Reducing code bloat in Genetic Programming Based on Subtree Substituting Technique

Thi Huong Chu
Faculty of Information Technology
Le Quy Don Technical University
Hanoi, Vietnam
Email: huongktqs@gmail.com

Quang Uy Nguyen
Faculty of Information Technology
Le Quy Don Technical University
Hanoi, Vietnam
Email: quanguyhn@gmail.com

***Abstract*—Code bloat is a phenomenon in Genetic Programming (GP) that increases the size of individuals during the evolutionary process. Over the years, there has been a large number of research that attempted to address this problem. In this paper, we propose a new method to control code bloat and reduce the complexity of the solutions in GP. The proposed method is called *Substituting a subtree with an Approximate Terminal* (SAT-GP). The idea of SAT-GP is to select a portion of the largest individuals in each generation and then replace a random subtree in every individual in this portion by an approximate terminal of the similar semantics. SAT-GP is tested on twelve regression problems and its performance is compared to standard GP and the latest bloat control method (neat-GP). The experimental results are encouraging, SAT-GP achieved good performance on all tested problems regarding to the four popular performance metrics in GP research.**

I. Introduction

Bloat is a well-known phenomenon in Genetic Programming (GP) and is one of the major issues in GP that many researchers have attempted to address. It is a phenomenon that average size and depth of trees grow during the evolution without a corresponding increase in fitness [1], [2]. According to Purohit et. al. [2], there are three popular theories that explain GP code bloat phenomenon: the introns theory, the fitness causes bloat theory and the removal bias theory. The introns theory explains that GP code bloat is due to the existence pieces of code that does not affect the fitness. The fitness causes bloat theory is based on assuming that the probability to find a larger program is greater than the probability to find a smaller program. Therefore, in the process of the evolution, the probability to find a good solution naturally tend to the larger programs. The removal bias theory has another explanation: removing large subprogram is often more dangerous than small ones. Thus having a natural bias is beneficial for preserving larger programs during the evolution.

Besides that, most of the research has focused on proposing bloat control methods. The most popular techniques include setting the maximum depth (the largest size) for the trees as a threshold where the evolution must ensure that the depth (the size) of the tree does not exceed the threshold [3], [4], [5], [6], punishing the largest individuals in the fitness function [3], [7], [8], or adjusting population size distribution at each generation [9], [10], [11], [12].

In the paper, we propose a simple method for controlling GP code bloat and reduce the complexity of the solutions. The proposed method is called Substituting a subtree with an Approximate Terminal and shorted as SAT-GP. SAT-GP is tested on twelve regression problems and compared to standard GP and a recent bloat control method (neat-GP). Experimental results show that SAT-GP helps to significantly reduce code bloat compared to standard GP. Moreover, SAT-GP also achieved significantly better training error, testing error and simpler solutions compared to standard GP and neat-GP.

The remainder of this paper is organized as follows. In the next section, a brief review of methods for controlling code bloat in GP is presented. Section III presents our bloat control method. Section IV presents the experimental settings adopted in this paper. The experimental results presented and discussed in Section V. Finally, Section VI concludes the paper and highlights some future work.

II. Related Work

There have been a large number of techniques for controlling code bloat in GP. The early technique proposed by Koza [3] aims to limit the depth of the tree which sets the maximum depth (or maximum size) of a tree as threshold. Any offspring whose tree depth is higher than this threshold is rejected and replaced by one of its parents. Some variants of this technique included retrying crossover N times until it obtain a valid-depth offspring [4], assigning depth limit (size limit) with the depth (the size) of the best-so-far individual at the current generation [6]. To some extend, these techniques have been successful in controlling code bloat in GP. However, settings too small value of the threshold in these methods may hider the improvement of the fitness value.

Another technique is called parsimony pressure. The parsimony pressure technique aims at punishing the large individuals in the fitness function. Some studies such as [3], [7], [8] suggested restructuring the fitness function as a linear combination between the individual size and its fitness. Other methods such as Tarpeian technique [13] assigned a very bad fitness for individuals whose the size above the average size of the population. The parsimony pressure technique is also used in selection stage such as Lexicographic Parsimony Pressure method that modifies selection to prefer smaller trees [14]. Biased Multi-objective Parsimony Pressure method (BMOPP) [15] is also an extension of parsimony pressure that considered the size and the fitness as two goals of a multi-objective optimization problem.

Another trend in controlling code bloat in GP is based on genetic operators. In [16], Alfaro-Cid et al. proposed a new genetic operator for reducing GP code bloat named Prune and Plant (PP). In PP, a random point on a parent tree is selected and the branch at this point is replaced by a terminal. Moreover, the selected branch is also added to the population as a new tree. In this method, both offspring trees have smaller size than their parents leading to considerably reducing GP code bloat.

Recently, researchers were interested in the methods for reducing bloat by controlling the distribution of program sizes at each generation. The first technique is based on the concept of histogram called Operator Equalization (OpEq) [9]. The width of the bin determines the size of the program belonging to the bin and the height represents the number of programs belong to the bin. OpEq directs the population to a target distribution by accepting or rejecting each newly created individual into its corresponding bin. The experimental results showed that GP using OpEq is essentially bloat free. After that, some variants of OpEq are developed such as Dynamic Operator Equalisation, Mutation-based Dynamic Operator Equalisation and Flat Operator Equalisation (FlatOpEq) [10], [12], [17].

Most recently, Trujillo et al. [18] proposed a bloat control methods called neat-GP. Neat-GP is inspired by the insights of FlatOpEq method and built around the basic features of the NeuroEvolution of Augmenting Topologies (NEAT) algorithm, that explicitly shapes the program size distributions toward a uniform or at shape. One interesting thing in neat-GP is NEAT-Crossover operator. NEAT-Crossover operator first identifies the shared topological structure S_{ij} between parents. Then, it swaps of a single internal node within S_{ij}. In this technique, offspring maintain the topological structure of their parents, so there is no increase in size when doing crossover.

III. Methods

This section presents the details of our method (SAT-GP) for reducing code bloat in GP. The idea of SAT-GP is to randomly replace a subtree in some largest individuals by a terminal of approximate semantics. For semantic concept, we follow the previous research [19], [20] in defining semantics of a (sub)tree. Let $K = (k_1, k_2, ...k_N)$ be the fitness cases of the problem, then the semantics of a (sub)tree is defined as the vector of output values obtained by running that sub(tree) on all fitness cases.

Let X be a random terminal selected from the terminal set and let $T_1(X)$ be a random subtree selected in an individual, then the objective of SAT-GP is to replace $T_1(X)$ by subtree $T_2(X) = \theta.X$ so that the semantics of $T_1(X)$ and $T_2(X)$ are most similar. Here θ is a parameter that we need to find.

In other words, let $S = (a_1, a_2, ..., a_N)$ and $S_1 = (b_1, b_2, ..., b_N)$ be the semantics of terminal X and of subtree $T_1(X)$ so that $S_2 = (\theta a_1, \theta a_2, ..., \theta a_N)$ is the semantics of $T_2(X)$, then the objective is to find θ that minimizes squared Euclidean distance between vectors S_1 and S_2, $f(\theta) = \sum_{i=1}^{N}(a_i\theta - b_i)^2$, as in Equation 1.

$$\theta^* = \arg\underset{\theta}{\text{Min}}\sum_{i=1}^{N}(a_i\theta - b_i)^2 \tag{1}$$

Calculating the derivative of function $f(\theta)$ with respect to θ , we have:

$$f'(\theta) = 2\sum_{i=1}^{N}a_i^2\theta - 2\sum_{i=1}^{N}a_ib_i \tag{2}$$

Function $f(\theta)$ achieves minimal value when $f'(\theta) = 0$ and the value of θ at the optimal value of $f(\theta)$ is:

$$\theta^* = \frac{\sum_{i=1}^{N}a_ib_i}{\sum_{i=1}^{N}a_i^2} \tag{3}$$

After obtaining θ^*, we can grow a new tree $T_2(X) = \theta^*X$. This tree is called the approximate tree of $T_1(X)$ and is used to substitute (sub)tree $T_1(X)$.

Based on the concept of approximate trees, we propose a new bloat control method called *Substituting a subtree with an Approximate Terminal* (abbreviated as SAT-GP). The idea of SAT-GP is to prune a portion of the largest individuals in GP population by replacing a random selected subtree in each of these individuals by an approximate tree grown using the above manner. Figure 1 illustrates how to substitute a subtree in an individual by an approximate tree in details. The details description of SAT-GP is presented in Algorithm 1.

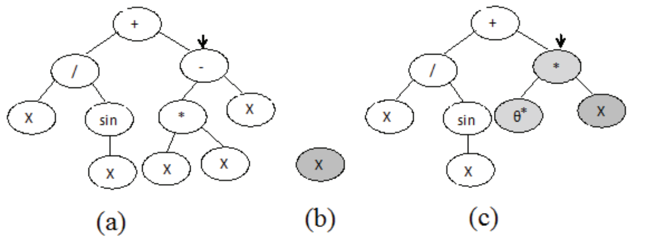

Fig. 1. (a) is the original tree with the selected subtree indicated by the down arrow, (b) is the selected terminal and (c) is the new tree obtained by substituting a branch of tree (a) with an approximate tree grown from the selected terminal (b).

```
Algorithm 1: Substituting a subtree with an Approximate Terminal
  Input: $Pop\_Size : P, Num\_Generations : G,$
         $Num\_Prune : k\%$
  $P_0 \longleftarrow initialize\_population()$
  Estimate fitness of all individuals in $P_0$
  for $i \leftarrow 1$ to $G$ do
      $P'_i \longleftarrow generate\_nextpop(P_{i-1})$
      $pool \longleftarrow$ get $k\%$ of the largest individuals of $P'_i$
      $P_i \longleftarrow P'_i - pool$
      foreach $p' \in pool$ do
          $X_j \longleftarrow$ Random select a terminal
          $p \longleftarrow Substitution(p', X_j)$
          $P_i \longleftarrow P_i \cup p$
      Estimate fitness of all individuals in $P_i$
```

In Algorithm 1, function $initialize\_population()$ is used to initialize a GP population. Function $generate\_nextpop(P_{i-1})$ applies genetic operators (tournament selection, crossover and mutation) on the population at generation $i-1$ (P_{i-1}) to generate a template population at generation i (P'_i). The next step in the algorithm is to select (and store in a pool) $k\%$ of the largest individuals of the template population (P'_i). After that, each individual in the pool is pruned by replacing a randomly selected subtree by a random terminal (Function $Substitution(p', X_j)$ in the algorithm selects a random subtree in tree p' and replace that subtree by with an approximate tree constructed from random terminal X_j). The new individual (p) is then used to form the full population at generation i (P_i). The population at generation i is then evaluated using the fitness function and the whole process is repeated until the last generation.

IV. Experimental Settings

In order to evaluate the effectiveness of SAT-GP, we tested this technique on twelve popular benchmarking problems in the GP literature [21]. Table I details description of the tested problems including its names, its abbreviation, number of features, number of training and testing samples.

TABLE I. Problems for testing SAT-GP.

Shorthanded	Name	Features	Training	Testing
F1	vladislavleva-2	1	100	221
F2	vladislavleva-4	5	500	500
F3	vladislavleva-6	2	30	93636
F4	vladislavleva-8	2	50	1089
F5	Korns-1	5	1000	1000
F6	Korns-2	5	1000	1000
F7	Korns-3	5	1000	1000
F8	Korns-4	5	1000	1000
F9	Korns-11	5	1000	1000
F10	Korns-12	5	1000	1000
F11	Korns-14	5	1000	1000
F12	Korns-15	5	1000	1000

The experimental GP parameters used in the paper are shown in Table II. These are the typical settings that are often used by GP researchers. The terminal set for each problem includes N variables corresponding to the number of features of that problem. The raw fitness is calculated as root mean squared error on all fitness cases. Therefore, smaller values are better. The elitism technique is also used for all tested systems, which means that the best individual of the current generation is always copied to the next generation. For each problem and each parameter setting, 30 runs were performed.

TABLE II. Evolutionary Parameter Values.

Parameter	Value
Population size	500
Generations	100
Selection	Tournament
Tournament size	3
Crossover, mutation probability	0.9; 0.1
Function set	$+, -, *, /, sin, cos$
Terminal set	$X_1, X_2, ..., X_n$
Initial Max depth	6
Max depth	17
Max depth of mutation tree	15
Raw fitness	root mean squared error on all fitness cases
Trials per treatment	30 independent runs for each value
Elitism	Copy the best individual to the next generation.

As described in Algorithm 1, $k\%$ of the largest individuals in the population are pruned at each geneartion. In this paper, we tested four values of k including 5%, 10%, 15% and 20%. Four configurations corresponding to four values of k are shorted as SAT-GP5, SAT-GP10, SAT-GP15, and SAT-GP20. Moreover SAT-GPs are shorted for all four configurations. It should be noted that using 0% for k would be equivalent to standard GP.

V. Results and Discussion

We compare the performance of SAT-GP with standard GP and the most recent bloat control method proposed by the Trujillo et al.(neat-GP) [18]. All these techniques were implemented in Java except neat-GP where we used the implementation in Python [1]. Wilcoxon signed rank test with the confidence level of 95% is used across all tables in this section to compare the performance of SAT-GP and neat-GP with standard GP in terms of statistics. If the test shows that SAT-GP and neat-GP is significantly better than standard GP, this result is marked + at the end. Conversely, if it is significantly worse compared to standard GP, this result is marked - at the end.

The first metric we analyze is the mean best fitness on the training data. Table III presents these values. The table indicates that SAT-GPs often achieve better training errors than standard GP and neat-GP. Especially, SAT-GP5 and SAT-GP10 achieved smaller training error than standard GP on 10 and 12 problems out of 12 tested problems, respectively. The training error of SAT-GP15 and SAT-GP20 is not as good as that of SAT-GP5 and SAT-GP10. However, two these configurations are still better than standard GP on some problems regarding to the training error. On the contrary, the training error of neat-GP is often much worse compared to standard GP and SAT-GPs on the tested problems.

TABLE III. The mean best fitness on all training data. Bold indicates the value of SAT-GPs and neat-GP are better (lower) than the value of standard GP. Underline indicates the best (lowest) value. The result is marked + if it significantly smaller than standard GP and marked - if it is significantly greater than standard GP.

Pro	GP	neat-GP	SAT-GP5	SAT-GP10	SAT-GP15	SAT-GP20
F1	0.006	0.011^{-}	$\underline{\mathbf{0.005}}$	**0.006**	0.008	0.007
F2	0.007	0.008^{-}	$\underline{\mathbf{0.006}}^{+}$	**0.007**	**0.007**	0.008^{-}
F3	0.121	0.193^{-}	**0.105**	**0.095**	$\underline{\mathbf{0.080}}^{+}$	**0.099**
F4	0.100	0.132^{-}	0.108	$\underline{\mathbf{0.100}}$	0.123^{-}	0.116^{-}
F5	0.164	0.312^{-}	**0.012**$^{+}$	$\underline{\mathbf{0.005}}^{+}$	**0.027**$^{+}$	**0.051**$^{+}$
F6	0.506	0.710	**0.255**	$\underline{\mathbf{0.063}}^{+}$	**0.339**	0.732
F7	1.134	1.928^{-}	**0.969**	$\underline{\mathbf{0.862}}$	1.217	1.777^{-}
F8	0.006	0.008^{-}	$\underline{\mathbf{0.0020}}^{+}$	**0.0021**$^{+}$	**0.0022**	**0.0024**
F9	0.257	0.272^{-}	**0.253**	$\underline{\mathbf{0.251}}$	**0.252**	**0.253**
F10	0.035	0.036^{-}	$\underline{\mathbf{0.0339}}$	**0.0341**$^{+}$	**0.0341**$^{+}$	**0.0342**$^{+}$
F11	9.90	16.48^{-}	10.56	**8.37**	13.53^{-}	26.10^{-}
F12	0.376	0.468^{-}	$\underline{\mathbf{0.332}}$	**0.366**	0.447^{-}	0.488^{-}

The statistical test using a Wilcoxon signed rank test with the confidence level of 95% also confirms that the SAT-GPs, especially SAT-GP5 and SAT-GP10, are often significantly better than standard GP on training error. In fact, SAT-GP5 and SAT-GP10 are significantly better than standard GP on 3 and 4 problems while standard GP is not significantly better than two these configurations on any problem. For SAT-GP15 and SAT-GP20, the result of Wincoxon test shows that their training error is roughly equal to the training error of standard GP. For neat-GP, the statistical test shows that, its training error is

[1]http://www.tree-lab.org/index.php/resources-2/downloads/open-source-tools

significantly worse than standard GP on most tested problem. Obviously, neat-GP is significantly worse than standard GP on 11 out of 12 problems [2].

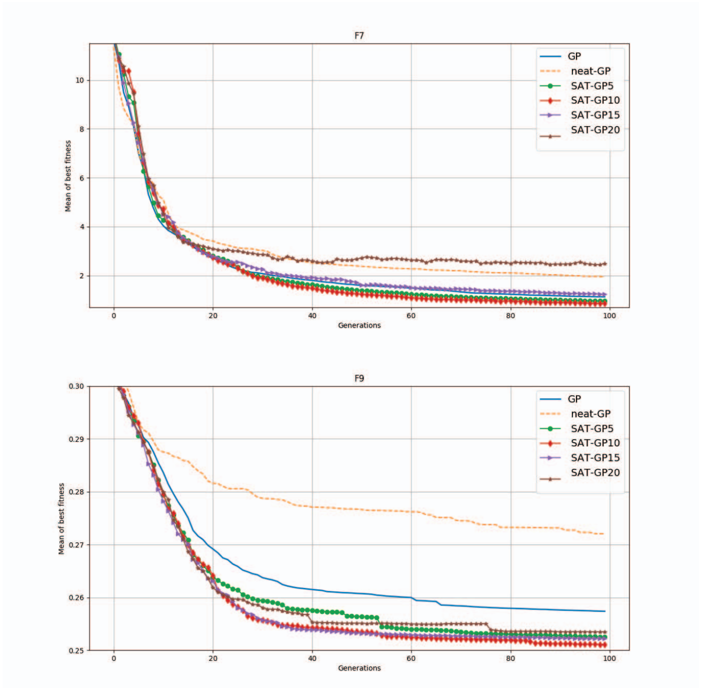

Fig. 2. Training error over the generations on two problems F7 and F9

Figure 2 shows the training error over the generations of the tested methods on two typical problems (F7 and F9) [3]. This figure shows that some configurations of SAT-GPs, particularly SAT-GP5 and SAT-GP10, achieved much better training error compared to standard GP. Actually, the training error of SAT-GP5 and SAT-GP10 is clearly smaller than that of standard GP considering from generation 20. On the other hand, the training error of neat-GP is usually much higher than standard GP started from generation 20.

The second metric used to analyse the performance of the tested methods is their ability to generalize beyond the training data. In machine learning [22], the generalization ability is perhaps the most desirable property of a learner. In each run, the best solution was selected and evaluated on the testing data (an unseen data set). The median of these values across 30 runs was calculated and the results are shown in Table IV.

The results in Table IV table are consistent with those in Table III, confirming the competitiveness of SAT-GPs to standard GP. The generalization of the SAT-GPs is mostly better than the standard GP on the unseen data. Particularly, SAT-GP5, SAT-GP10, SAT-GP15 and SAT-GP20 achieved better performance than standard GP on 11, 9, 10 and 10 problems out of 12 tested problems respectively. Interestingly, although SAT-GP15 and SAT-GP20 are not better than standard GP on training data, their performance on testing data is much more

TABLE IV. THE MEDIAN OF TESTING ERROR. BOLD INDICATES THE VALUE OF SAT-GPS AND NEAT-GP ARE BETTER (LOWER) THAN THE VALUE OF STANDARD GP. UNDERLINE INDICATES THE BEST (LOWEST) VALUE. THE RESULT IS MARKED + IF IT SIGNIFICANTLY SMALLER THAN STANDARD GP AND MARKED - IF IT IS SIGNIFICANTLY GREATER THAN STANDARD GP.

Pro	GP	neat-GP	SAT-GP5	SAT-GP10	SAT-GP15	SAT-GP20
F1	0.006	0.010^{-}	0.006	$\underline{\mathbf{0.005}}^{+}$	0.006	0.006
F2	0.008	0.009^{-}	$\underline{\mathbf{0.007}}^{+}$	**0.008**	**0.008**	0.008^{-}
F3	0.006	0.007	**0.004**	0.006	$\underline{\mathbf{0.002}}$	**0.004**
F4	0.056	**0.056**	**0.056**	**0.051**	0.056	$\underline{\mathbf{0.051}}$
F5	0.229	0.271^{-}	$\mathbf{0.003}^{+}$	$\underline{\mathbf{0.003}}^{+}$	$\mathbf{0.021}^{+}$	$\mathbf{0.050}^{+}$
F6	0.314	1.017	**0.286**	$\underline{\mathbf{0.085}}^{+}$	$\mathbf{0.117}^{+}$	**0.262**
F7	33.01	**24.66**	**29.33**	$\mathbf{19.67}^{+}$	$\mathbf{8.31}^{+}$	$\mathbf{6.55}^{+}$
F8	0.003	0.010^{-}	$\underline{\mathbf{0.002}}^{+}$	**0.003**	**0.003**	$\mathbf{0.003}^{+}$
F9	0.256	0.275	**0.255**	**0.255**	**0.255**	$\underline{\mathbf{0.254}}^{+}$
F10	0.0343	**0.0342**	**0.0342**	**0.0342**	$\mathbf{0.0342}^{+}$	$\mathbf{0.0341}^{+}$
F11	45.88	47.34	**45.49**	46.98	**44.48**	**44.42**
F12	2.189	2.192	**2.187**	2.194	**2.184**	$\underline{\mathbf{2.177}}^{+}$

convincing. Perhaps, the reason for the better testing error of SAT-GPs is due to their ability to obtain very simple solutions as shown in Table V.

For neat-GP, its testing error is better than standard GP on only threes problems. This results is consistent with the results in [18] in which the authors also reported that neat-GP is not better than standard GP on the testing data although this method reduce GP code bloat considerably. In terms statistical comparison, the table shows that SAT-GPs are significantly better than standard GP on 3, 4, 4 and 6 problems corresponding to four configurations SAT-GP5, SAT-GP10, SAT-GP15 and SAT-GP20 while neat-GP is not significantly better than standard GP on any problem. In fact, neat-GP is significantly worse than standard GP on four problems (F1, F2, F5 and F8).

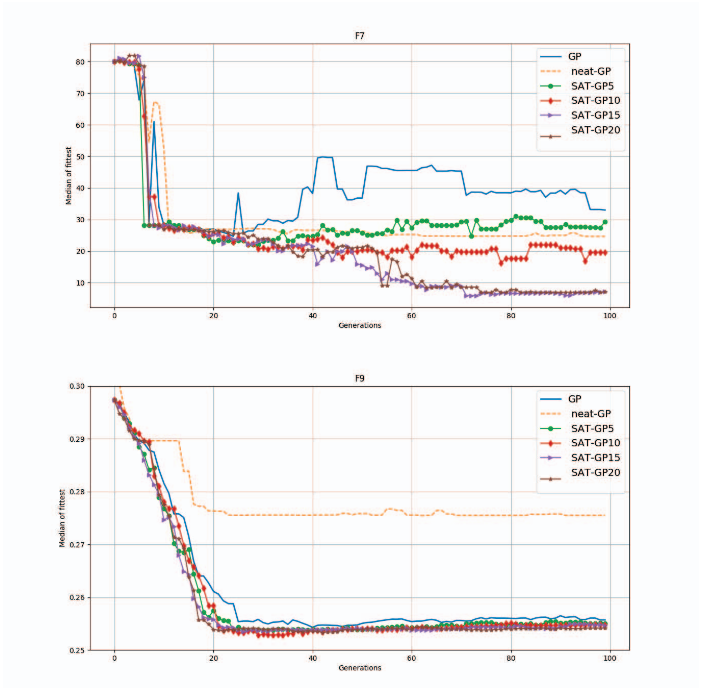

Fig. 3. Testing error over the generations on two problems F7 and F9

In Figure 3 we present the testing error of the tested methods on two problem F7 and F9 over the evolutionary process. The figure shows that the testing error of SAT-GPs, especially

[2]It is noted that the training error is not reported in Trujillo et al.'s paper [18].

[3]Due to space limitation, we only present the figures of two problems F7 and F9 in this paper. The figures of other problems have been uploaded to the link: https://github.com/chuthihuong/SAT-GP.

SAT-GP15 and SAT-GP20, is often much smaller than that of standard GP started from generation 20. Particularly, on problem F7, the testing error of SAT-GP15 and SAT-GP20 kept improving until the end of the evolution while standard GP, neat-GP and SAT-GP5 are overfitted (the testing error is increased) from generation 40.

The next metric for analyzing the effect of the proposed method is the complexity of the solutions found by each technique. This is the main objective of the proposed method (SAT-GP). In each GP run, we recorded the size of the selected solution. These values are then averaged over 30 runs and they are presented in Table V. It can be seen from this table, all tested bloat control methods, including SAT-GPs and neat-GP methods help to find the solutions of smaller size compared to standard GP. Among these methods, SAT-GP20 is the best configuration that always achieved the smallest solutions. It can be observed that, the size of the solutions found of SAT-GP20 is about only one over five to one over ten ($1/5$ to $1/10$) the size of the solutions found by standard GP. For example, on problem F5, the average size of the solutions of SAT-GP20 is 8.7 while that value of standard GP is 179.9. Perhaps, the ability to obtained simple solution is one of the reason explaining for the better testing error of SAT-GPs based on occam's razor principle [23].

TABLE V. THE AVERAGE OF SOLUTION'S SIZE. BOLD INDICATES THE VALUE OF SAT-GPS AND NEAT-GP ARE BETTER (LOWER) THAN THE VALUE OF STANDARD GP. UNDERLINE INDICATES THE BEST (LOWEST) VALUE. THE RESULT IS MARKED + IF IT SIGNIFICANTLY SMALLER THAN STANDARD GP AND MARKED - IF IT IS SIGNIFICANTLY GREATER THAN STANDARD GP.

Pro	GP	neat-GP	SAT-GP5	SAT-GP10	SAT-GP15	SAT-GP20
F1	163.5	**94.0**$^+$	**124.9**$^+$	**93.1**$^+$	**87.0**$^+$	$\underline{\mathbf{63.1}}^+$
F2	100.9	**29.0**$^+$	**73.5**$^+$	**50.5**$^+$	**38.8**$^+$	$\underline{\mathbf{17.0}}^+$
F3	152.3	$\underline{\mathbf{43.6}}^+$	**106.0**$^+$	**72.0**$^+$	**58.6**$^+$	**46.9**$^+$
F4	180.9	**48.4**$^+$	**85.8**$^+$	**66.7**$^+$	$\underline{\mathbf{43.3}}^+$	**63.0**$^+$
F5	179.9	**120.3**$^+$	**79.1**$^+$	**47.9**$^+$	**15.0**$^+$	$\underline{\mathbf{8.7}}^+$
F6	187.3	**75.7**$^+$	**111.2**$^+$	**72.9**$^+$	**41.5**$^+$	$\underline{\mathbf{20.0}}^+$
F7	162.5	**65.8**$^+$	**96.8**$^+$	**59.5**$^+$	**25.6**$^+$	$\underline{\mathbf{13.1}}^+$
F8	140.1	**41.9**$^+$	**95.8**$^+$	**71.1**$^+$	**47.1**$^+$	$\underline{\mathbf{24.5}}^+$
F9	216.9	**68.4**$^+$	**125.6**$^+$	**87.3**$^+$	**53.9**$^+$	$\underline{\mathbf{27.5}}^+$
F10	153.6	**52.9**$^+$	**101.2**$^+$	**70.5**$^+$	**45.0**$^+$	**20.9**$^+$
F11	161.0	**75.3**$^+$	**97.9**$^+$	**63.4**$^+$	**25.9**$^+$	$\underline{\mathbf{18.0}}^+$
F12	237.8	**76.6**$^+$	**122.3**$^+$	**78.5**$^+$	**36.0**$^+$	$\underline{\mathbf{14.1}}^+$

The average size of the population (the average size of the individuals in the population) over the generation on two problems F7 and F9 are showed in Figure 4. This figure shows that SAT-GP achieves its main objective in reducing GP code bloat. Obviously, the size of the population of SAT-GPs is much smaller than that of standard GP. Among all tested methods, the figure shows that SAT-GP15 and SAT-GP20 mostly does not impose any code bloat during the whole course of the evolution. In other words, the average size of the population two these configurations is mostly constant during the learning process. This not only helps to improve the testing error of two configurations but also to remarkably reduce their running time as shown in Table VI.

The last metric we examine is the average running time of all tested GP systems. The total time needed to complete a GP run is recorded and then these values are averaged over 30 runs. The results are showed in Table VI. It can be observed from this table that SAT-GPs runs faster than standard GP on most of the tested problems. This is not surprising, since the previous

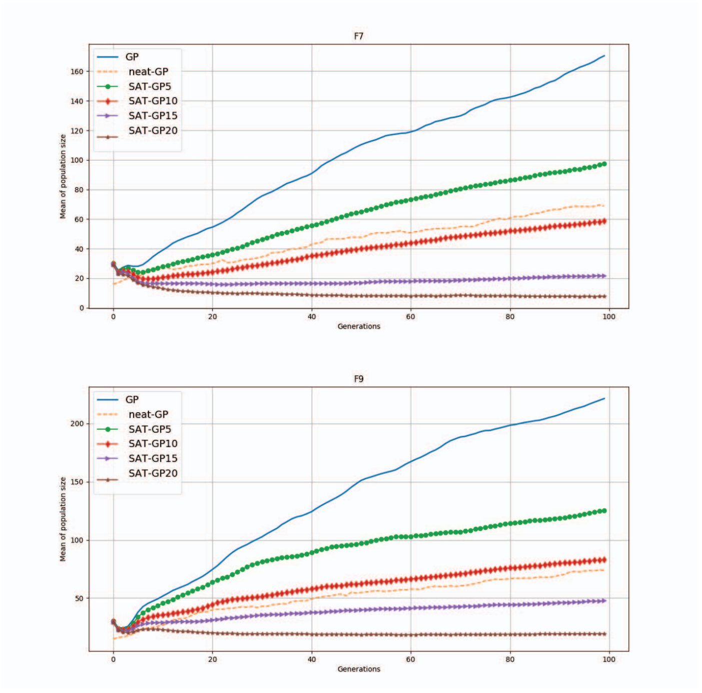

Fig. 4. The average size of population over the generations on F7 and F9

analysis has shown that SAT-GPs maintain a population of much smaller size than standard GP. For neat-GP, we can not directly compare its running time with other tested systems since this technique is implemented on a different platform (Python vs Java).

TABLE VI. AVERAGE RUNNING TIME IN SECONDS. BOLD INDICATES THE VALUE OF SAT-GPS AND NEAT-GP ARE BETTER (LOWER) THAN THE VALUE OF STANDARD GP. UNDERLINE INDICATES THE BEST (LOWEST) VALUE. THE RESULT IS MARKED + IF IT SIGNIFICANTLY SMALLER THAN STANDARD GP AND MARKED - IF IT IS SIGNIFICANTLY GREATER THAN STANDARD GP.

Pro	GP	neat-GP	SAT-GP5	SAT-GP10	SAT-GP15	SAT-GP20
F1	9.6	940.6$^-$	**7.5**$^+$	**5.9**$^+$	$\underline{\mathbf{5.5}}^+$	**5.7**$^+$
F2	23.1	240.2$^-$	**17.6**	**13.0**$^+$	**9.3**$^+$	$\underline{\mathbf{5.6}}^+$
F3	2.8	274.9$^-$	**2.1**$^+$	**1.3**$^+$	**1.4**$^+$	$\underline{\mathbf{1.2}}^+$
F4	4.6	313.1$^-$	**3.3**$^+$	**2.0**$^+$	$\underline{\mathbf{1.3}}^+$	**1.6**$^+$
F5	37.3	1411.3$^-$	**19.1**$^+$	**8.8**$^+$	$\underline{\mathbf{4.6}}^+$	**4.8**$^+$
F6	51.5	497.5$^-$	**30.7**$^+$	**17.8**$^+$	**9.9**$^+$	$\underline{\mathbf{6.2}}^+$
F7	53.2	490.5$^-$	**29.3**$^+$	**18.3**$^+$	**7.9**$^+$	$\underline{\mathbf{5.0}}^+$
F8	67.4	349.0$^-$	**45.8**$^+$	**32.3**$^+$	**21.6**$^+$	$\underline{\mathbf{12.8}}^+$
F9	81.2	552.1$^-$	**51.5**$^+$	**31.9**$^+$	**20.4**$^+$	$\underline{\mathbf{11.3}}^+$
F10	64.9	432.5$^-$	**42.2**$^+$	**29.4**$^+$	**19.6**$^+$	$\underline{\mathbf{9.0}}^+$
F11	66.3	746.1$^-$	**38.0**$^+$	**22.8**$^+$	**11.4**$^+$	$\underline{\mathbf{5.8}}^+$
F12	60.5	561.8$^-$	**39.8**$^+$	**22.1**$^+$	**12.5**$^+$	$\underline{\mathbf{6.8}}^+$

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a method for controlling bloat code and reducing the solution's complexity in GP. The proposed method is named *Substituting a subtree with an Approximate Terminal* (SAT-GP). SAT-GP replaces a subtree in the portion largest trees in GP population with a smaller tree of approximate semantics. Four configurations of SAT-GP were tested on twelve regression problems and analyzed using four different performance metrics. Results showed that the SAT-GPs achieved significantly better performance on most tested problems compared to standard GP and a recent bloat control method in GP (neat-GP).

There are a number of future research that arises from this paper. First, we want to examine methods for automatically self-adapting the percentage of the largest individuals in GP population for pruning. Second, a library of predefined subtrees can be generated from which we have more option to select approximate trees. Third is to use a pruning probability so that the large trees have a chance to survive. This may be useful for the problems of high complexity. Last but not least, we want to apply SAT-GP to a wider range of real-world applications to better understand its performance.

Acknowledgment

The work in this paper was funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED), under grant number 102.01-2014.09.

References

[1] P. A. Whigham and G. Dick, "Implicitly controlling bloat in genetic programming," *IEEE Transaction on Evolutionary Computation*, vol. 14, no. 2, pp. 173–190, APRIL 2010.

[2] A. Purohit, N. S. Choudhari, and A. Tiwari, "Code bloat problem in genetic programming," *International Journal of Scientific and Research Publications*, vol. 3, no. 4, p. 1612, 2013.

[3] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[4] P. Martin and R. Poli, "Crossover operators for a hardware implementation of gp using fpgas and handel-c," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2002, pp. 845–852.

[5] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evolutionary Computation*, vol. 14, no. 3, pp. 309–344, 2006.

[6] 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*, vol. 10, no. 2, pp. 141–179, 2009.

[7] M. J. Cavaretta and K. Chellapilla, "Data mining using genetic programming: the implications of parsimony on generalization error," in *Proceedings of the 1999 Congress on Evolutionary Computation*, vol. 2, 1999, p. 1337 Vol. 2.

[8] T. Belpaeme, "Evolution of visual feature detectors," in *University of Birmingham School of Computer Science technical*. Citeseer, 1999.

[9] S. Dignum and R. Poli, "Operator equalisation and bloat free gp," *Lecture Notes in Computer Science*, vol. 4971, pp. 110–121, 2008.

[10] S. Silva and S. Dignum, "Extending operator equalisation: Fitness based self adaptive length distribution for bloat free gp." in *EuroGP*. Springer, 2009, pp. 159–170.

[11] S. Silva and L. Vanneschi, "Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 1115–1122.

[12] 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*, vol. 13, no. 2, pp. 197–238, 2012.

[13] R. Poli, "A simple but theoretically-motivated method to control bloat in genetic programming," *Genetic programming*, pp. 43–76, 2003.

[14] S. Luke and L. Panait, "Lexicographic parsimony pressure," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2002, pp. 829–836.

[15] L. Panait and S. Luke, "Alternative bloat control methods," in *Genetic and Evolutionary Computation–GECCO 2004*. Springer, 2004, pp. 630–641.

[16] E. Alfaro-Cid, A. Esparcia-Alcázar, K. Sharman, F. F. de Vega, and J. Merelo, "Prune and plant: a new bloat control method for genetic programming," in *Hybrid Intelligent Systems 2008*. IEEE, 2008, pp. 31–35.

[17] S. Silva and L. Vanneschi, "The importance of being flat-studying the program length distributions of operator equalisation," *Genetic Programming Theory and Practice IX*, pp. 211–233, 2011.

[18] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva, "neat genetic programming: Controlling bloat naturally," *Information Sciences*, vol. 333, pp. 21–43, 2016.

[19] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galvan-Lopez, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, Jun. 2011.

[20] A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2012, pp. 21–31.

[21] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, and S. Luke, "Better GP benchmarks: community survey results and proposals," *Genetic Programming and Evolvable Machines*, vol. 14, no. 1, pp. 3–29, 2013.

[22] M. Castelli, L. Manzoni, S. Silva, and L. Vanneschi, "A comparison of the generalization ability of different genetic programming frameworks," in *IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–8.

[23] S. Needham and D. L. Dowe, "Message length as an effective ockham's razor in decision tree induction." in *AISTATS*, 2001.