 ISBN: 978-989-95618-4-7

# Experiments on Controlling Overfitting in Genetic Programming

Ivo Gonçalves[1] and Sara Silva[2,1]

[1]CISUC, DEI/FCTUC, University of Coimbra, Portugal
[2]INESC-ID Lisboa, IST, Technical University of Lisbon, Portugal
icpg@student.dei.uc.pt
sara@kdbio.inesc-id.pt

**Abstract.** One of the most important goals of any Machine Learning approach is to find solutions that perform well not only on the cases used for learning but also on cases never seen before. This is known as generalization ability, and failure to do so is called overfitting. In Genetic Programming this issue has not yet been given the attention it deserves, although the number of publications on this subject has been increasing in the past few years. Here we perform several experiments on a small and yet difficult toy problem specifically designed for this work, where a perfect fitting of the training data inevitably results in poor generalization on the unseen test data. The results show that, on this problem, a Random Sampling Technique with parameter settings that maximize the variation between generations can significantly reduce overfitting when compared to a standard GP approach. We also report the results of some techniques that failed to achieve better generalization.

**Keywords:** Genetic Programming, Overfitting, Generalization

## 1 Introduction

Genetic Programming (GP) is an evolutionary computation technique that automatically solves problems without needing to know the structure of the solution in advance [2]. One of the areas in GP that has been recently recognized as an open issue that needs to be addressed in order for GP to realize its full potential is the one of generalization [6]. Generalization is the ability to find solutions that perform well not only on the cases used for learning but also on cases never seen before. Achieving good generalization ability is one of the most important goals of any Machine Learning (ML) approach such as GP. Failure to generalize well is called overfitting, when the solution performs well on the training cases but poorly on the test cases. This indicates that the underlying relationships of the whole data were not learned, and instead a set of relationships existing only on the training cases were learned, but these have no correspondence over the whole possible set of cases.

The issue of generalization in GP has not yet received the attention it deserves. Other non-evolutionary ML methods have dedicated a far larger amount

of research effort to it, although the number of publications dealing with overfitting in GP has been increasing in the past few years. Notably, in Koza [7] most of the problems presented did not use separate training and test data sets, so performance was never evaluated on unseen cases [8]. Part of the lack of generalization efforts can be related to another issue occurring in GP - bloat. Bloat can be defined as an excess of code growth without a corresponding improvement in fitness [9]. This phenomenon occurs in GP as in most other progressive search techniques based on discrete variable-length representations. Bloat was one of the main areas of research in GP, not only because its occurrence hindered the search progress but also because it was hypothesized, in light of theories such as Occam's razor and the Minimum Description Length, that a reduced code size could lead to a better generalization ability. Researchers had a common agreement that these two issues were related and that counteracting bloat would lead to positive effects on generalization ability. This, however, has been recently challenged. Contributions show that bloat free GP systems can still overfit, while highly bloated solutions may generalize well [10]. This leads to the conclusion that bloat and overfitting are in most part two independent phenomena. In light of this finding, new approaches to improve GP generalization ability are in need, particularly those not based on merely biasing the search towards shorter solutions. It was this challenge that motivated the development of the current work, which is only one of the first steps of a larger research effort directed at understanding and controlling overfitting in GP.

Section 2 reviews the state of the art of the generalization issue in GP. Section 3 reports the proposed techniques and the experimental settings. Section 4 presents and discusses the results. Section 5 concludes and presents the future work.

## 2 State of the art

The most common approaches to reducing overfitting in GP are those based on biasing the search towards shorter solutions. Becker and Seshadri [11] proposed adding a complexity penalty factor to the fitness function. Mahler et al. [12] explored to what extent Tarpeian bloat control affects GP generalization ability. Gagné et al. [13] tested the application of parsimony pressure. Cavaretta and Chellapilla [14] used a low-complexity-bias algorithm that uses a modification in the fitness function meant to penalize larger individuals. Zhang et al. [15] also addressed the relationship between size and generalization performance by using a fitness function with two components: fitting error and size.

More recent approaches bias the search process to less complex solutions. In these approaches complexity is not simply defined as solution size. Vladislavleva et al. [16] proposed a complexity measure called order of nonlinearity. This measure adopts the notion of the minimal degree of the best-fit polynomial, approximating an analytical function with a certain precision. The main objective behind the proposed complexity measure is to favor smooth and extrapolative behavior of the response surface and to discourage highly nonlinear behavior,

which is unstable towards minor changes in inputs and is dangerous for extrapolation. Vanneschi et al. [17] proposed a functional complexity measure based on the classic mathematical concept of curvature. Informally, the curvature of a function can be defined as the amount by which its geometric representation deviates from being straight. This complexity measure expresses the complexity of a function by counting the number of different slopes. Other complexity measures have been recently studied [18, 19].

Also recently, approaches based on similarities between solutions have started to appear. Uy et al. [20] proposed a Semantic Similarity based Crossover approach which is based on the Sampling Semantics Distance between two trees (or subtrees), which is calculated by choosing $N$ random points (fitness cases) and calculating the mean absolute difference between each corresponding points on the two trees. The authors argue that the exchange of subtrees is most likely to be beneficial if the two subtrees are not too similar or too dissimilar. Vanneschi and Gustafson [21] proposed avoiding solutions similar to already known overfitted solutions. The proposed method (repGP) keeps a list of overfitting individuals (called repulsors) and prevents any new individual to enter the next generation if they are similar to any of the known repulsors.

Various other approaches were proposed. A simple and elegant idea was proposed by Da Costa and Landry [22]. The idea is to relax the training set by allowing a wider definition of the desired solution which translates into considering not only the desired output $y$ correct but allow a more broader range to be considered, i.e. allow any output in the range $[ymin, ymax]$. Chan et al. [23] proposed a statistical method called Backward Elimination that works by eliminating insignificant terms in polynomials models such as those produced by GP. Nikolaev et al. [24] proposed several techniques to balance the statistical bias and variance. In the context of financial applications, Chen and Kuo [25] proposed a measure of degree of overfitting based on the extracted signal ratio. Foreman and Evett [26] proposed Canary Functions, where the idea is to measure overfitting during the run by using a validation set. When overfitting starts to occur the search process is stopped. Vanneschi et al. [27] argued that using GP with a multi-optimization approach can enhance the generalization ability of the resulting solutions. This approach uses two other criteria besides the traditional sum of errors. These are: the correlation between outputs and targets (to maximize) and the diversity of pairwise distances between outputs and targets (to minimize). Robilliard and Fonlupt [28] applied a method called Backwarding that goes back as much as needed in the evolution process until the point that overfitting is not yet very relevant. This is achieved by saving two copies of the solutions: one copy for the best solution on the training set and another copy for the best solution on the validation set. At the end of the GP process the best saved solution for the validation set is returned. Finally, in the context of the Compiling Genetic Programming System, Banzhaf et al. [29] showed the positive influence of the mutation operator in generalization ability.

Although these and a number of other works have addressed the issue of overfitting in GP, they appear as a set of isolated efforts scattered along the

 ISBN: 978-989-95618-4-7

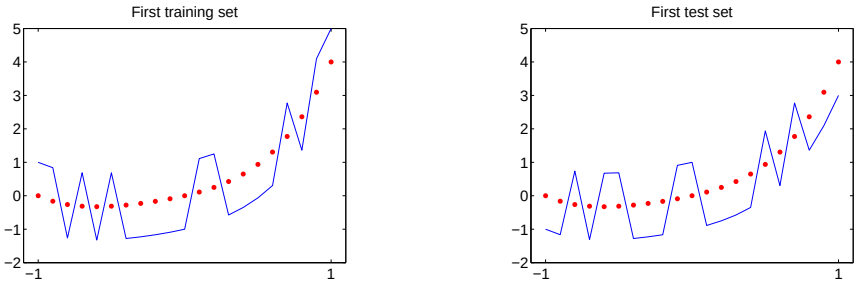


**Fig. 1.** First pair of training (left) and test (right) data sets used in our experiments (blue lines) drawn with the original quartic function (red dots)

years and among different applications. Nevertheless, as GP matures and slowly becomes a mainstream ML approach, also the overfitting issue is slowly becoming a central research subject.

## 3 Experiments

### 3.1 Data

Given that GP is still lacking any benchmarks specifically designed for studying generalization and overfitting, we have designed a very simple toy problem where to run our experiments. It would be possible, of course, to use a real world problem as a test case, however the unpredictable errors and inaccuracies of real data could bias the results and prevent us from drawing solid conclusions.

We took one of the simplest GP benchmark problems commonly used to study the learning ability of GP, the symbolic regression of the quartic polynomial $x^4 + x^3 + x^2 + x$ using only 21 equidistant points in the interval $-1$ to $+1$. To the expected output value of each point we randomly added or subtracted the fixed value 1. We did this 10 times to create 10 different training sets, and an additional 10 times to create their 10 respective test sets. We performed 30 independent runs for each of the experiments described in the next section, in each one using one training + test pair, each pair being used in 3 different runs. Fig. 1 plots the modified training and test data used as the first pair, together with the original data.

Note that this is not the usual way to partition a data set into training + test. What is usually done is to use a fraction of the data points for training, reserving the rest for testing. However, we wanted to ensure that, in our toy problem, a perfect fit of the training data would inevitably result in overfitting, so we used the same set of points for both training and testing, but with different expected outputs. It can be argued that this problem is impossible to solve, but this is exactly what most real world problems are. Our goal is to develop a technique that avoids overfitting such that the final solution will not be perfect in either training or test set, but will have the same error in both sets. That solution is

precisely the one that perfectly fits the original quartic polynomial before the noise was added.

Also note that, although originally simple, the newly designed noisy quartic problem is now very difficult to solve. In particular, our goal is very difficult to achieve, since each of the 21 modified points heavily biases the search towards solutions that are wrong, i.e., that overfit. It is possible to decrease the difficulty of the problem by using more points and applying smaller changes to their original value, but we have used it exactly as described above.

### 3.2 Techniques

Two main types of techniques are proposed: an extension to the Random Sampling Technique and several validation set based approaches.

The first proposal is to extend the Random Sampling Technique (RST) used by Liu and Khoshgoftaar [1]. This technique had been previously used to improve the speed of a GP run [30], however in [1] it was used to reduce overfitting in a classification task in the context of software quality assessment. In the RST, the training set is never entirely used in the search process. Instead, at each generation, a random subset of the training data is chosen and evolution is performed taking into account the fitness of the solutions in this subset only. This implies that only individuals that perform well on various different subsets will remain in the population. It is expected that, since these surviving individuals perform reasonably well on different subsets, they have captured the underlying relationships of the data instead of overfitting it. In [1] the size of each random subset was 50% of the whole training set. The approach proposed here will be more flexible in two main aspects. Firstly, the size of the random subset can be defined as any percentage of the training set. Secondly, the rate at which a new random subset is chosen can be defined as either being at each $N$ generations or as a percentage of the total number of generations. These two RST parameters are respectively labelled as Random Subset Size (RSS) and Random Subset Reset (RSR). In this extended approach they can be defined as any value, as opposed to their static nature in the abovementioned work. Experiments are performed in order to find promising combinations of these parameters.

Two simple variants of the RST are also experimented. These are RST NENR and RST Std Dev. RST NENR is simply the RST without elitism or any kind of replication of individuals into the next generation. NENR stands for No Elitism and No Replication. RST Std Dev uses a two component fitness function as opposed to the Root Mean Squared Error (RMSE) used on the regular RST. The first component is the RMSE and the second is the standard deviation (Std Dev) of the differences between outputs and targets. Both components are weighted equally.

The other proposed approaches are based on the usage of a validation set during the search process and the inclusion of information about this set in the fitness function. The validation set is formed by taking 50% random samples from the training set, which for our toy problem results in very small training and validation sets (approximately 10 samples each). The techniques presented

in this section are called Validation Start (VS), Validation Evaluation (VE), Validation Std Dev Start (VSDS), Validation Std Dev Evaluation (VSDE) and Validation Complexity Start (VCS).

VS and VE define fitness as a weighted combination of the RMSE on the training set and the absolute difference between the RMSE in the training set and the RMSE in the validation set. The goal is to optimize the RMSE on the training set while maintaining similar RMSE values on the validation set without specifically measuring it on this set. These components are weighted by different but correlated weights $w_1$ and $w_2$. $w_1$ is randomly chosen between 0 and 1, and the $w_2$ is such that $w_1 + w_2 = 1$. VS and VE differ from each other in the moment when the weights are chosen. In VS this happens only once at the start of the run. In VE new weights are chosen every time there is a fitness evaluation.

VSDS and VSDE build on VS and VE by adding another component to the fitness function. This component is Std Dev, the same used in RST Std Dev, calculated on the original training data (current training and validation sets), with the goal of promoting smoothness of the solutions. The three weights $w_1$, $w_2$ and $w_3$ are chosen randomly with $w_1 + w_2 + w_3 = 1$. As with VS and VE, both a single (at start) and a periodical (at each fitness evaluation) generation of weights are possible, respectively designated as VSDS and VSDE.

VCS builds on VS by also adding one fitness component to the fitness function. This component is a measure of solution complexity based on the concept of curvature [17] mentioned in Section 2 and is calculated on the training set only. All the weights sum to 1 and are chosen once at the start of the run. The goal is also to promote the smoothness of the solutions.

Standard GP with RMSE as the fitness function is used as the baseline technique. Standard NENR refers to Standard GP without elitism or replication. Standard Std Dev refers to Standard GP with a fitness function using the same two components (RMSE and Std Dev) as RST Std Dev.

### 3.3 Tools and Parameters

All the experiments were performed using a modified version of GPLAB [4], a Genetic Programming Toolbox for MATLAB. Statistical significance of the null hypothesis of no difference was determined with pairwise Kruskal-Wallis non-parametric ANOVAs at p=0.05. A non-parametric ANOVA was used because the data is not guaranteed to follow a normal distribution. For the same reason, the median was preferred over the mean in all the evolution plots shown in the next section. The median is also more robust to outliers.

The experimental parameters are provided in Table 1. Furthermore, crossover and mutation points are selected with uniform probability. Unless stated otherwise, fitness is calculated as the Root Mean Squared Error (RMSE) between outputs and targets.

| | |
|---|---|
| Runs | 30 |
| Population | 500 |
| Generations | 100 |
| Crossover operator | Standard subtree crossover, probability 0.9 |
| Mutation operator | Standard subtree mutation, probability 0.1, new branch maximum depth 6 |
| Tree initialization | Ramped Half-and-Half [2], maximum depth 6 |
| Function set | +, -, *, and /, protected as in [2] |
| Terminal set | Input variable and no random constants |
| Selection for reproduction | Lexicographic Parsimony Pressure [3], tournaments of size 10 |
| Elitism | Replication rate 0.1, best individual always survives |
| Maximum tree depth | 17 |

**Table 1.** GP parameters used in our experiments

## 4 Results and Discussion

This section presents and discusses the results achieved following the experimentation chronological order and the reasoning behind each new experiment. For the remainder of this paper, the terms training fitness and test fitness are to be interpreted in the following way: training fitness is the fitness of the best individual in the training set; test fitness is the fitness of that same individual in the test set.

### 4.1 Experiments on RST

Our first set of experiments was performed in order to determine what would be the most promising parameter values for the RST. RSS was the first parameter tested on preliminary experiments, where values 50%, 40%, 30%, 20%, 10%, 5% and 1 were tested (1 means each subset contains only 1 data sample, not 1% of the samples). These initial experiments suggested that lower values of RSS are normally better in terms of test fitness. They are also worse, as expected, in terms of training fitness. The best value found in the initial experiments was precisely 1, the one that uses only one fitness case. Considering these preliminary results, and since an exhaustive experimentation would be impractical, we decided to deepen the study of the RSS influence only for values 1 and 50%. The first is justified by the preliminary good results and the second is justified for being a good middle ground to experiment with, and also the value used in [1]. The same preliminary experiments also suggested that an RSR value of 1 is better than 5, so we fixed RSR to value 1 while varying the RSS value. Results show that an RSS of 1 is statistically better than an RSS of 50% in terms of test fitness (Fig. 2, left). As for training fitness (Fig. 2, right), the reverse is true,

 ISBN: 978-989-95618-4-7

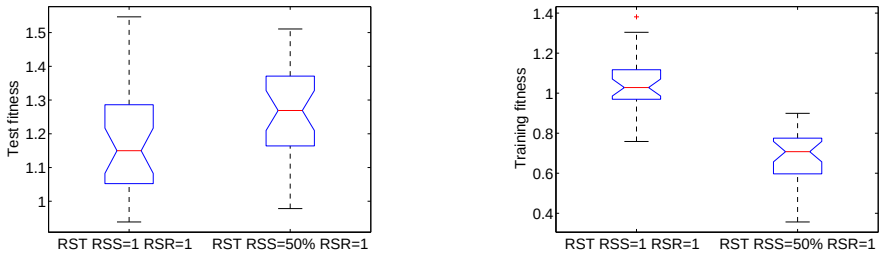


**Fig. 2.** Boxplot of test fitness (left) and training fitness (right) for RST RSS=1/50% RSR=1

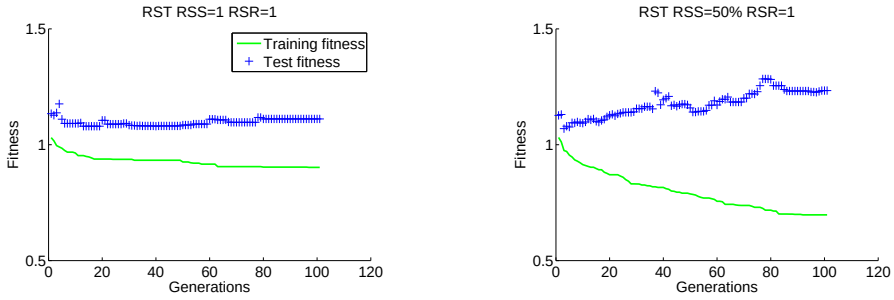


**Fig. 3.** Evolution plots for RST RSS=1 RSR=1 (left) and RST RSS=50% RSR=1 (right)

being value 50% statistically better than value 1. Test fitness results confirm the validity of the initial impression over the RSS value 1. It seems that using very few fitness cases (in this case only one) is, at least in this data set, enough to improve the generalization ability. Training fitness results come as no surprise since using more learning cases intuitively facilitates the learning process and, consequently in this case, aggravates the overfitting issue.

The evolution plots (Fig. 3) show that an RSS of 1 produces a constant or slightly changing gap between the training and test fitness values. On the other hand, value 50% presents a constantly widening gap between both values. It seems that the higher the RSS value, the larger the gap, i.e., the more overfitting, so the value 1 was concluded to be the best value for the RSS.

The second step in this first set of experiments was to find a suitable value for RSR. As mentioned before, preliminary experiments showed that the value 1 seemed more promising than the value 5, so the following values of RSR were tested: 1, 5, 10 and 20. For 100 generations per run, this implies that we are ranging the algorithm from choosing a new subset 100 times per run (RSR value 1) to choosing a new subset 5 times per run (RSR value 20). Given the previous results, the RSS is set to value 1. Results show a tendency for test fitness to worsen as the RSR is increased (Fig. 4, left), although the difference

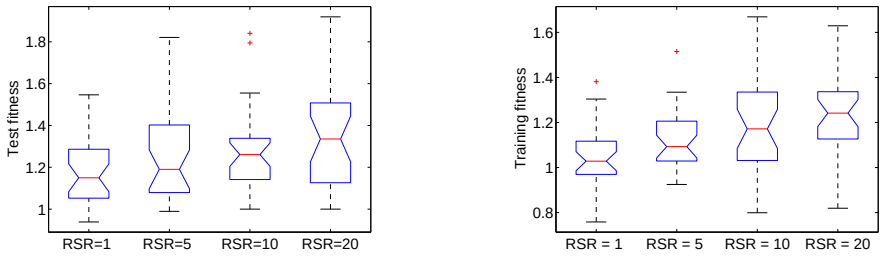


**Fig. 4.** Boxplot of test fitness (left) and training fitness (right) for RST RSS=1 RSR=1/5/10/20

only achieves statistical significance for the comparison between values 1 and 20. A similar tendency is seen in the training fitness results (Fig. 4, right), where value 1 is shown to be the best, being statistically better than every other value. Also, value 5 is statistically better than value 20.

When looking at the corresponding evolution plots, only little differences can be observed for the different values of RSR, and the most noticeable difference is a better training fitness achieved with RSR = 1. We do not include here the plots obtained with RSR = 5/10/20 and instead refer the reader to Fig. 3 (left) that shows the results for RSR = 1. RSS seems to be a much more influential parameter than RSR.

The results from this first set of experiments allow us to conclude that the best parameter configuration, among the tested values, is setting both RSS and RSR to value 1. It seems that using a low RSS value helps achieving better test fitness while, in the process, resulting in worse training fitness. As for the RSR, low values seem to be suited to achieve better training fitness while preserving test fitness. We conclude that having a low RSS value is not that damaging to the training fitness as long as the changes on the random subset occur often enough (low RSR value). Having low values on both RSS and RSR appears to be a good compromise between improving test fitness and not having excessive decline on training fitness.

### 4.2 Comparison with Standard GP

The next step was to assess the RST performance against the baseline technique - standard GP. Results show that the RST is statistically better on test fitness (Fig. 5, left), which supports our initial view on the potential of the RST for improving generalization ability. Using less fitness cases in each generation and inducing frequent changes on the chosen fitness cases is indeed helpful in improving test fitness. On training fitness, standard GP is statistically better (Fig. 5, right). This last result is not surprising for the same reason presented in the comparison between values 50% and 1 of RSS shown earlier (using more fitness cases facilitates the learning process). The evolution plot for Standard GP (not

 ISBN: 978-989-95618-4-7

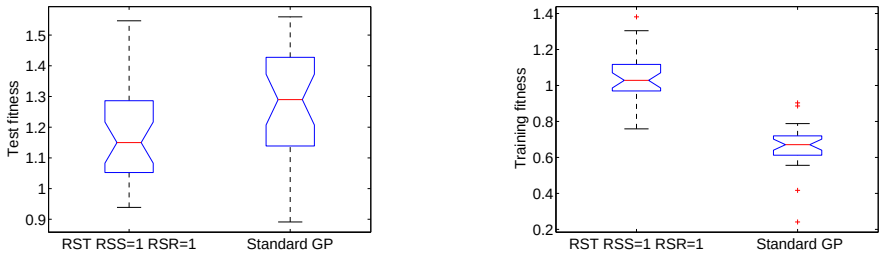


**Fig. 5.** Boxplot of test fitness (left) and training fitness (right) for RST RSS=1 RSR=1 and Standard GP

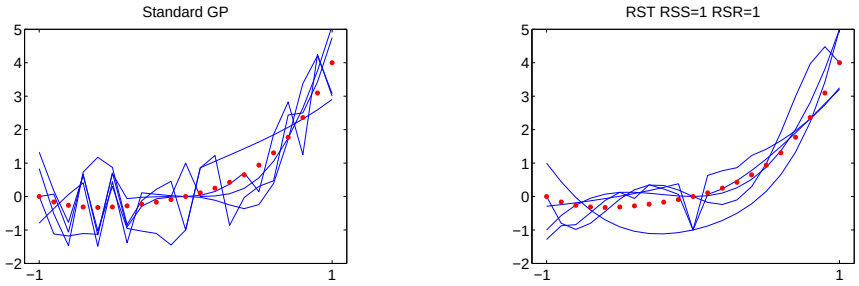


**Fig. 6.** Best solutions obtained in the first five training sets (blue lines) drawn with the original quartic function (red dots) for Standard GP (left) and RST RSS=1 RSR=1 (right)

shown) is similar to the one of RST RSS=50% RSR=1% (Fig. 3 (right)) with a large and constantly widening gap between the training and test fitness values.

We looked deeper into the results and observed the best solutions obtained by Standard GP and RST RSS=1 RSR=1 in the five first training sets of our toy problem (Fig. 6). The differences in the shape of the solutions are obvious. Standard GP finds rugged functions that closely fit the noisy training data (which is not plotted), while RST finds smoother functions that approximate the original noiseless data (red dots) much better. Another important difference between both techniques is found at the population level. In the Standard GP population there is a large amount of individuals that are equal or very similar to the best individual. In RST the top best individuals of the population usually show large differences between them, suggesting a much higher diversity at least among the best of the population, and possibly a very quick "rotation" of the best as the training subset changes.

### 4.3 Variations on Standard GP and RST

The smoothness and diversity of best solutions found by RST, described above, prompted our next experimentation steps. In an attempt to replicate or further promote each of these two features (smoothness and diversity) we applied small variations to Standard GP and RST, namely a fitness function that also tries to minimize the standard deviation of the differences between outputs and targets, and a non elitist version where no individual is ever copied into the next generation. As described in Section 3, we designate these variants as Standard Std Dev, Standard NENR, RST Std Dev, and RST NENR. Both RST variants use the best configuration found previously: both RSS and RSR set to 1.

Standard NENR revealed to be statistically better than Standard GP on training fitness, but on test fitness there were no differences. This suggests that the elitism is driving the search into local optima, possibly due to a loss of diversity that is particularly obvious among the elite part of the population. We looked at some examples of solutions found by Standard Std Dev, comparing them to the ones found by Standard GP (not shown), and found them to be slightly smoother visually, but not enough to award statistical significance. This small smoothing effect does not accumulate with the smoothness of RST, as RST Std Dev is not statistically different from the regular RST. In fact, the regular RST and its two variants did not show any significant difference between them, either in training or test fitness. This implies that these variations do not influence the learning or the generalization. Regarding RST NENR, this conclusion is not so surprising because the diversity was already high in the regular RST. However, the RST Std Dev results were somewhat unexpected, and it is not clear why the modified fitness function did not affect the learning process. We hypothesize that assigning equal weights to RMSE and Std Dev may be insufficient, and if Std Dev was given a higher weight the results could be different. This matter was not yet further explored. We can, however, view the comparison between RST and RST Std Dev in light of the Occam's Razor and reason that, since adding a new fitness component does not bring anything new, then we prefer the simpler method - RST. Given the previous arguments we shall keep regular RST as our chosen technique.

It is important to emphasize that, although RST Std Dev and RST NENR did not improve RST results, they are both statistically better on test fitness, and statistically worse on training fitness, than Standard GP. Also, although Standard NENR was better than Standard GP on training fitness, we do not consider this to be relevant in terms of generalization so we keep Standard GP as the baseline technique.

### 4.4 Additional Techniques

The last set of experiments was intended to measure the performance of the validation based techniques (described in Section 3) against Standard GP and RST. The five techniques under analysis are: Validation Start (VS), Validation

 ISBN: 978-989-95618-4-7

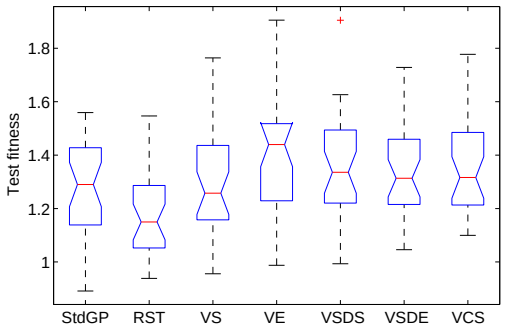


**Fig. 7.** Boxplot of test fitness for Standard GP, RST RSS=1 RSR=1, VS, VE, VSDS, VSDE and VCS (see Section 3.2 for the description of the acronyms)

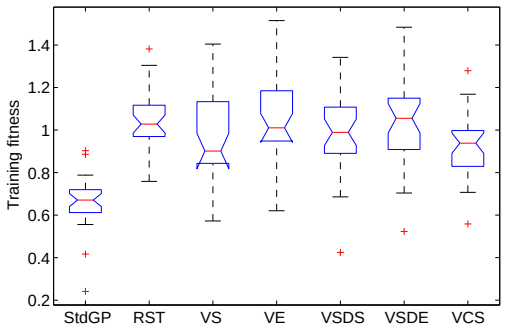


**Fig. 8.** Boxplot of training fitness for Standard GP, RST RSS=1 RSR=1, VS, VE, VSDS, VSDE and VCS (see Section 3.2 for the description of the acronyms)

Evaluation (VE), Validation Std Dev Start (VSDS), Validation Std Dev Evaluation (VSDE) and Validation Complexity Start (VCS). The results (Figs. 7 and 8) show that none of these techniques is able to achieve better test fitness than Standard GP. VE is even statistically worse than Standard GP on test fitness. Furthermore, all of the techniques are worse than Standard GP on training fitness. As expected given the results reported previously, RST is statistically better on test fitness than all the other techniques. On training fitness VS and VCS are statistically better while the others are neither better nor worse.

The evolution plots of these five techniques (not shown) are rather similar to each other. They lie between the evolution plots of the RST and the Standard GP. The gaps between training and test fitness are larger that on the RST but smaller than on Standard GP. This, however, does not result in any significant advantage in terms of test fitness against Standard GP.

These results show that the five validation based techniques are not interesting in terms of generalization ability. However, this may have happened because of the small number of samples in the training set, so we believe they deserve further investigation.

## 5 Conclusions and Future Work

In this work we have taken some of our first steps towards the understanding and controlling overfitting in GP in order to improve the generalization ability of its solutions. We have summarized the current state of the art on this subject, which is composed of a set of isolated efforts scattered along the years and among different applications. However we have also noticed that, as GP matures and slowly becomes a mainstream ML approach, also the overfitting issue is slowly becoming a central research subject.

Given that GP is still lacking any benchmarks specifically designed for studying generalization and overfitting, we have designed a small but difficult toy problem where we could run all our experiments without the danger of drawing biased conclusions from real world inaccurate data. In this problem, a perfect fitting of the training data inevitably results in poor generalization on the unseen test data, thus replicating the difficulties that are usually present in real applications.

We have proposed two main types of techniques to control overfitting in GP, using a standard GP approach as the baseline for comparison. The first set of techniques is an extension of the previously published Random Sampling Technique (RST), previously used to improve the speed of a GP run [30] and more recently to reduce overfitting in the context of software quality classification [1]. In the original RST the training set is never entirely used in the search process. Instead, at each generation, a random subset of the training data is chosen and evolution is performed taking into account the fitness of the solutions in this subset only. In our extended implementation of RST we have introduced two parameters that control the size of the subset (RSS) and how often its samples are changed (RSR). The results have shown that, on our toy problem, RST with parameter settings that maximize the variation between generations (RSS=1 and RSR=1) can significantly reduce overfitting when compared to a standard GP approach. Slight variations were also tested, with the goal of promoting the smoothness of the functions proposed as solutions and the diversity of the best individuals in the population, and both performed equally well.

The second set of techniques we proposed are based on the usage of a validation set during the search process and the inclusion of information about this set in the fitness function. The validation set is formed by taking half of the samples from the training set, which for our toy problem results in very small training and validation sets. This may have been the reason why none of these techniques succeeded in improving generalization ability when compared to Standard GP.

The subject of generalization and overfitting in GP is crucial in order for GP to become a mainstream ML technique. Our next steps will be to perform

a more thorough review of the published work on this subject, along with the formalization of benchmark problems where new and old overfitting control techniques may be tested and compared to each other. The techniques described here are currently being used in hard multidimensional real world problems, already with some reported successes. Comparing these achievements with the results of other regression methods, evolutionary or not, is part of our future work. Understanding the dynamics of overfitting in GP and developing a simple technique to completely eliminate it from the search process is our ultimate goal.

**Acknowledgments.** This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds. The authors also acknowledge project PTDC/EIA-CCO/103363/2008 from FCT, Portugal.

## References

1. Y. Liu and T. Khoshgoftaar (2004). Reducing Overfitting in Genetic Programming Models for Software Quality Classification. In Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering, 56–65. IEEE Press
2. R. Poli, W.B. Langdon and N.F. McPhee (2008). A field guide to genetic programming, `http://lulu.com`, `http://www.gp-field-guide.org.uk`, (With contributions by J.R. Koza)
3. S. Luke, and L. Panait (2002). Lexicographic parsimony pressure. In Proceedings of GECCO 2002, 829–836. Morgan Kaufmann
4. S. Silva (2009). GPLAB - a genetic programming toolbox for MATLAB, version 3.0 (2009), `http://gplab.sourceforge.net`
5. T.M. Mitchell (1997). Machine Learning. McGraw-Hill
6. M. O'Neill, L. Vanneschi, S. Gustafson and W. Banzhaf (2010). Open Issues in Genetic Programming. Genetic Programming and Evolvable Machines 11: 339-363
7. J. Koza (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press
8. I. Kushchu (2002). An Evaluation of Evolutionary Generalisation in Genetic Programming. Artificial Intelligence Review 18: 3-14
9. S. Silva and E. Costa (2009). 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
10. L. Vanneschi and S. Silva (2009). Using Operator Equalisation for Prediction of Drug Toxicity with Genetic Programming. In Proceedings of EPIA 2009, 65-76. Springer
11. L.A. Becker and M. Seshadri (2003). Comprehensibility and Overfitting Avoidance in Genetic Programming for Technical Trading Rules. Technical report, Worcester Polytechnic Institute
12. S. Mahler, D. Robilliard and C. Fonlupt (2005). Tarpeian Bloat Control and Generalization Accuracy. In Proceedings of EuroGP 2005, 203-214. Springer
13. C. Gagné, M. Schoenauer, M. Parizeau and M. Tomassini (2006). Genetic Programming, Validation Sets, and Parsimony Pressure. In Proceedings of EuroGP 2006, 109-120. Springer

14. M.J. Cavaretta and K. Chellapilla (1999). Data Mining using Genetic Programming: The implications of parsimony on generalization error. In Proceedings of the 1999 IEEE Congress on Evolutionary Computation, 1330–1337. IEEE Press
15. B.-T. Zhang and H. Mhlenbein (1995). Balancing Accuracy and Parsimony in Genetic Programming. Evolutionary Computation 3(1): 17-38
16. E.J. Vladislavleva, G.F. Smits and D. den Hertog (2009). 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
17. L. Vanneschi, M. Castelli and S. Silva (2010). Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming. In Proceedings of GECCO 2010, 877–884. ACM Press
18. M. Castelli, L. Manzoni, S. Silva and L. Vanneschi. A Quantitative Study of Learning and Generalization in Genetic Programming. In Proceedings of EuroGP 2011, 25–36, Springer
19. L. Trujillo, S. Silva, P. Legrand and L. Vanneschi. An empirical study of functional complexity as an indicator of overfitting in Genetic Programming. In Proceedings of EuroGP 2011, 263–274, Springer
20. Q.U. Nguyen, T.H. Nguyen, X.H. Nguyen and M. ONeill (2010). Improving the Generalisation Ability of Genetic Programming with Semantic Similarity based Crossover. In Proceedings of EuroGP 2010, 184–195. Springer
21. L. Vanneschi and S. Gustafson (2009). Using Crossover Based Similarity Measure to Improve Genetic Programming Generalization Ability. In Proceedings of GECCO 2009, 1139–1146. ACM Press
22. L.E. Da Costa and J.-A. Landry (2006). Relaxed Genetic Programming. In Proceedings of GECCO 2006, 937-938. ACM Press
23. K.Y. Chan, C.K. Kwong and E. Chang (2011). Reducing Overfitting in Manufacturing Process Modeling using a Backward Elimination Based Genetic Programming. Applied Soft Computing 11(2): 1648–1656
24. N. Nikolaev, L.M. de Menezes and H. Iba (2002). Overfitting Avoidance in Genetic Programming of Polynomials. In Proceedings of the 2002 IEEE Congress on Evolutionary Computation, 1209–1214. IEEE Press
25. S.-H. Chen and T.-W. Kuo (2003). Overfitting or Poor Learning: A Critique of Current Financial Applications of GP. In Proceedings of EuroGP 2003, 34-46. Springer
26. N. Foreman and M. Evett (2005). Preventing overfitting in GP with canary functions. In Proceedings of GECCO 2005, 1779-1780. ACM Press
27. L. Vanneschi, D. Rochat and M. Tomassini (2007). Multi-optimization improves genetic programming generalization ability, In Proceedings of GECCO 2007, 1759. ACM Press
28. D. Robilliard and C. Fonlupt (2002). Backwarding: An Overfitting Control for Genetic Programming in a Remote Sensing Application. In Artificial Evolution, EA 2001 Selected Papers, 57–74. Springer
29. W. Banzhaf, F.D. Francone and P. Nordin (1996). The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming using Sparse Data Sets. In Proceedings of PPSN IV, 300–309. Springer
30. C. Gathercole and P. Ross (1994). Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. In Proceedings of PPSN III, 312-321. Springer