SEMANTIC APPROXIMATION BASED OPERATOR FOR REDUCING CODE BLOAT IN GENETIC PROGRAMMING

Chu Thi Huong[1]

Abstract

In Genetic Programming, code bloat is a well-known problem that is the increase in the average program size without a corresponding improvement in fitness. In order to address this problem, we proposed a new operator called *Prune and Plant based on Approximate Terminal* (shortened as PP-AT). PP-AT aims at reducing GP code bloat. It replaces a random subtree in a parent by an approximate tree of semantics to obtain the first offspring. This subtree is also added to the population as the second offspring. PP-AT is tested on fifteen regression problems and compared to standard GP and three recent bloat control methods. The experimental results demonstrate that PP-AT outperforms standard GP and other bloat control methods under comparison.

Index terms

Genetic Programming, Semantic Approximation, Code Growth, Code Bloat

1. Introduction

Genetic programming (GP) is an evolutionary computation technique that automatically solves problems without requiring users to know or specify the form or structure of the solution in advance [1]. GP has been successfully applied to a wide variety of problems in many fields. However, GP has three clearly identified challenges, code bloat, huge search space and problem difficulty [2]. In these challenges, code bloat is one of the most important pragmatic limitations in the development of real-world GP solutions. In GP, code bloat is a well-known phenomenon in which the average size and depth of trees grow during the evolution without a corresponding increase in fitness [3]. There are three common theories that try to explain bloat including the replication accuracy theory, the introns theory and the remove bias theory [4], [5]. Theory replication accuracy theory argues that the success of an individual is to generate offspring being functionally similar to it. This is achieved by reproducing the important pieces of code of this individual during evolution to increase replication accuracy. The intron theory explains that GP code bloat is caused by the existing pieces of code that can be removed without changing the fitness value of the solution. The remove bias

[1]Faculty of Information Technology, Le Quy Don Technical University

theory has another argument that redundant codes tend to be lower subtrees of the tree, and applying genetic operators such as crossover to these subtrees does not modify the fitness. Moreover, the evolution will naturally favor the replacement of these small subtrees, and the replacement subtrees are often bigger leading consequently to bigger trees.

Code bloat negatively affects the GP performance: the evolutionary process is more time consuming, it is harder to interpret the solutions, and the larger solutions are prone to overfitting. There are a number of approaches that have been carried out to overcome this problem. Most common approaches consist of setting the size or depth limits [6], [7], punishing the largest individuals in the fitness function [8], [9] or adjusting the population size distribution at each generation [10], [11].

Recently, we proposed a technique to grow an approximate tree so that its semantics is the most semantically similar to the semantics of a given subtree [12]. In this paper, we apply this technique for Prune and Plant operator [13] to propose a new bloat control operator. The operator is called *Prune and Plant based on Approximate Terminal* and shortened as PP-AT. PP-AT selects a random subtree in a parent and replaces it by an approximate tree of semantics. Moreover, this subtree is also planted as a second offspring. The performance of PP-AT is evaluated via fifteen regression problems with comparison to standard GP, neat-GP [14], Prune and Plant [13] and SAT-GP [12]. The experimental results are encouraging, the new proposed operator achieved better performance than other tested systems. Especially, PP-AT improves the performance of Prune and Plant operator.

The remainder of this paper is organized as follows. We briefly review the related works in Section 2. After that, Section 3 presents a new proposed operator. The experimental settings are presented in Section 4. Next, the results are presented and discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

There are several approaches to control code bloat in GP. Setting size or depth limits to control the size of individuals is an earlier approach [6]. Any offspring with its size or depth above the limits is rejected and replaced by one of its parents. Later, methods use dynamic limits that have been proposed. The dynamic limits are assigned at each generation and are derived from the size of the best-so-far individual [7]. This approach has succeeded in controlling bloat code. However, if the limits are set at too small values, the improvement of fitness values may be prevented.

Parsimony pressure technique is another approach that aims at punishing the large individuals. The fitness function was suggested restructuring as a linear combination between the individual size and its fitness [8], [9]. Another way, the fitness function is assigned a very bad fitness for individuals whose size above the average size of the population [4]. Based on genetic operators, Alfaro-Cid et al. [13] proposed Prune and Plant operator that splits a tree into two subtrees to reduce GP code bloat. The operator

selects a random subtree in a parent and replaces this subtree by a terminal. The selected subtree is also added to the population as a new tree.

Another approach is to redistribute the population size distribution aiming to reduce code bloat including Operator Equalization, Dynamic Operator Equalisation, Mutation-based Dynamic Operator Equalisation and FlatOpEq [10], [11], [15]. These operators calculate the histogram of the individual size and control the population to a target distribution by accepting or rejecting each newly created individual into its corresponding bin. Inspired by FlatOpEq method, Trujillo et al. [14] proposed a bloat control method called neat-GP. The main genetic operator in neat-GP is Neat-crossover operator. The operator first identifies the shared topological structure S_{ij} between parents, and then swaps of a single internal node within S_{ij}. In this way, offspring maintain the topological structure of their parents. Thus, the size of the resultant tree does not increase when doing crossover.

Incorporating semantic information into GP is another recent approach to address the code bloat problem. Chu et al. [12] proposed a semantic-based method for reducing GP code bloat that is called SAT-GP. In each generation, SAT-GP selects a portion of the largest individuals and then replace a random subtree in every individual in this portion by an approximate tree. The approximate tree is grown from a terminal so that its semantics is the most similar to the semantics of this subtree. Then, SAT-GP is extended to SAS-GP [16]. In SAS-GP, the approximate tree is grown from a small tree taken from a predefined library instead of the terminal set.

3. Proposed method

In this section, we first present the technique of constructing an approximate tree derived from SAT-GP method [12], followed by Prune and Plant operator [13], and finally a proposed new operator, PP-AT, is presented.

The idea of SAT-GP [12] is to replace a random subtree in the several largest individuals by an approximate tree of similar semantics. The approximate tree is constructed from a random terminal. For the semantic concept, we use a popular defining semantics of a (sub)tree in the previous research [17], [18]. Let $\mathbb{K} = (k_1, k_2, ...k_N)$ be the fitness cases of the problem, then the semantics of a (sub)tree is the vector of output values obtained by running that sub(tree) on all fitness cases. Following this definition, the semantic distance may be calculated by Euclidean or Manhattan distance. In this research, we use Euclidean distance.

Let X be a random terminal selected from the terminal set and let T_1 be a random subtree selected in an individual. From X, T_2 is grown in the form of $T_2 = \theta * X$ (where θ be a parameter) so that the semantics of T_2 is the most semantically similar to the semantics of a given subtree T_1. 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 then $S_2 = (\theta a_1, \theta a_2, ..., \theta a_N)$ will be the semantics of T_2. The objective is to find θ that minimizes semantic distance between S_1 and S_2. This means that minimizes function $f(\theta) = \sum_{i=1}^{N}(a_i\theta - b_i)^2$. The

quadratic function $f(\theta)$ reaches the minimum value at the vertex, θ^{op} as in Equation 1. $T_2 = \theta^{op} * X$ is called the approximate tree of X.

$$\theta^{op} = \frac{\sum_{i=1}^{N} a_i b_i}{\sum_{i=1}^{N} a_i^2} \tag{1}$$

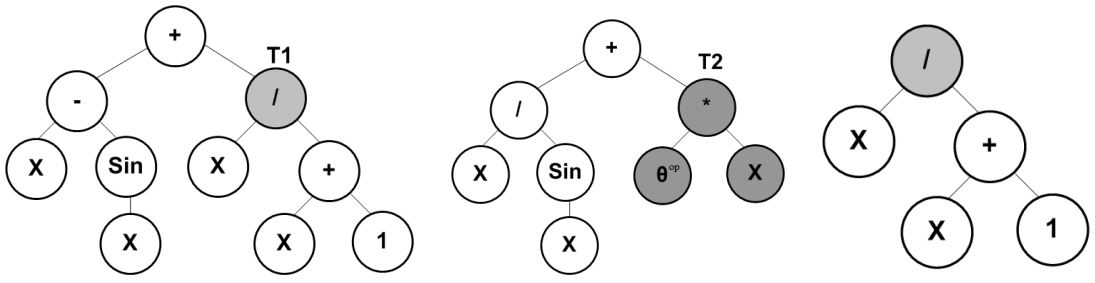
Prune and Plant [13] selects a random subtree in the parent and substitutes this subtree by a terminal. The subtree will be planted in the population as a new tree.

The new proposed operator is an extension of Prune and Plant that is called PP-AT. PP-AT selects a random subtree in a parent and replaces it by an approximate tree of semantics to obtain the first offspring. Additionally, this subtree is also added to the population as the second offspring. Algorithm 1 presents this operator in detail.

Algorithm 1: PP-AT

Input: A parent: P.
Output: Two offspring.
$T_1 \longleftarrow Select\_Child(P)$;
$X \longleftarrow Select\_Terminal()$;
$S \longleftarrow Caculate\_Semantics(X)$;
$S_1 \longleftarrow Caculate\_Semantics(T_1)$;
$\theta^{op} \longleftarrow Caculate\_Theta(S, S_1)$;
$T_2 \longleftarrow \theta^{op} * X$
return T_1, T_2;

In Algorithm 1, functions $Select\_Child(P)$ and $Select\_Terminal()$ select randomly a subtree from the parent P and a terminal from the terminal set, respectively. Function $Caculate\_Semantics(X)$ calculates the semantics of subtree X. Next, function $Caculate\_Theta(S, S_1)$ estimates the optimal parameter theta, θ^{op}, from two semantics S, S_1 using Equation 1.



(a) Parent *(b) Child 1* *(c) Child 2*

Fig. 1. An example of PP-AT.

Figure 1 demonstrates an example of PP-AT. In this figure, PP-AT selects a random subtree T_1 and then replaces it with an approximate tree T_2. T_2 is grown so that the

semantics of T_2 and T_1 are the most semantically similar. Moreover, T_1 is grown in the population as a new another child. Similar to Prune and Plant, PP-AT is an unusual operator since it creates two offspring from a single parent. We follow the implementation of Prune and Plant [5]: when an offspring is generated by crossover or reproduction, PP-AT occurs with a certain probability.

4. Experimental Settings

We compared PP-AT with standard GP (referred to as GP) and three recent bloat control methods, including neat-GP [14], Prune and Plant (PP) [13] and SAT-GP [12]. The probability of PP is set to 0.5, and $k\%$ the largest individuals of SAT-GP is set 10%[1]. PP-AT is tested with three probabilities as 0.1, 0.2 and 0.5, and three configurations corresponding to these values are shortened as PP-AT0.1, PP-AT0.2 and PP-AT0.5. We tested these methods on fifteen regression problems including seven GP benchmark problems recommended in the literature [19] and eight problems taken from UCI machine learning dataset [20]. The detailed descriptions of the tested problems including its name, its abbreviation, number of features, number of training and testing samples are shown in Table 1.

Table 1. Problems for testing PP-AT

Shorthanded	Name	Features	Training	Testing
F1	Korns-1	5	1000	1000
F2	Korns-2	5	1000	1000
F3	Korns-4	5	1000	1000
F4	Korns-11	5	1000	1000
F5	Korns-12	5	1000	1000
F6	Korns-14	5	1000	1000
F7	Korns-15	5	1000	1000
F8	airfoil_self_noise	5	800	703
F9	ccpp	4	1000	1000
F10	wpbc	31	100	98
F11	Protein_Tertiary_Structure	9	1000	1000
F12	slump_test_Compressive	7	50	53
F13	slump_test_FLOW	7	50	53
F14	slump_test_SLUMP	7	50	53
F15	concrete	8	500	530

The experimental GP parameters are shown in Table 2. These are the typical settings often used by GP researchers. The raw fitness is the root mean squared error on all fitness cases. Therefore, smaller values are better. For each problem and each parameter setting, 30 runs were performed.

For statistical analysis, Kruskal-Wallis test with a confident level of 95% is used on the results in all result tables. If the result of Kruskal-Wallis test shows that at least one method is significantly different from the others, a post hoc analysis with Dunn's

[1]This value is selected for the best performance of PP and SAT-GP.

Test is conducted. p-values adjusted with the Benjamini-Hochberg method. In all result tables, if the test shows that the tested system is significantly better than GP, this result is marked + at the end [2]. Conversely, the result is marked - at the end. Furthermore, if the result of the tested system is better than that of GP, it is printed bold face. In addition, if the result is the best (the lowest), it is printed underline.

Table 2. Evolutionary Parameter Values

Parameter	Value
Population size	1024
Generations	200
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.

5. Results and Discussion

This section analyses the GP performance of the proposed method using four popular metrics in GP research that are training error, testing error, solution size and running time. Performance on the training data is a popular metric to provide some useful

Table 3. The mean best fitness on all training data

Pro	GP	neat-GP	PP	SAT-GP	PP-AT0.1	PP-AT0.2	PP-++A+T0.5
F1	0.106	0.232$^-$	0.143	**0.000**$^+$	**0.001**$^+$	**0.001**$^+$	**0.001**$^+$
F2	0.068	0.648$^-$	0.105$^-$	**0.006**$^+$	0.074	**0.055**$^+$	**0.048**
F3	0.003	0.006$^-$	0.003	**0.000**$^+$	**0.001**$^+$	**0.001**$^+$	**0.002**
F4	0.244	0.255$^-$	0.247$^-$	**0.244**$^+$	0.247$^-$	0.245	0.245$^-$
F5	0.034	0.034$^-$	**0.034**$^+$	**0.033**	**0.033**	**0.033**	**0.034**$^+$
F6	4.024	11.951$^-$	5.493$^-$	**3.866**	**3.839**	4.617	5.367$^-$
F7	0.219	0.466$^-$	0.341$^-$	**0.206**	**0.202**	**0.191**	0.275
F8	0.229	0.686$^-$	0.380	**0.134**$^+$	**0.136**$^+$	**0.120**$^+$	**0.133**$^+$
F9	0.252	0.430$^-$	0.315$^-$	**0.159**$^+$	**0.152**$^+$	**0.150**$^+$	**0.160**$^+$
F10	2.722	3.070$^-$	3.278$^-$	3.163$^-$	**2.694**	2.777	3.136$^-$
F11	0.164	0.170$^-$	0.169$^-$	0.166	**0.164**	**0.164**	0.169$^-$
F12	0.452	0.704$^-$	0.572$^-$	**0.392**	**0.433**	**0.415**	0.545$^-$
F13	1.188	1.561$^-$	1.548$^-$	1.403$^-$	1.222	1.212	1.514$^-$
F14	0.643	0.840$^-$	0.877$^-$	0.746$^-$	**0.632**	0.667	0.831$^-$
F15	0.410	0.548$^-$	0.483$^-$	**0.401**	**0.365**$^+$	**0.358**$^+$	0.440

[2]The p_values of the Kruskal-Wallis test with the post hoc analysis are presented in the supplement 1 of the paper at https://github.com/chuthihuong/PP-AT.

awareness into the learning process. Thus, it is first analysed in this section. The mean of the best fitness values across 30 runs is presented in Table 3. The table shows that SAT-GP and PP-AT probably performed better than GP. The training error of SAT-GP, PP-AT0.1 and PP-AT0.2 is smaller than that of GP on 11, 12, and 10 problems out of 15 tested problems, respectively. PP-AT0.5 is roughly equal to GP. Conversely, neat-GP and PP are significantly worse than GP on all tested problems.

The semantic distance between parents and their children of GP, SAT-GP and PP-AT [3] is also measured. These values aim to analyse the ability of an algorithm to discover different areas in the search space. The semantic distance between a pair of individuals (a parent and its offspring) on four typical problems, F1, F6, F8 and F15, over the evolutionary process is presented in Figure 2 [4].

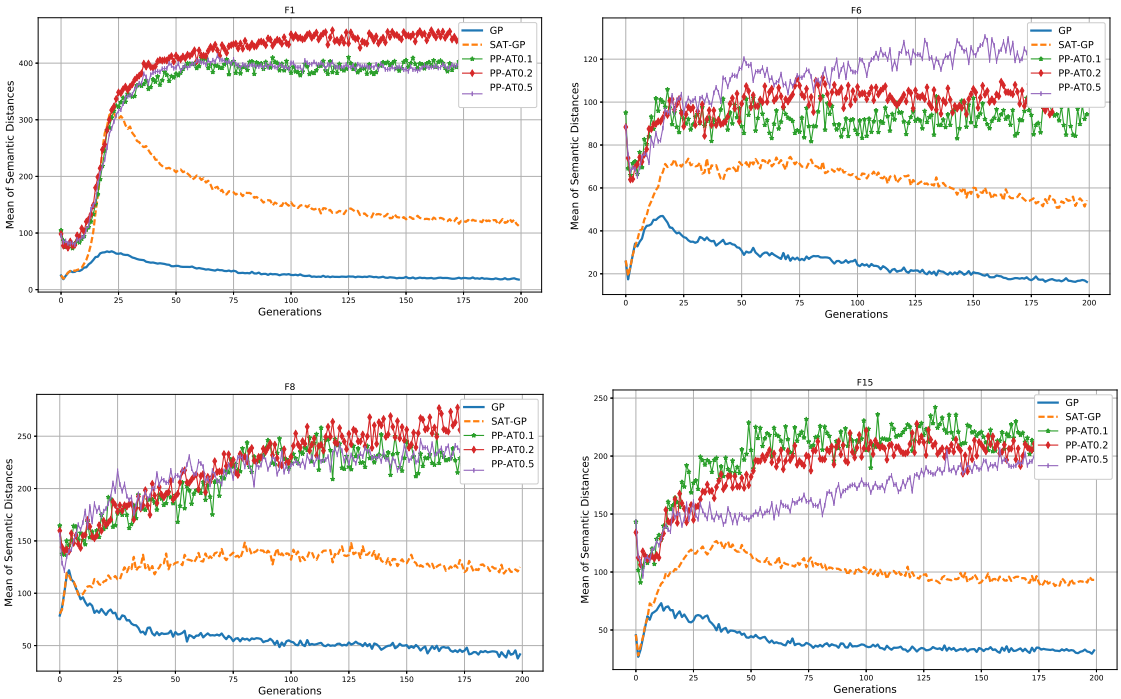


Fig. 2. Semantic distances over the generations

This figure shows that both SAT-GP and PP-AT often maintained higher semantic diversity compared to GP. Moreover, PP-AT often maintained better diversity than that of SAT-GP during the evolutionary process. These results demonstrate that PP-AT has enhanced the semantic diversity of GP population.

The second metric, used to analyse the performance of the tested methods, is their ability to generalise beyond the training data. In each run, the best solution was selected

[3]We focus on analysing SAT-GP and PP-AT since these are the bloat control methods based on semantics.

[4]The results on the other problems are shown in the supplement 2 of the paper at https://github.com/chuthihuong/PP-AT.

and evaluated on the testing data (an unseen data set). The mean of these values across 30 runs was calculated, and the results are shown in Table 4.

Table 4. The mean of testing error

Pro	GP	neat-GP	PP	SAT-GP	PP-AT0.1	PP-AT0.2	PP-AT0.5
F1	0.134	0.239	0.151	**0.000**$^{+}$	**0.001**$^{+}$	**0.001**$^{+}$	**0.001**$^{+}$
F2	1.266	**0.740**	**0.532**	**0.021**$^{+}$	**0.483**$^{+}$	**0.468**$^{+}$	**0.182**$^{+}$
F3	0.003	0.006^{-}	0.004	**0.000**$^{+}$	**0.002**	**0.001**$^{+}$	**0.002**
F4	0.263	**0.262**$^{+}$	**0.257**$^{+}$	**0.259**	**0.261**	**0.261**	**0.256**$^{+}$
F5	0.035	**0.034**$^{+}$	**0.034**	**0.034**	0.037	**0.034**	**0.034**
F6	73.279	74.857	**53.436**	**53.427**	74.084	**64.118**	**45.451**
F7	2.400	2.402^{-}	3.384	2.976	**2.294**	2.965^{-}	**2.286**
F8	17.510	**1.670**	**1.635**	**0.249**$^{+}$	**0.597**$^{+}$	**0.424**$^{+}$	**0.399**$^{+}$
F9	0.689	**0.552**	**0.371**	**0.192**$^{+}$	**0.347**$^{+}$	**0.372**$^{+}$	**0.281**
F10	42.192	**12.854**	**6.138**	**4.078**$^{+}$	**6.924**$^{+}$	**4.975**$^{+}$	**3.973**$^{+}$
F11	0.276	**0.176**$^{+}$	**0.172**	**0.169**$^{+}$	**0.173**$^{+}$	**0.170**$^{+}$	**0.199**
F12	25.824	**12.853**	5253.643	**2.291**$^{+}$	567.005^{-}	**5.593**$^{+}$	**25.088**$^{+}$
F13	4616.189	4.01E+05	1.16E+07	1.12E+04^{-}	**516.014**	8157.943	**17.992**$^{+}$
F14	52.203	5142.130	5124.724	**9.206**$^{+}$	**30.753**	**45.744**$^{+}$	4139.477^{-}
F15	2.500	**0.869**	**0.881**	**0.578**$^{+}$	**2.438**$^{+}$	**0.577**$^{+}$	**0.573**$^{+}$

This table highlights that SAT-GP and PP-AT consistently outperformed GP on the unseen data. In fact, the testing error of SAT-GP, PP-AT0.1, PP-AT0.2 and PP-AT0.5 is better than that of GP on 13, 12, 13 and 14 problems, respectively. For neat-GP and PP, they are slightly better than GP. The testing error of both neat-GP and PP is better than that of GP on 9 problems.

In terms of statistical comparison, the results of Kruskal-Wallis test again confirm the good generalization of both SAT-GP and PP-AT. The testing error of SAT-GP is significantly better than that of GP on 10 problems. PP-AT0.1, PP-AT0.2 and PP-AT0.5 are significantly better than that of GP on 7, 10 and 8 problems, respectively. Conversely, GP is only significantly better than SAT-GP and PP-AT on one problem. For PP, it is marginally better than GP. PP is significantly better than GP on only one problem. Similarly, neat-GP is significantly better than GP on 3 problems, but it also significantly worse than GP on 2 problems.

To compare among the bloat control methods, the results of the post hoc analysis of the Kruskal-Wallis test on testing error are summarised in Table 5. In this table, if a

Table 5. Summary of the post hoc analysis of the Kruskal-Wallis test on testing error. Each cell presents the number of problems that the method in a column is significantly better than the method in a row

	neat-GP	PP	SAT-GP	PP-AT0.1	PP-AT0.2	PP-AT0.5
neat-GP		1	11	8	12	10
PP	0		9	6	7	10
SAT-GP	1	0		0	1	0
PP-AT0.1	3	3	3		1	0
PP-AT0.2	0	2	1	0		3
PP-AP0.5	2	0	2	2	0	

method in the columns is significantly better than a method in the rows on k problems, k is presented in the corresponding cell. For example, SAT-GP is significantly better than neat-GP on 11 problems, 11 is presented in the cell at column "SAT-GP" and row "neat-GP".

The results of the post hoc analysis again confirm the good generalization ability of PP-AT. Table 5 shows that PP-AT is often significantly better than neat-GP and PP. For example, PP-AT0.2 is significantly better than neat-GP and PP on 12 and 7 problems, and the vice versa is not true on any problem with neat-GP and 2 problems with PP. Comparing with SAT-GP, PP-AT is roughly equal to SAT-GP.

One of the main reasons for performing bloat control is to help the algorithm find good and small solutions. We thus conducted an analysis of the size of solutions. To do this, we recorded the size of the selected solution (the number of nodes of this solution) in each run. These values are then averaged over 30 runs and presented in Table 6.

Table 6. The average of solution size

Pro	GP	neat-GP	PP	SAT-GP	PP-AT0.1	PP-AT0.2	PP-AT0.5
F1	348.9	**176.5**$^+$	**173.6**$^+$	**64.2**$^+$	**160.1**$^+$	**137.7**$^+$	**86.3**$^+$
F2	294.4	**92.8**$^+$	**142.9**$^+$	**82.7**$^+$	**248.9**	**218.4**	**148.9**$^+$
F3	196.7	**58.6**$^+$	**139.7**$^+$	**86.1**$^+$	**195.2**	**180.5**	**116.4**$^+$
F4	379.6	**88.2**$^+$	**202.3**$^+$	**112.0**$^+$	**342.6**	**329.3**	**184.9**$^+$
F5	259.9	**83.6**$^+$	**137.3**$^+$	**89.8**$^+$	**255.6**	**240.5**	**170.9**$^+$
F6	252.8	**91.3**$^+$	**129.2**$^+$	**96.4**$^+$	273.8	**232.2**	**131.8**$^+$
F7	315.7	**98.3**$^+$	**165.3**$^+$	**94.6**$^+$	**270.6**	**221.8**$^+$	**131.9**$^+$
F8	288.1	**173.6**$^+$	**158.1**$^+$	**83.6**$^+$	**280.7**	**219.8**$^+$	**135.1**$^+$
F9	287.3	**154.7**$^+$	**130.9**$^+$	**95.7**$^+$	**198.8**$^+$	**180.6**$^+$	**116.4**$^+$
F10	326.2	**210.5**$^+$	**171.4**$^+$	**37.0**$^+$	**277.0**	**250.6**$^+$	**109.1**$^+$
F11	271.9	**144.4**$^+$	**122.5**$^+$	**83.1**$^+$	**216.2**$^+$	**209.7**$^+$	**123.7**$^+$
F12	280.8	**138.9**$^+$	**155.2**$^+$	**94.1**$^+$	**220.2**$^+$	**222.6**$^+$	**105.7**$^+$
F13	336.1	**143.1**$^+$	**154.4**$^+$	**93.4**$^+$	**284.2**	**286.9**	**135.1**$^+$
F14	320.6	**156.2**$^+$	**139.5**$^+$	**102.6**$^+$	**295.6**	**276.4**	**129.8**$^+$
F15	265.5	**149.5**$^+$	**152.8**$^+$	**72.9**$^+$	**200.9**$^+$	**196.6**$^+$	**96.8**$^+$

Table 6 shows that all control bloat methods help to find simple solutions. The size of solutions found by them is significantly smaller than that of GP on most tested problems. Apparently, the size of solutions created PP-AT reduces when the probability of this operator increases, and PP-AT achieved its objective to reduce code bloat.

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 these values are then averaged over 30 runs. The results are shown in Table 7. It can be observed from this table that the average running time of PP, SAT-GP and PP-AT are smaller than that of GP. Comparing between PP and PP-AP0.5, although both of them use the same probability as 0.5, PP-AP0.5 often run fatter than PP. For neat-GP, since we used its implementation in Python [5] while others in Java, so they cannot directly compare. However, it also

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

requires a much longer time to run compared to others.

Table 7. The average of running time in seconds

Pro	GP	neat-GP	PP	SAT-GP	PP-AT0.1	PP-AT0.2	PP-AT0.5
F1	815.6	25 772.6$^-$	**224.2**$^+$	**70.3**$^+$	**162.8**$^+$	**144.7**$^+$	**102.1**$^+$
F2	1094.0	6842.0$^-$	**318.0**$^+$	**110.6**$^+$	**287.4**$^+$	**217.5**$^+$	**141.8**$^+$
F3	1097.7	3140.9	**482.3**$^+$	**217.1**$^+$	**362.2**$^+$	**288.6**$^+$	**189.2**$^+$
F4	1617.3	6672.2$^-$	**685.3**$^+$	**217.3**$^+$	**524.1**$^+$	**517.1**$^+$	**263.7**$^+$
F5	1247.4	3772.2	**416.7**$^+$	**188.7**$^+$	**416.7**$^+$	**438.1**$^+$	**243.6**$^+$
F6	1321.9	4388.9$^-$	**391.9**$^+$	**173.2**$^+$	**416.0**$^+$	**440.1**$^+$	**248.9**$^+$
F7	1237.5	4617.1	**367.5**$^+$	**141.9**$^+$	**343.8**$^+$	**332.1**$^+$	**174.5**$^+$
F8	1040.6	14 313.6$^-$	**247.0**$^+$	**54.9**$^+$	**183.2**$^+$	**168.6**$^+$	**98.3**$^+$
F9	706.7	7933.8$^-$	**252.1**$^+$	**76.1**$^+$	**186.7**$^+$	**200.9**$^+$	**94.8**$^+$
F10	366.9	12 453.2$^-$	**29.4**$^+$	**8.8**$^+$	**55.2**$^+$	**45.2**$^+$	**17.2**$^+$
F11	790.2	5798.4$^-$	**254.7**$^+$	**93.2**$^+$	**267.2**$^+$	**271.5**$^+$	**122.8**$^+$
F12	280.4	6526.6$^-$	**16.8**$^+$	**9.4**$^+$	**28.6**$^+$	**20.0**$^+$	**8.3**$^+$
F13	316.9	6246.6$^-$	**17.1**$^+$	**10.8**$^+$	**31.0**$^+$	**25.4**$^+$	**10.0**$^+$
F14	306.1	6747.4$^-$	**19.6**$^+$	**11.5**$^+$	**33.9**$^+$	**24.7**$^+$	**9.3**$^+$
F15	508.7	6705.9$^-$	**129.4**$^+$	**36.3**$^+$	**102.4**$^+$	**102.2**$^+$	**46.9**$^+$

6. Conclusions

In this paper, we used the technique of growing an approximate tree in the previous research [12] to improve Prune and Plant [13]. The new proposed operator, PP-AT, selects a random subtree in a parent and replaces this subtree by an approximate tree. The subtree is also added to the population as a second offspring. Experimental results showed that PP-AT boosted the performance of Prune and Plant, reduced GP code bloat and achieved significantly better performance compared to GP and other tested systems.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.05-2019.05

References

[1] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[2] M. A. Haeri, M. M. Ebadzadeh, and G. Folino, "Statistical genetic programming for symbolic regression," *Applied Soft Computing*, vol. 60, pp. 447–469, 2017.

[3] 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.

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

[5] E. Alfaro-Cid, J. J. Merelo, F. F. de Vega, A. I. Esparcia-Alcázar, and K. Sharman, "Bloat control operators and diversity in genetic programming: A comparative study," *Evolutionary Computation*, vol. 18, no. 2, pp. 305–332, 2010.

[6] 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.
[7] 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.
[8] T. Belpaeme, "Evolution of visual feature detectors," in *University of Birmingham School of Computer Science technical*. Citeseer, 1999.
[9] 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.
[10] S. Dignum and R. Poli, "Operator equalisation and bloat free gp," *Lecture Notes in Computer Science*, vol. 4971, pp. 110–121, 2008.
[11] 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.
[12] T. H. Chu and Q. U. Nguyen, "Reducing code bloat in genetic programming based on subtree substituting technique," in *IES2017*. IEEE, 2017, pp. 25–30.
[13] 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.
[14] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva, "neat genetic programming: Controlling bloat naturally," *Information Sciences*, vol. 333, pp. 21–43, 2016.
[15] M.-A. Gardner, C. Gagné, and M. Parizeau, "Controlling code growth by dynamically shaping the genotype size distribution," *Genetic Programming and Evolvable Machines*, vol. 16, no. 4, pp. 455–498, 2015.
[16] T. H. Chu, Q. U. Nguyen, and V. L. Cao, "Semantics based substituting technique for reducing code bloat in genetic programming," in *Proceedings of the Ninth International Symposium on Information and Communication Technology*. ACM, 2018, pp. 77–83.
[17] 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.
[18] T. H. Chu, Q. U. Nguyen, and M. O'Neill, "Semantic tournament selection for genetic programming based on statistical analysis of error vectors," *Information Sciences*, vol. 436, pp. 352–366, 2018.
[19] 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.
[20] K. Bache and M. Lichman, "UCI machine learning repository," 2013, http://archive.ics.uci.edu/ml.

Manuscript received 12-5-2020; Accepted 25-8-2020.

■

Chu Thi Huong received her B. Eng. degree in Applied Mathematics and Informatics from Hanoi University of Science and Technology and MSc Degree in Computer Science from Le Quy Don Technical University. She received the PhD degree in Mathematical Foundations for Informatics from Le Quy Don Technical University, in 2020. Since 2002, she has been teaching at Faculty of Information Technology, Le Quy Don Technical University. Her research interest are in the domain of Evolutionary Algorithms, Genetic Programming and Machine Learning.

TOÁN TỬ DỰA TRÊN XẤP XỈ NGỮ NGHĨA CHO VIỆC GIẢM PHÌNH MÃ TRONG LẬP TRÌNH DI TRUYỀN

Tóm tắt

Trong lập trình di truyền (GP), phình mã là hiện tượng phổ biến được đặc trưng bởi sự gia tăng kích thước chương trình mà không có sự cải thiện tương ứng về độ thích nghi. Để giải quyết bài toán này, chúng tôi đề xuất một toán tử điều khiển phình mã mới được gọi là *Prune and Plant based on Approximate Terminal* (ký hiệu là PP-AT). PP-AT nhằm mục tiêu giảm hiện tượng phình mã trong GP. PP-AT thay thế một cây con ngẫu nhiên trên cha mẹ bằng một cây xấp xỉ ngữ nghĩa để thu được con thứ nhất. Đồng thời, cây con này cũng được đưa vào quần thể như là con thứ hai. PP-AT được thử nghiệm trên mười lăm bài toán hồi quy và được so sánh với GP chuẩn và ba phương pháp điều khiển phình mã được đề xuất gần đây. Kết quả thực nghiệm chứng minh PP-AT vượt trội hơn GP chuẩn và các phương pháp điều khiển phình mã khác được so sánh.