1.0**	**1.0**	**0.2**
Dyn. Depth Limit	0.051	0.236	23.6	14.8	59.1
DynOpEq	**0.040**	**0.207**	24.6	28.7	68.4
FlatOpEq	**0.047**	**0.220**	5.0	6.6	3.1
HARM-GP	0.047	**0.219**	2.2	1.9	3.6
MutOpEq	0.050	0.235	7.8	5.5	11.4
Prune And Plant	0.060	0.239	16.4	10.6	35.5
SS+E	**0.048**	**0.205**	7.7	6.5	29.8
Static Depth Limit	**0.042**	**0.217**	2.9	3.3	6.4
Tarpeian	0.051	0.234	2.0	1.9	4.2
Reference	-	-	*30*	*2 741 243*	-



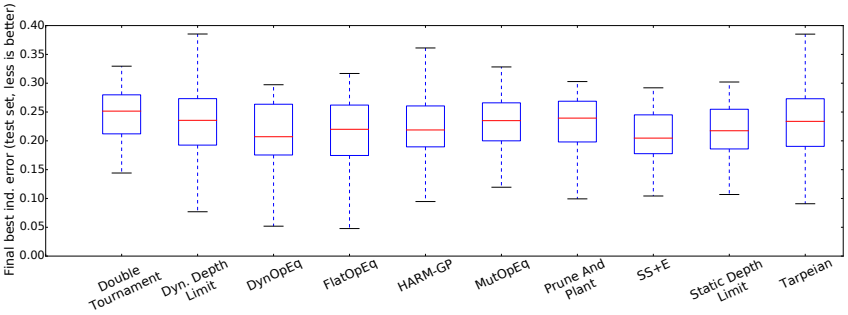
(a) Best RMSE on the training set vs. mean size of the population



(b) Best-of-run training RMSE boxplot

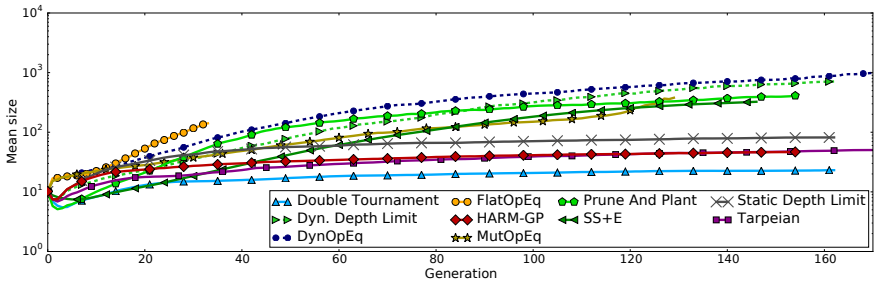


(c) Best RMSE on the test set vs. mean size of the population

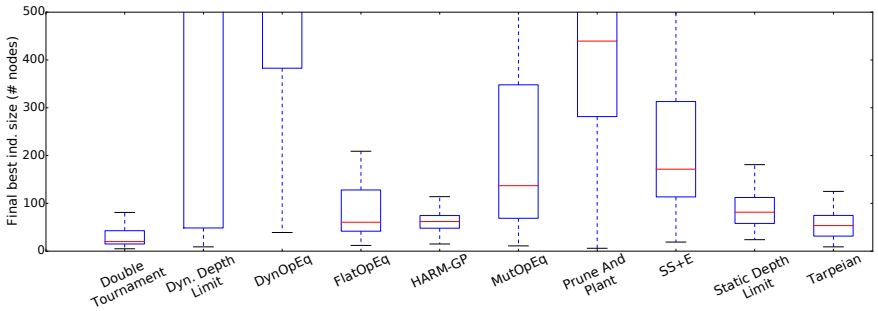


(d) Best-of-run test RMSE boxplot

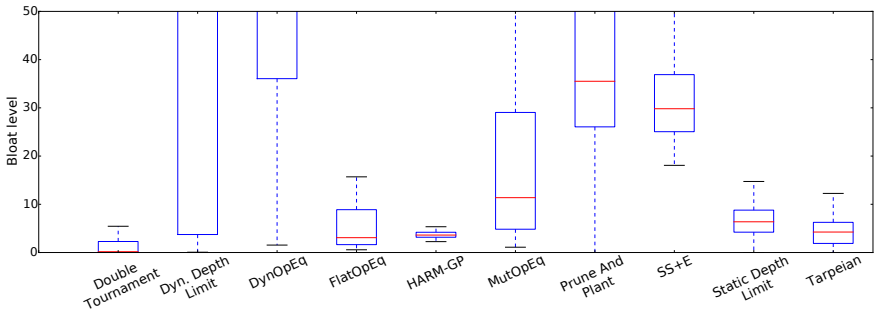
Figure 21: Performance and size results for Pagie-1, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



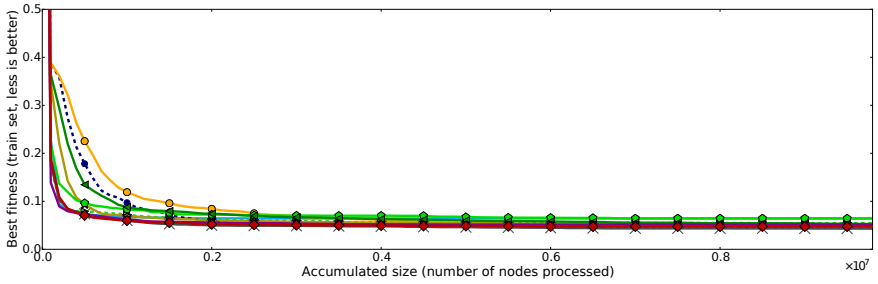
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best RMSE on the training set vs. accumulated size

Figure 22: Size and bloat results for Pagie-1: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 19: Detailed results of experiments conducted with the different bloat control methods for Pagie-1, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.045	**0.217**	**1.0**	**1.0**	**2.5**
Dyn. Depth Limit	0.051	0.236	15.5	9.7	59.1
DynOpEq	0.040	**0.207**	16.1	18.8	68.4
FlatOpEq	0.047	**0.220**	3.3	4.3	3.1
HARM-GP	0.041	**0.216**	1.5	1.3	3.7
MutOpEq	0.050	0.235	5.1	3.6	11.4
Prune And Plant	0.060	0.239	10.8	7.0	35.5
SS+E	0.048	**0.205**	5.0	4.3	29.8
Static Depth Limit	**0.033**	**0.209**	2.2	2.4	7.1
Tarpeian	0.042	**0.211**	1.5	1.5	4.4
Reference	-	-	*46*	*4 182 382*	-

B.7 Symbolic Regression: Vladislavleva-4

B.7.1 Problem Description and Parameters Used

This problem involves a five variables symbolic regression over the equation:

$$\mathrm{f}(x_1, x_2, \ldots, x_5)\frac{10}{5 + \sum_{i=1}^{5}(x_i - 3)^2} \tag{11}$$

It uses 1024 training points from a uniform distribution over the range $[0.05, 6.05]$, and 500 test points, over the range $[-0.25, 6.35]$. The primitive set used is a standard symbolic regression set: $\{+, -, *, /, \mathrm{neg}, \cos, \sin, \ln\}$, with the division and the natural logarithmic operator protected, plus an ephemeral constant in the range $[-1, 1]$. The initialization is performed with a ramped half-and-half method, with a $[2 - 5]$ range for the maximum initial depth. The reported errors are the computed RMSE.

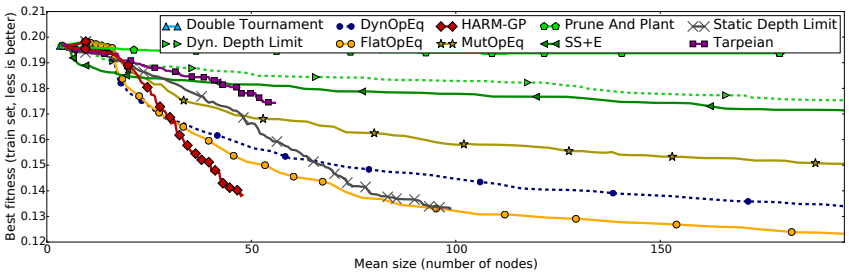
B.7.2 Previous Results and References

This problem was introduced by Vladislavleva *et al.* in [71] as a good problem to tackle generalization capabilities of GP. They report a training RMSE of 0.173 and a test RMSE of about 0.277. This problem is part of the recommended GP benchmarks in [73].

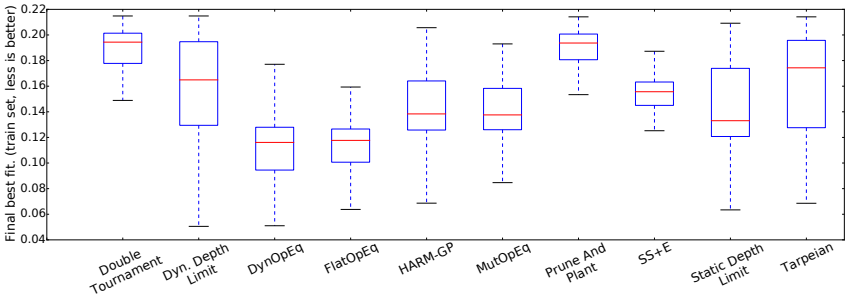
B.7.3 Results

Table 20: Detailed results of experiments conducted with the different bloat control methods for Vladislavleva-4.

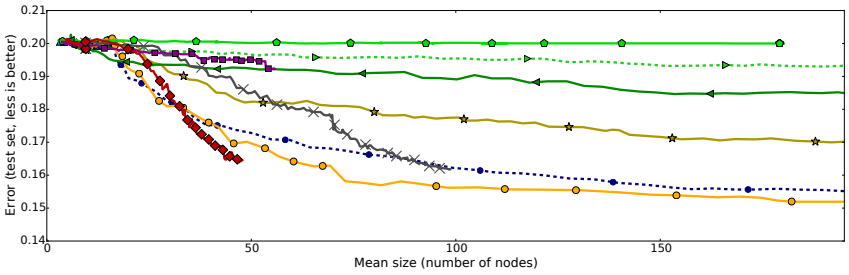
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.194	0.199	**1.0**	**1.0**	**-0.7**
Dyn. Depth Limit	0.165	0.180	39.2	30.0	58.2
DynOpEq	**0.116**	**0.149**	56.7	87.0	113.3
FlatOpEq	**0.118**	**0.150**	23.1	34.1	40.8
HARM-GP	0.137	0.165	3.3	3.2	**4.1**
MutOpEq	0.138	0.163	23.0	19.3	42.2
Prune And Plant	0.194	0.200	**10.8**	**7.8**	**-0.8**
SS+E	0.156	0.171	12.8	14.0	35.4
Static Depth Limit	0.133	0.162	4.7	6.2	**9.6**
Tarpeian	0.174	0.192	2.7	3.4	**2.9**
Reference	-	-	*22*	*1 565 337*	-



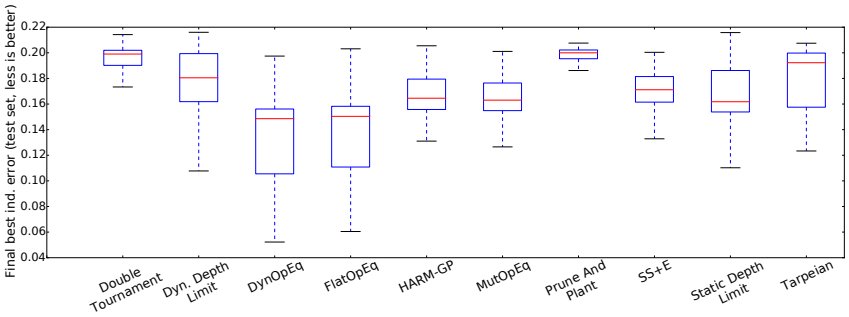
(a) Best RMSE on the training set vs. mean size of the population



(b) Best-of-run training RMSE boxplot

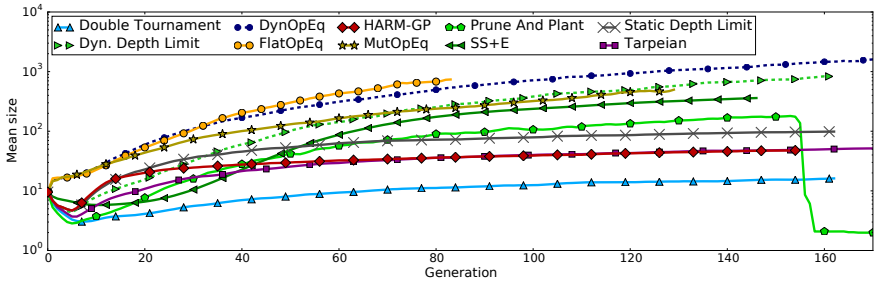


(c) Best RMSE on the test set vs. mean size of the population

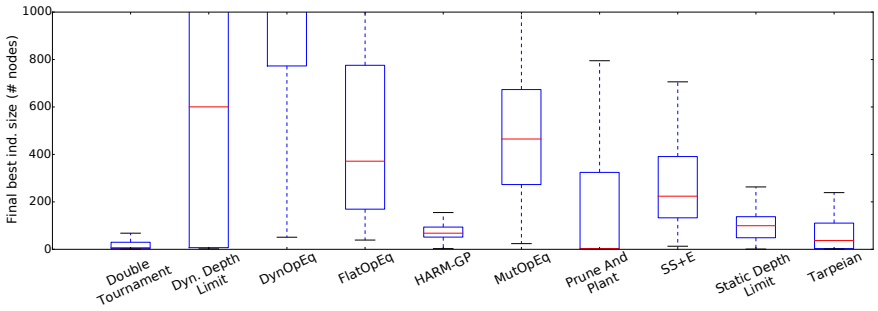


(d) Best-of-run test RMSE boxplot

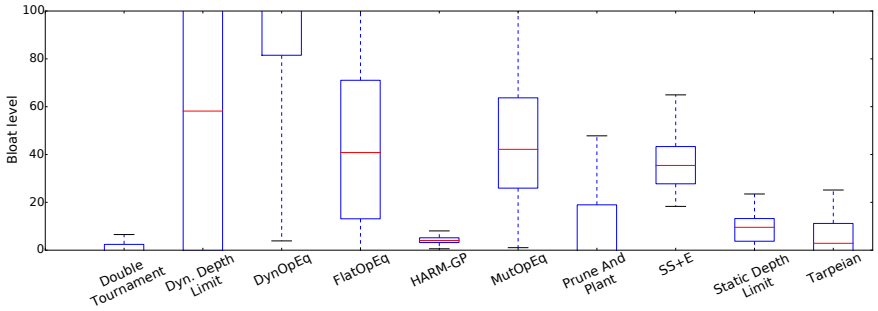
Figure 23: Performance and size results for Vladislavleva-4, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



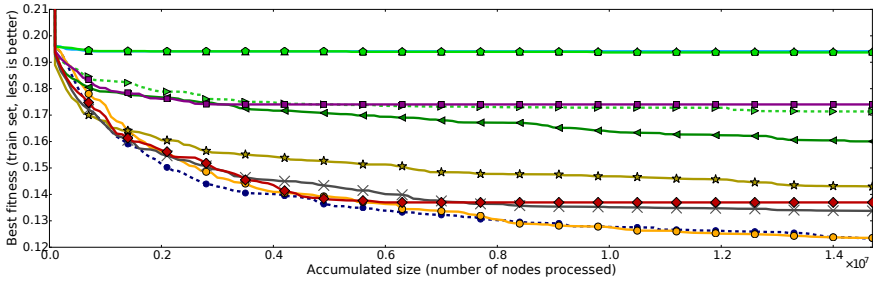
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best RMSE on the training set vs. accumulated size

Figure 24: Size and bloat results for Vladislavleva-4: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 21: Detailed results of experiments conducted with the different bloat control methods for Vladislavleva-4, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.119	0.154	**1.0**	**1.0**	**5.1**
Dyn. Depth Limit	0.159	0.178	12.2	8.3	61.7
DynOpEq	**0.116**	**0.149**	16.8	22.8	113.3
FlatOpEq	**0.118**	**0.150**	6.8	9.0	40.8
HARM-GP	**0.115**	**0.150**	1.3	**1.0**	**4.8**
MutOpEq	0.129	0.155	6.1	5.1	40.7
Prune And Plant	0.148	0.170	7.5	4.6	34.5
SS+E	0.147	0.165	3.6	3.6	34.3
Static Depth Limit	**0.118**	**0.152**	2.0	2.2	12.7
Tarpeian	0.117	**0.152**	1.7	1.8	12.1
Reference	-	-	*75*	*5 963 511*	-

B.8 Symbolic Regression: Bioavailability

B.8.1 Problem Description and Parameters Used

This problem is a real-world symbolic regression, with the aim of estimating the *bioavailability* of chemicals based on some chemical properties. The dataset itself contains 718 samples, each with 241 properties, making it a high-dimension problem with a relatively small training set. The reported errors are the computed RMSE. For each run, 30% of the data is kept as a test set, while the remaining 70% is used to train the individuals. We took care for the data separation to be the same across all methods (e.g., the run # 46 of all methods uses the same training set). The primitive set used is a reduced symbolic regression set: $\{+, -, *, /\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. The initialization is conducted with a ramped half-and-half method, with a $[2 - 5]$ range for the maximum initial depth. The dataset is publicly available at `http://personal.disco.unimib.it/Vanneschi/bioavailability.txt`.

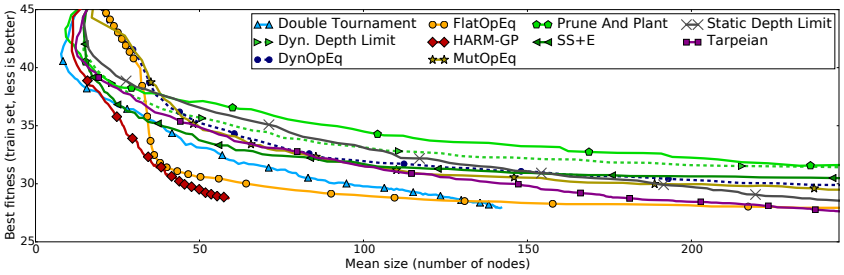
B.8.2 Previous Results and References

Initially proposed by Silva and Vanneschi in [58], this problem has been used in the analysis of some bloat control methods. The author report a RMSE of approximately 29 and 34, respectively on the training and the test set. In [62], median errors of about 38 (training) and 42 (test) are reported over 200 runs. This problem is proposed as a GP benchmark in [39].

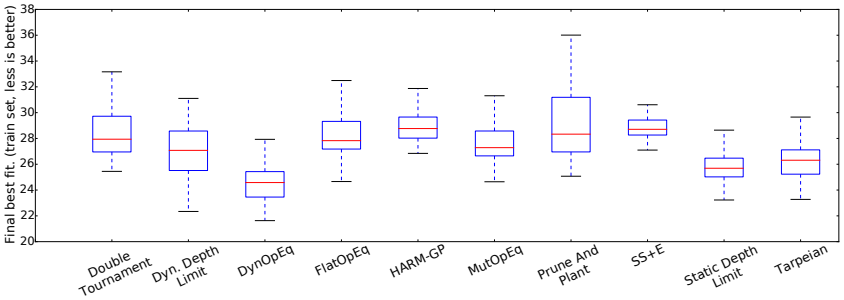
B.8.3 Results

Table 22: Detailed results of experiments conducted with the different bloat control methods for Bioavailability.

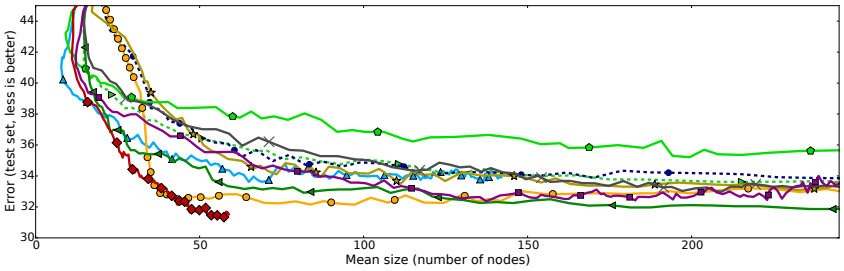
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	27.874	33.903	1.6	2.0	6.0
Dyn. Depth Limit	27.076	36.457	31.8	27.7	143.5
DynOpEq	**24.581**	46.796	34.9	58.9	148.9
FlatOpEq	27.829	**33.019**	3.7	4.7	5.0
HARM-GP	28.691	**31.511**	**1.0**	**1.0**	**1.8**
MutOpEq	27.273	35.539	7.6	6.6	23.4
Prune And Plant	28.333	36.449	14.9	10.1	41.7
SS+E	28.707	**31.332**	7.8	6.1	44.7
Static Depth Limit	25.690	33.982	4.3	5.8	20.8
Tarpeian	26.305	34.847	3.3	4.2	26.8
Reference	-	-	*100*	*6 280 874*	-



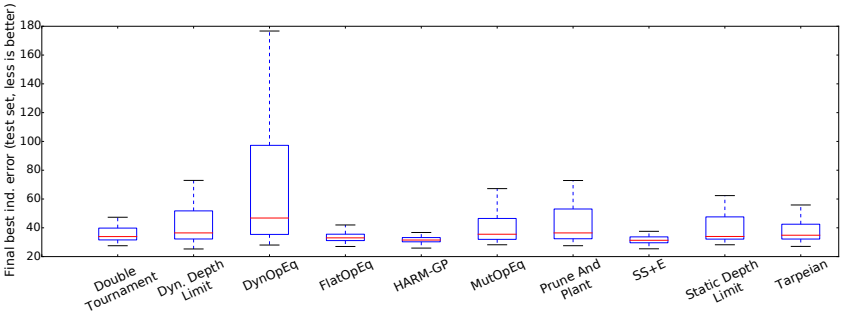
(a) Best RMSE on the training set vs. mean size of the population



(b) Best-of-run training RMSE boxplot

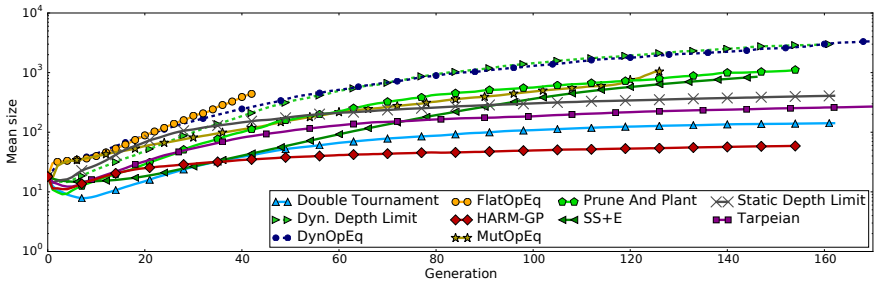


(c) Best RMSE on the test set vs. mean size of the population

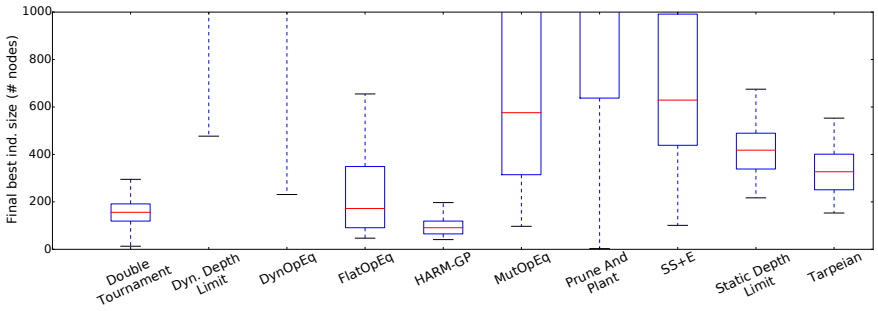


(d) Best-of-run test RMSE boxplot

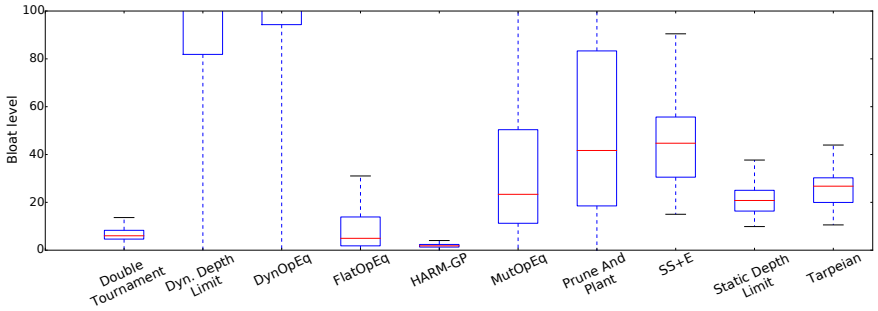
Figure 25: Performance and size results for Bioavailability, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



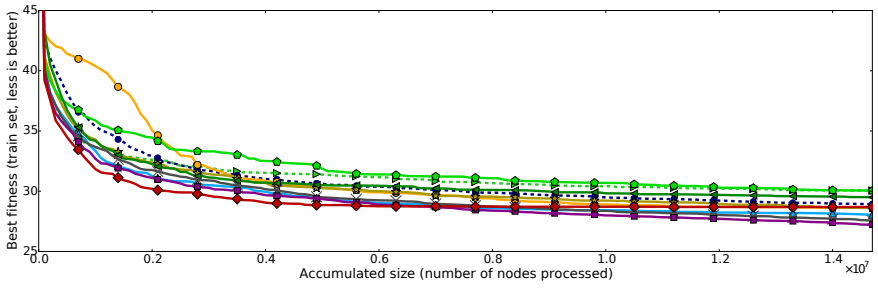
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best RMSE on the training set vs. accumulated size

Figure 26: Size and bloat results for Bioavailability: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 23: Detailed results of experiments conducted with the different bloat control methods for Bioavailability, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	26.887	32.911	1.5	1.9	6.7
Dyn. Depth Limit	27.076	36.457	29.6	25.7	143.5
DynOpEq	**24.581**	46.796	32.5	54.8	148.9
FlatOpEq	27.595	33.406	4.0	4.8	6.5
HARM-GP	27.354	**32.009**	**1.0**	**1.0**	**2.1**
MutOpEq	27.273	35.539	7.1	6.1	23.4
Prune And Plant	28.333	36.449	13.9	9.4	41.7
SS+E	28.707	**31.332**	7.3	5.7	44.7
Static Depth Limit	25.459	34.707	3.9	5.4	20.8
Tarpeian	25.828	33.881	3.0	3.9	25.4
Reference	-	-	*107*	*6 758 465*	-

B.9 Symbolic Regression: Dow Chemical

B.9.1 Problem Description and Parameters Used

Dow Chemical is a symbolic regression problem, with a dataset which is already divided into training/test set containing 747 training samples and 319 test samples. It has 57 features. We applied a whitening transform (i.e., transform data to follow a zero mean, zero covariance, and unitary variance multivariate normal distribution, $\mathcal{N}_D(0, \mathbf{I})$) over the dataset, computed on the training set. However, the RMSE values reported are on the original data. The primitive set used is a reduced symbolic regression set: $\{+, -, *, /\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. The initialization is carried out with a ramped half-and-half method, with a $[2 - 5]$ range for the maximum initial depth. The dataset is available online at `http://casnew.iti.upv.es/index.php/evocompetitions/105-symregcompetition`.

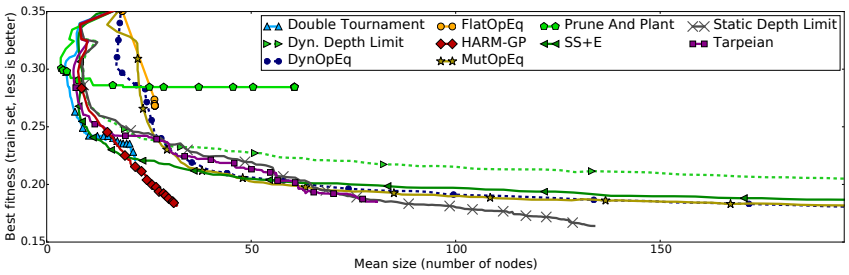
B.9.2 Previous Results and References

This problem was the subject of the symbolic regression competition of the EvoCompetitions event at the 2010 EvoStar conference. It has been used in several studies, such as [28], where the GP performance combined to some other heuristics was described as a "robust stacked empirical model". It is also one of the recommended benchmarks in [73].

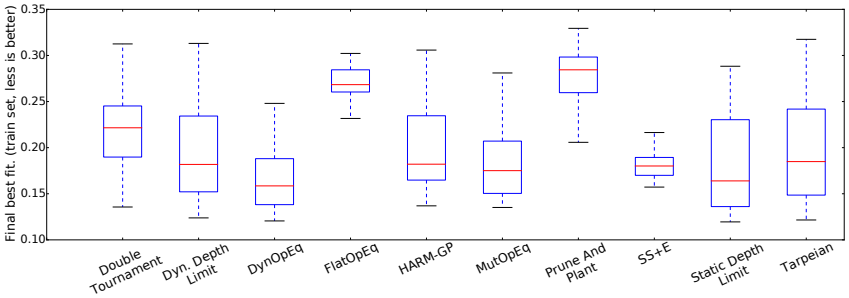
B.9.3 Results

Table 24: Detailed results of experiments conducted with the different bloat control methods for Dow Chemical.

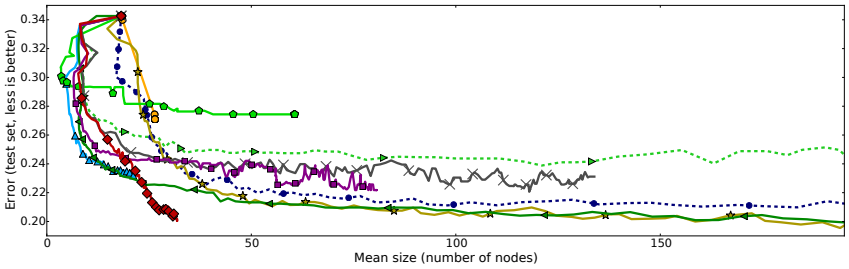
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.221	0.234	3.9	**1.0**	-0.6
Dyn. Depth Limit	0.182	0.257	185.4	27.7	63.4
DynOpEq	**0.159**	0.228	198.6	52.6	65.8
FlatOpEq	0.268	0.271	**1.0**	17.9	0.6
HARM-GP	0.181	**0.201**	7.9	1.7	1.0
MutOpEq	0.175	**0.195**	59.5	10.2	15.9
Prune And Plant	0.284	0.274	**11.5**	1.7	**-0.8**
SS+E	0.180	**0.193**	38.1	5.6	18.7
Static Depth Limit	**0.164**	0.231	19.7	5.3	5.6
Tarpeian	0.184	0.222	12.3	2.9	2.7
Reference	-	-	*7*	*2 199 072*	-



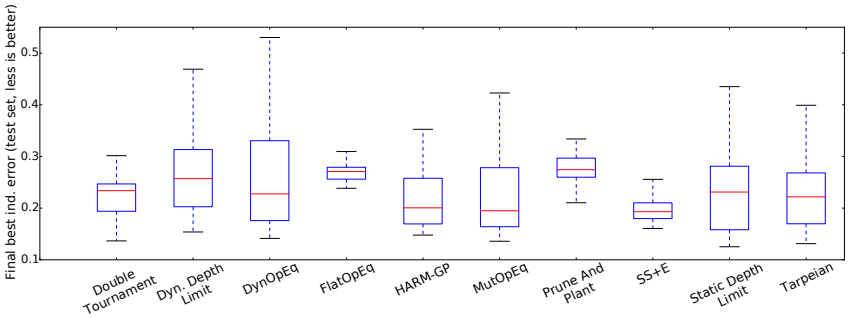
(a) Best RMSE on the training set vs. mean size of the population



(b) Best-of-run training RMSE boxplot

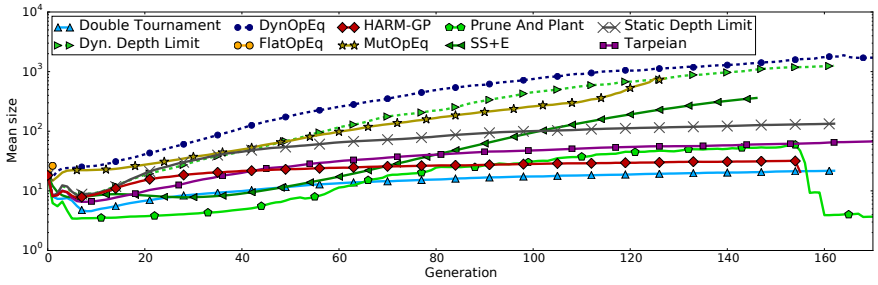


(c) Best RMSE on the test set vs. mean size of the population

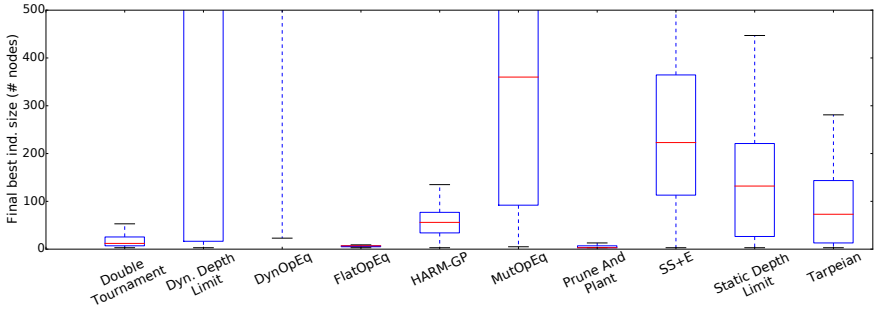


(d) Best-of-run test RMSE boxplot

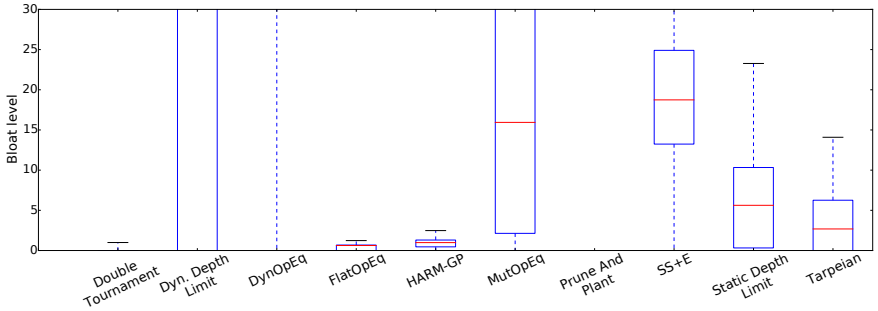
Figure 27: Performance and size results for DowChemical, over 100 runs: (a) and (c) plot the median of the RMSE of the best individual found so far (according to the training set) against the corresponding average mean solution size, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the RMSE of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



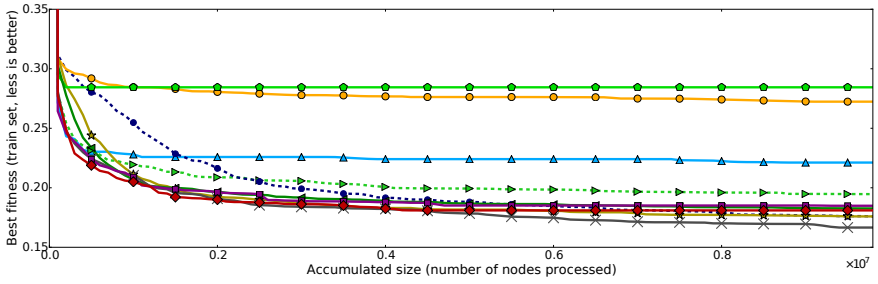
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best RMSE on the training set vs. accumulated size

Figure 28: Size and bloat results for DowChemical: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best RMSE on the training set achieved according to the accumulated size.

Table 25: Detailed results of experiments conducted with the different bloat control methods for Dow Chemical, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	**0.154**	**0.166**	10.6	**1.2**	2.2
Dyn. Depth Limit	0.182	0.257	185.4	13.4	63.4
DynOpEq	**0.159**	0.228	198.6	25.5	65.8
FlatOpEq	0.268	0.271	**1.0**	8.7	**0.6**
HARM-GP	**0.160**	**0.166**	10.2	**1.0**	1.2
MutOpEq	0.175	0.195	59.5	4.9	15.9
Prune And Plant	0.244	0.247	29.3	1.7	**-0.7**
SS+E	0.180	0.193	38.1	2.7	18.7
Static Depth Limit	**0.158**	0.227	20.7	2.7	6.4
Tarpeian	**0.155**	0.184	14.5	1.7	4.1
Reference	-	-	*7*	*4 535 017*	-

B.10 Classification: Adult Dataset

B.10.1 Problem Description and Parameters Used

This is a classification problems based on a U.S. census. The aim of the problem is to determine if the average income of a person is greater or less than 50 k$, therefore making it a 2-class problem. The dataset is available on the UCI database, but we used a slightly modified version, with the continuous features discretized, from the libsvm repository (namely the a9a dataset from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`). In this form, the dataset has 123 features. We used the following set of primitives: $\{+, -, *, /, \text{iflessthan}\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. The fitness corresponds to the classification error rate on the training set.

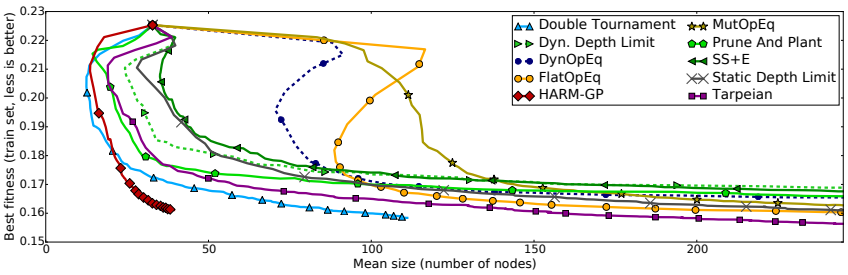
B.10.2 Previous Results and References

This problem has often been used in machine learning papers, and is part of the so-called *CHIRP suite*, which is one of the 20 classification problems used to assert the performance of 50 classifiers in [74], where the reported error rate for the best algorithm was about 15%. This is also the error rate reported for a SVM in [45]. As another reference, Madden reported an error of about 13% for the best classification methods presented in [38]. Note that this dataset is among the recommended problems in [73], being one of the CHIRP suite datasets.

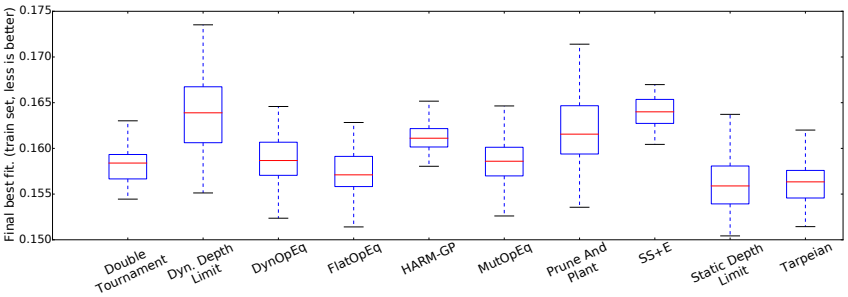
B.10.3 Results

Table 26: Detailed results of experiments conducted with the different bloat control methods for Adult.

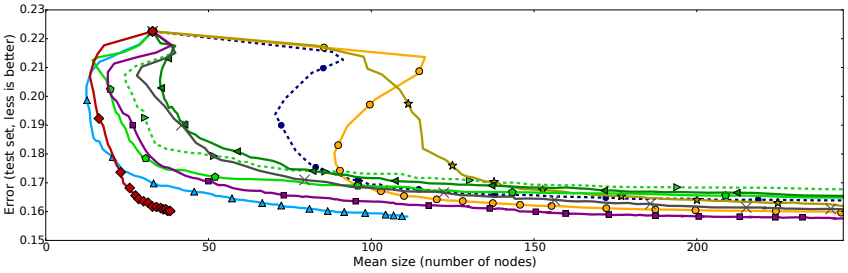
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.158	**0.158**	1.9	2.3	2.9
Dyn. Depth Limit	0.164	0.164	20.8	17.0	40.0
DynOpEq	0.159	0.160	26.5	41.1	52.4
FlatOpEq	0.157	**0.158**	12.3	19.4	13.6
HARM-GP	0.161	0.160	**1.0**	**1.0**	**0.2**
MutOpEq	0.159	0.159	7.2	8.3	14.1
Prune And Plant	0.162	0.162	17.0	11.1	36.2
SS+E	0.164	0.162	5.8	6.4	18.0
Static Depth Limit	**0.156**	**0.158**	6.6	7.9	13.8
Tarpeian	**0.156**	**0.158**	4.0	4.5	7.2
Reference	-	-	*65*	*4 630 486*	-



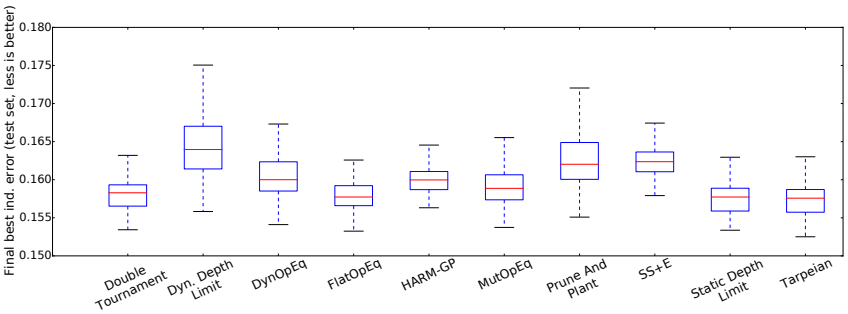
(a) Best classification error on the training set vs. mean size of the population



(b) Boxplot of the best-of-run classification error on the training set

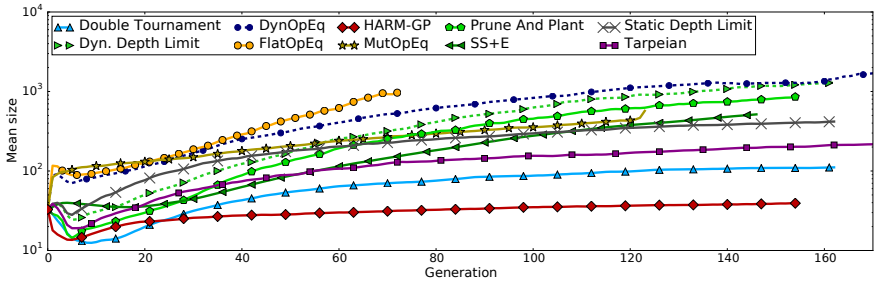


(c) Best classification error on the testing set vs. mean size of the population

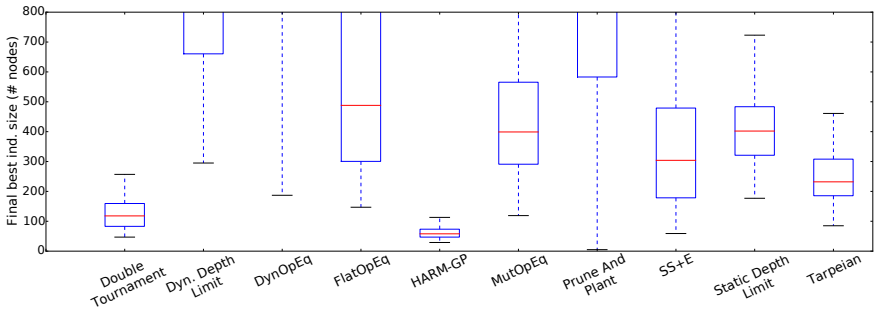


(d) Boxplot of the best-of-run classification error on the testing set

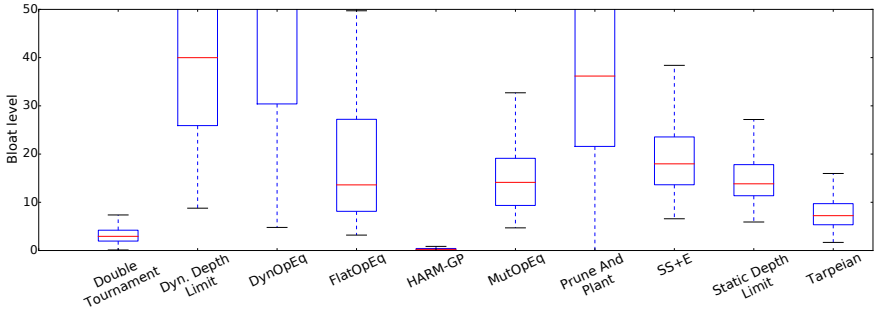
Figure 29: Performance and size results for Adult, over 100 runs: (a) and (c) plot the median of the classification error of the best individual found so far (according to the training set), against the corresponding average mean solution sizes, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the classification error of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



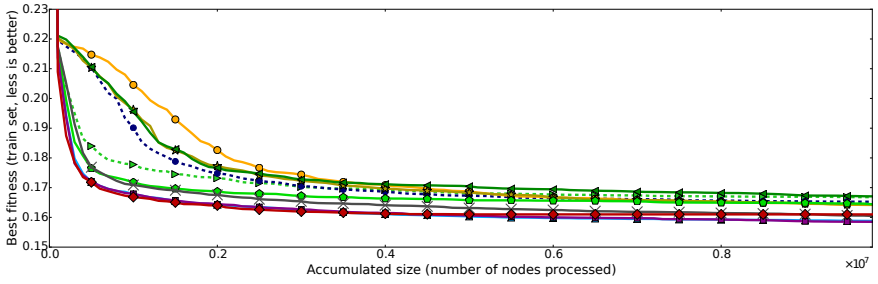
(a) Mean individual size vs. generation



(b) Boxplot of the best-of-run size



(c) Boxplot of the bloat level



(d) Best classification error on the training set vs. accumulated size

Figure 30: Size and bloat results for Adult: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best classification error on the training set achieved according to the accumulated size.

Table 27: Detailed results of experiments conducted with the different bloat control methods for Adult, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.157	**0.157**	2.0	2.4	3.2
Dyn. Depth Limit	0.164	0.164	19.7	16.4	40.0
DynOpEq	0.159	0.160	25.0	39.9	52.4
FlatOpEq	0.157	**0.158**	11.7	18.8	13.6
HARM-GP	0.160	0.159	**1.0**	**1.0**	**0.3**
MutOpEq	0.159	0.159	6.8	8.1	14.1
Prune And Plant	0.162	0.162	16.0	10.8	36.2
SS+E	0.164	0.162	5.4	6.2	18.0
Static Depth Limit	**0.156**	**0.158**	6.2	7.6	13.8
Tarpeian	**0.156**	**0.158**	3.7	4.4	7.2
Reference	-	-	*69*	*4 772 445*	-

B.11 Classification: Magic Gamma Telescope Dataset

B.11.1 Problem Description and Parameters Used

This is a classification problem, where the classifier aims to differentiate a signal representing some interesting phenomena for astrophysicists from the background noise. The classifier is fed an unbalanced dataset, each sample consisting of ten features. We used the following set of primitives: $\{+, -, *, /, \text{iflessthan}\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. The data is whitened beforehand, and the error rate was chosen as a fitness metric. The dataset can be downloaded from the UCI database at `http://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope`.

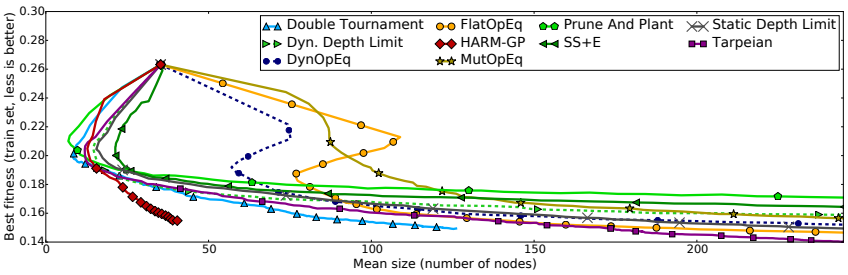
B.11.2 Previous Results and References

This problem is a difficult one, due to its unbalanced dataset, and the fact that its definition renders a false positive (qualifying background noise as a signal) significantly worse than a false negative (missing a signal). However, we can still use the classification error rate using uniform per-class loss to compare the performance of many classifiers. Bock *et al.* [10] reported an error rate of about 12% in the test set.

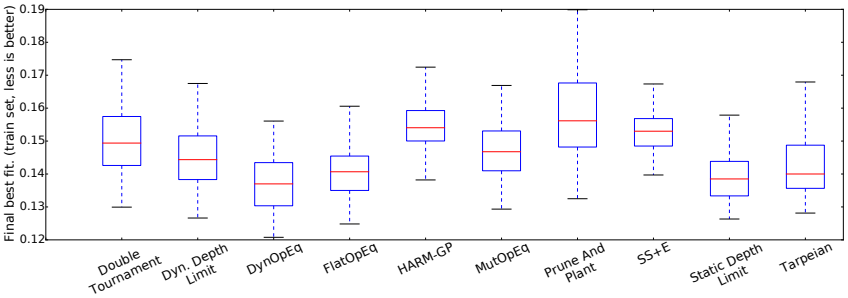
B.11.3 Results

Table 28: Detailed results of experiments conducted with the different bloat control methods for Magic Telescope.

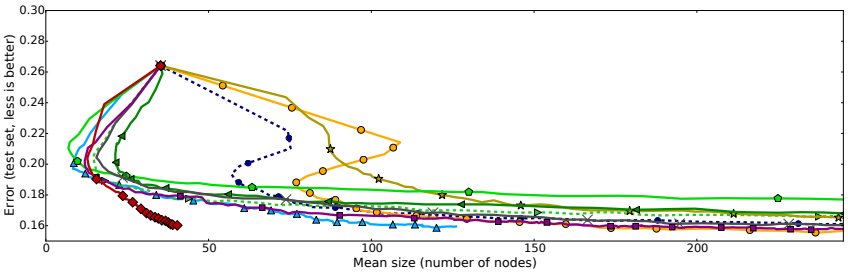
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.149	0.159	1.8	2.1	4.0
Dyn. Depth Limit	0.144	0.161	22.3	20.0	55.9
DynOpEq	**0.137**	**0.155**	19.4	40.2	52.4
FlatOpEq	0.141	**0.153**	7.8	15.3	14.7
HARM-GP	0.153	0.160	**1.0**	**1.0**	**0.3**
MutOpEq	0.147	0.163	7.2	8.8	16.2
Prune And Plant	0.156	0.167	14.9	13.3	46.8
SS+E	0.153	0.159	7.7	7.4	26.9
Static Depth Limit	**0.139**	**0.157**	5.8	7.7	15.5
Tarpeian	0.140	**0.158**	3.3	4.4	7.8
Reference	-	-	*78*	*4 858 638*	-



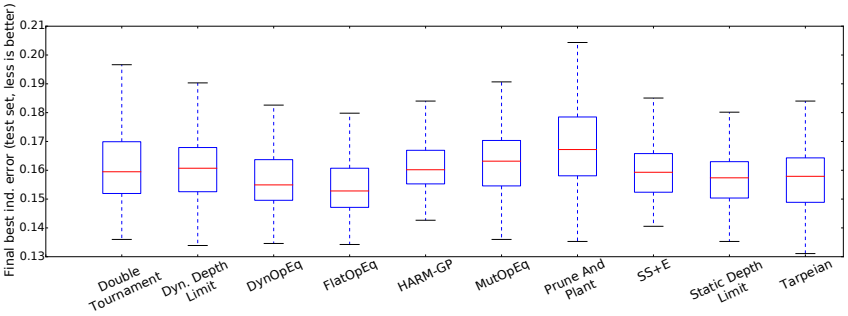
(a) Best classification error on the training set vs. mean size of the population



(b) Best-of-run training classification error boxplot

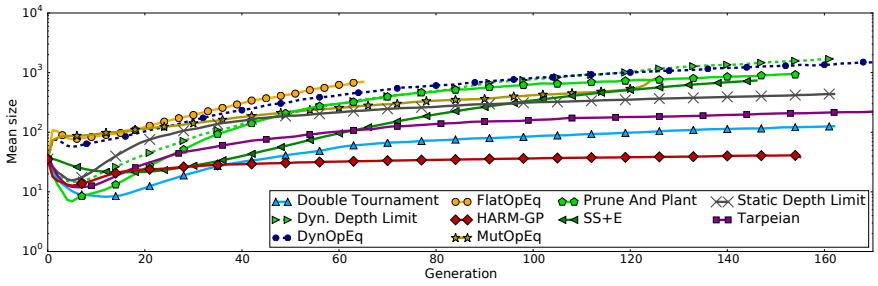


(c) Best classification error on the test set vs. mean size of the population

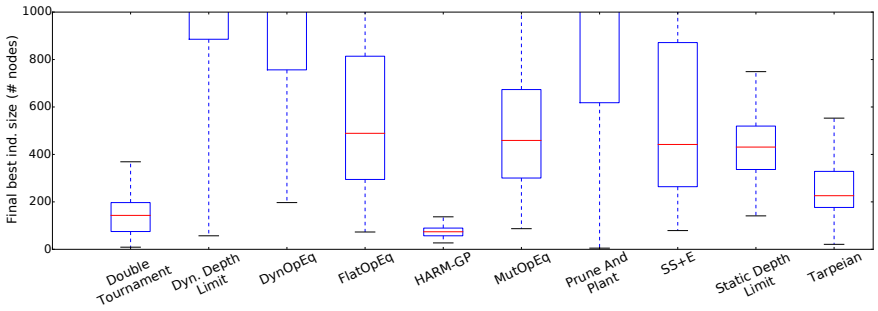


(d) Best-of-run test classification error boxplot

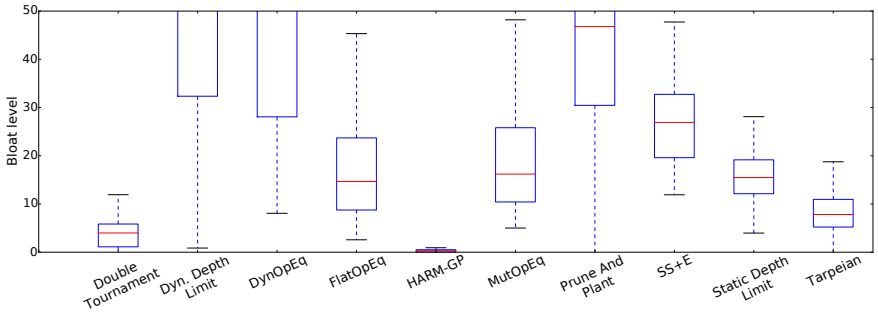
Figure 31: Performance and size results for Magic Telescope, over 100 runs: (a) and (c) plot the median of the classification error of the best individual found so far (according to the training set), against the corresponding average mean solution sizes, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the classification error of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



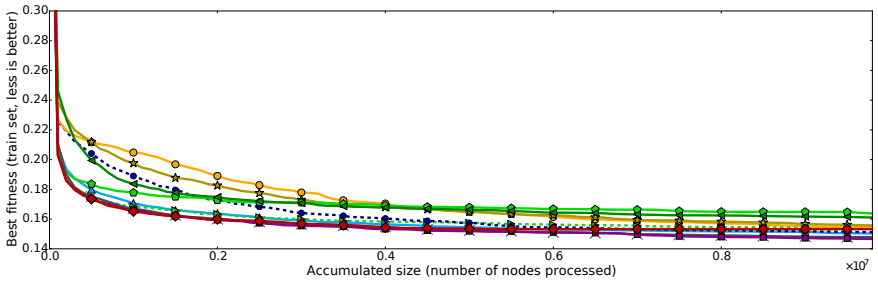
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best classification error on the training set vs. accumulated size

Figure 32: Size and bloat results for Magic Telescope: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best classification error on the training set achieved according to the accumulated size.

Table 29: Detailed results of experiments conducted with the different bloat control methods for Magic Telescope, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	**0.136**	**0.151**	2.3	2.7	5.0
Dyn. Depth Limit	0.144	0.161	22.7	19.2	55.9
DynOpEq	0.137	0.155	19.7	38.6	52.4
FlatOpEq	0.141	**0.153**	7.9	14.6	14.7
HARM-GP	0.141	**0.154**	**1.0**	**1.0**	**0.4**
MutOpEq	0.143	0.160	7.1	8.2	15.0
Prune And Plant	0.153	0.165	15.3	12.7	50.8
SS+E	0.149	0.157	7.7	7.3	26.2
Static Depth Limit	**0.134**	**0.155**	5.8	7.5	14.7
Tarpeian	**0.134**	**0.151**	4.0	4.7	9.0
Reference	-	-	*77*	*5 065 981*	-

B.12 Classification: Spambase

B.12.1 Problem Description and Parameters Used

This classification problem involves a spam filter. This dataset contains 4601 samples, with 57 features each. The task is to classify each of these samples (each sample containing the information concerning one email) into the spam or no-spam. We used the following set of primitives: $\{+, -, *, /, \text{iflessthan}\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. We randomly partitioned the dataset, retaining 30% as a test set, and using the remainder to train the classifier. Again, the classification error rate in training was chosen as a fitness function. The dataset can be downloaded from the UCI database at `http://archive.ics.uci.edu/ml/datasets/Spambase`.

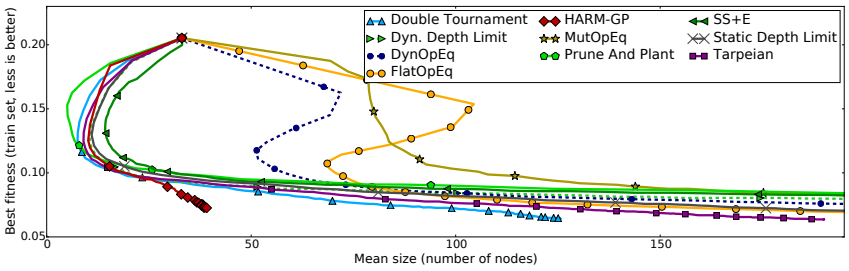
B.12.2 Previous Results and References

This problem has been widely explored. Lee *et al.* [34] have conducted a review of the relevant literature, and found classification error rates on the test set varying between 5% and 12%, sometimes with specific parameter optimization.

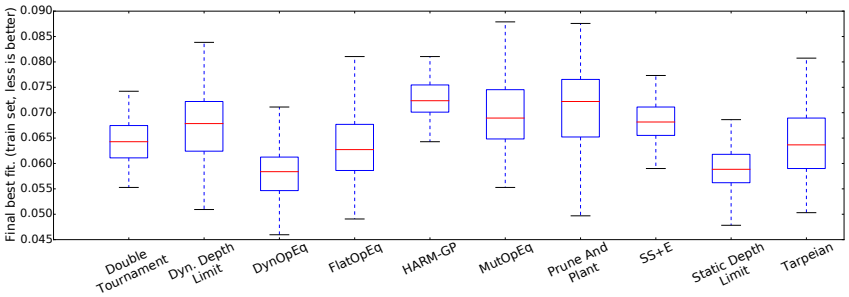
B.12.3 Results

Table 30: Detailed results of experiments conducted with the different bloat control methods for Spambase.

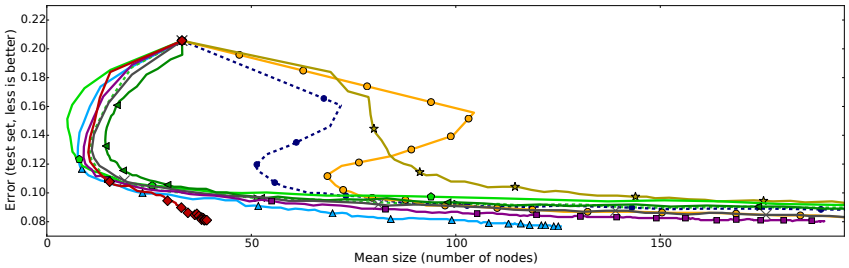
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.064	**0.077**	1.5	2.6	3.2
Dyn. Depth Limit	0.068	0.085	12.5	16.0	37.3
DynOpEq	**0.058**	0.081	11.2	31.3	27.5
FlatOpEq	0.063	0.080	5.5	12.1	10.4
HARM-GP	0.071	0.080	**1.0**	**1.0**	**0.3**
MutOpEq	0.069	0.087	4.8	7.3	12.6
Prune And Plant	0.072	0.085	16.0	19.1	45.1
SS+E	0.068	0.080	12.0	11.0	41.2
Static Depth Limit	**0.059**	**0.079**	3.7	6.6	10.8
Tarpeian	0.064	0.080	2.1	3.6	5.2
Reference	-	-	*98*	*5 210 493*	-



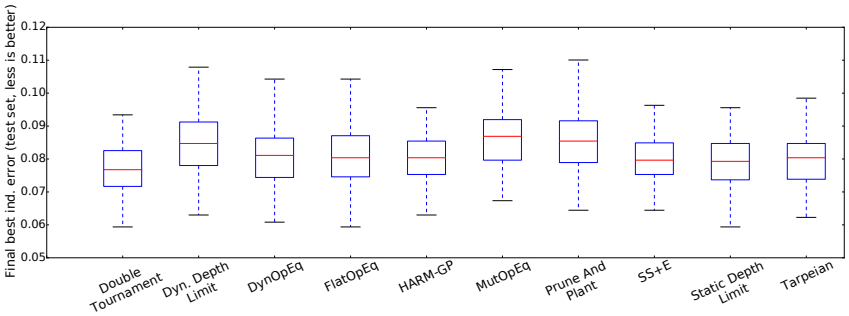
(a) Best classification error on the training set vs. mean size of the population



(b) Best-of-run training classification error boxplot

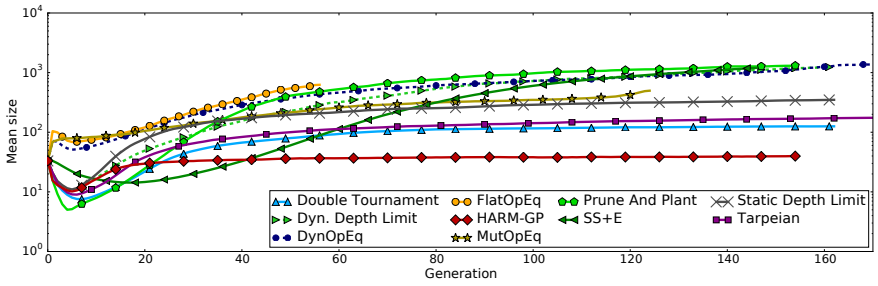


(c) Best classification error on the test set vs. mean size of the population

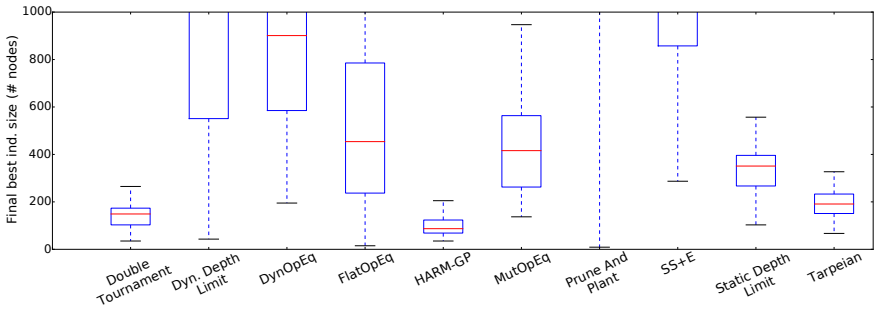


(d) Best-of-run test classification error boxplot

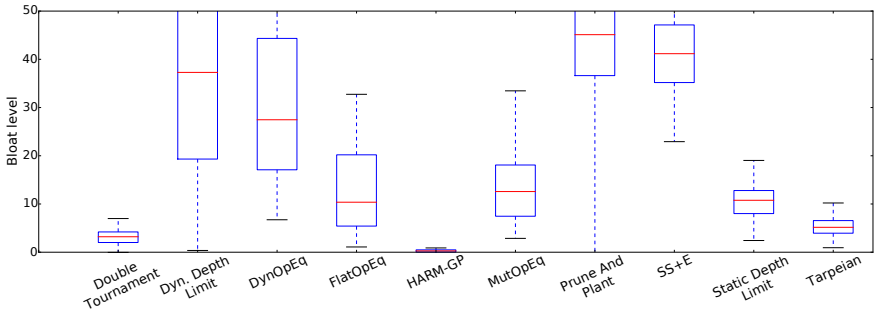
Figure 33: Performance and size results for Spambase, over 100 runs: (a) and (c) plot the median of the classification error of the best individual found so far (according to the training set), against the corresponding average mean solution sizes, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the classification error of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



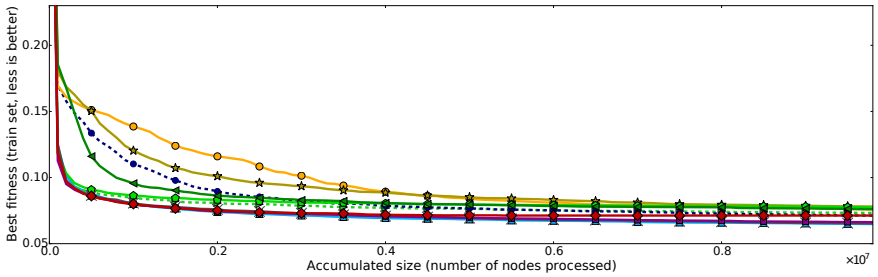
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best classification error on the training set vs. accumulated size

Figure 34: Size and bloat results for Spambase: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best classification error on the training set achieved according to the accumulated size.

Table 31: Detailed results of experiments conducted with the different bloat control methods for Spambase, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.064	**0.077**	1.6	2.5	3.2
Dyn. Depth Limit	0.068	0.085	13.1	15.5	37.3
DynOpEq	**0.058**	0.081	11.8	30.3	27.5
FlatOpEq	0.063	0.080	5.8	11.8	10.4
HARM-GP	0.068	0.081	**1.0**	**1.0**	**0.3**
MutOpEq	0.069	0.087	5.0	7.1	12.6
Prune And Plant	0.072	0.085	16.9	18.5	45.1
SS+E	0.068	0.080	12.7	10.6	41.2
Static Depth Limit	**0.059**	**0.079**	3.9	6.4	10.8
Tarpeian	0.064	0.080	2.2	3.5	5.2
Reference	-	-	*93*	*5 380 360*	-

B.13 Classification: Protein Structure Prediction (PSP-100)

B.13.1 Problem Description and Parameters Used

This problem involves the prediction of a protein folding, formulated as a binary classification problem. We used a dataset of 100 real-valued features consisting of 10 folds of about 230 000 instances each for training and 25 000 each for testing. The folds are distributed to the runs in a round-robin fashion. A run uses 10 000 training instances (sampled over the 230 000 instances of the fold) and 5 000 test instances (sampled over the 25 000 of the fold), a stratified random sampling being made to preserve the class ratio. Moreover, the same training and testing samples are used for all methods at a specific run i. The following set of primitives is used: $\{+, -, *, /, \text{iflessthan}\}$, with the division operator protected, plus an ephemeral constant in the range $[-1, 1]$. The training error rate was chosen as a fitness function. The dataset is available in the PSP database at `http://icos.cs.nott.ac.uk/datasets/psp/download.html`.

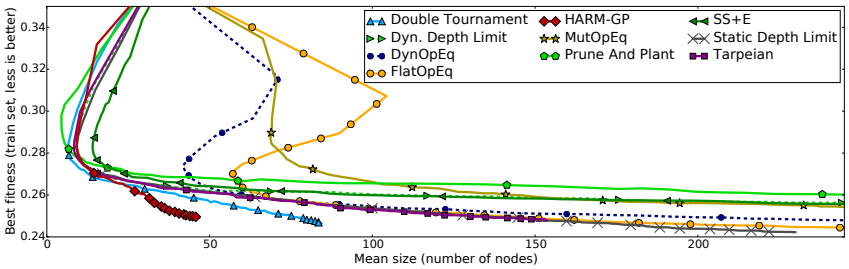
B.13.2 Previous Results and References

This dataset has been proposed in various variants to form a complete test suite with various difficulties (number of classes, number of features, class imbalance, high number of instances, etc.). For this reason, it is difficult to compare results because one must identify the exact dataset used. Yet, one can find some reference scores. In [5], an accuracy between 72 and 76% is reported for a SVM, while a GA approach achieves an accuracy between 70 and 73%. Franco *et al.* [20] reported an accuracy of 72.5% for the binary version of the classification problem. Note that this dataset is also recommended in the recent GP benchmarks review [73].

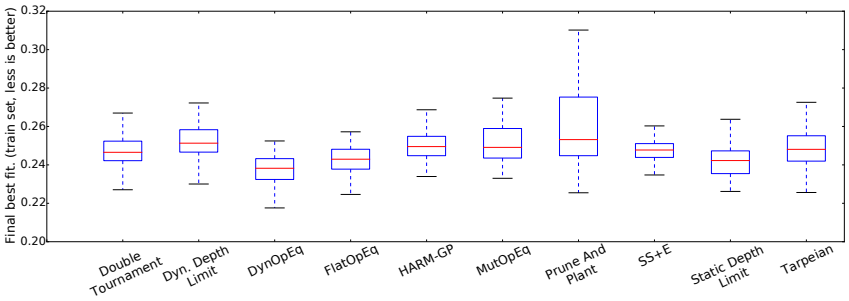
B.13.3 Results

Table 32: Detailed results of experiments conducted with the different bloat control methods for PSP100.

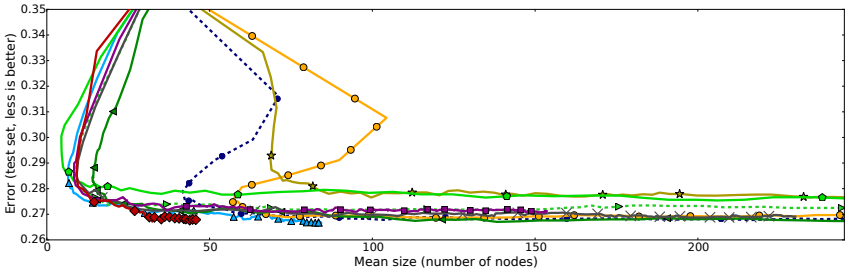
Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	0.246	**0.267**	**1.0**	1.7	2.7
Dyn. Depth Limit	0.251	0.272	8.4	10.5	40.7
DynOpEq	**0.238**	0.269	10.2	26.4	42.0
FlatOpEq	0.243	0.270	4.4	10.2	12.7
HARM-GP	0.248	0.268	**1.0**	**1.0**	**0.8**
MutOpEq	0.249	0.278	4.2	6.6	16.6
Prune And Plant	0.253	0.276	10.1	11.0	52.2
SS+E	0.248	**0.265**	9.3	7.8	53.1
Static Depth Limit	0.242	0.269	2.4	4.3	11.0
Tarpeian	0.248	0.271	1.7	2.7	5.7
Reference	-	-	*98*	*5 461 516*	-



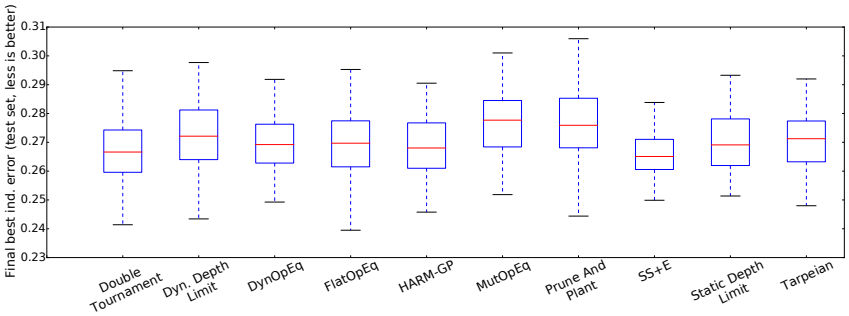
(a) Best classification error on the training set vs. mean size of the population



(b) Best-of-run training classification error boxplot

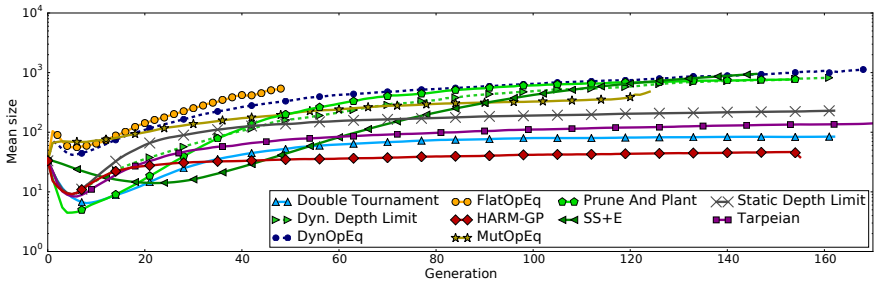


(c) Best classification error on the test set vs. mean size of the population

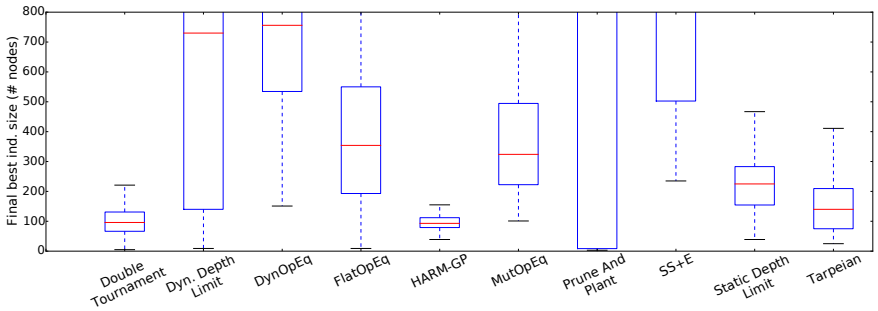


(d) Best-of-run test classification error boxplot

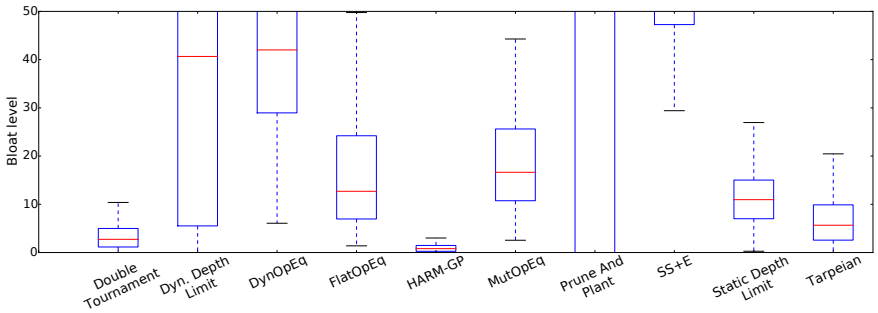
Figure 35: Performance and size results for PSP100, over 100 runs: (a) and (c) plot the median of the classification error of the best individual found so far (according to the training set), against the corresponding average mean solution sizes, respectively measured on the training set and the testing set, with each marker accounting for 10 000 evaluations, having some markers that may overlap one over another or extend to the right of the graph; (b) and (d) are boxplots of the classification error of the best-of-run individual (according to the training set), respectively measured against the training set and the testing set.



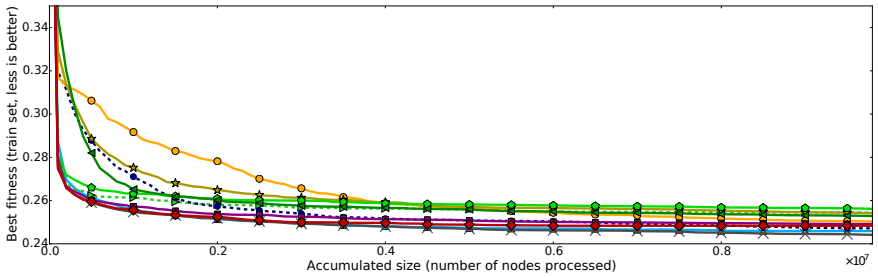
(a) Mean individual size vs. generation



(b) Best-of-run size boxplot



(c) Bloat level boxplot



(d) Best classification error on the training set vs. accumulated size

Figure 36: Size and bloat results for PSP100: (a) average over 100 runs of the mean individual size over the generations; (b) boxplot of the best-of-run individual size; (c) boxplot of the bloat level at the last generation; and (d) median over 100 runs of the best classification error on the training set achieved according to the accumulated size.

Table 33: Detailed results of experiments conducted with the different bloat control methods for PSP100, with equivalent computational effort.

Method	Train Error ($\bar{f}_x^*$)	Test Error ($\bar{g}_x^*$)	Size ($\bar{s}_x^*$)	Effort ($\bar{c}_x$)	Bloat ($\bar{b}_x$)
Double Tournament	**0.239**	**0.267**	1.3	2.3	4.0
Dyn. Depth Limit	0.251	0.272	8.5	11.7	40.7
DynOpEq	**0.238**	0.269	10.2	29.5	42.0
FlatOpEq	0.243	0.270	4.4	11.4	12.7
HARM-GP	0.242	**0.267**	**1.0**	**1.0**	**0.2**
MutOpEq	0.248	0.277	4.0	7.0	15.7
Prune And Plant	0.253	0.276	10.1	12.3	52.2
SS+E	0.248	**0.265**	9.3	8.7	53.1
Static Depth Limit	**0.237**	**0.268**	2.6	5.0	11.8
Tarpeian	0.240	0.270	2.1	3.8	8.1
Reference	-	-	*98*	*4 882 353*	-