A Double Lexicase Selection Operator for Bloat Control in Evolutionary Feature Construction for Regression

Hengzhe Zhang
hengzhe.zhang@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Qi Chen
qi.chen@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Bing Xue
bing.xue@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Wolfgang Banzhaf
banzhafw@msu.edu
Michigan State University
East Lansing, MI, USA

Mengjie Zhang
mengjie.zhang@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

ABSTRACT

Evolutionary feature construction is an important technique in the machine learning domain for enhancing learning performance. However, traditional genetic programming-based feature construction methods often suffer from bloat, which means the sizes of constructed features increase excessively without improved performance. To address this issue, this paper proposes a double-stage lexicase selection operator to control bloat while not damaging search effectiveness. This new operator contains a two-stage selection process, where the first stage selects individuals based on fitness values and the second stage selects individuals based on tree sizes. Therefore, the proposed operator can control bloat meanwhile leveraging the advantage of the lexicase selection operator. Experimental results on 98 regression datasets show that compared to the traditional bloat control method of having a depth limit, the proposed selection operator not only significantly reduces the sizes of constructed features on all datasets but also keeps a similar level of predictive performance. A comparative experiment with seven bloat control methods shows that the double lexicase selection operator achieves the best trade-off between the model performance and the model size.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**.

KEYWORDS

Evolutionary feature construction, genetic programming, bloat control

ACM Reference Format:
Hengzhe Zhang, Qi Chen, Bing Xue, Wolfgang Banzhaf, and Mengjie Zhang. 2023. A Double Lexicase Selection Operator for Bloat Control in Evolutionary Feature Construction for Regression. In *GECCO '23: The Genetic and Evolutionary Computation Conference, July 15-19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583131.3590365

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07…$15.00
https://doi.org/10.1145/3583131.3590365

1 INTRODUCTION

Feature construction is a critical and challenging task in a machine-learning pipeline. The general idea of feature construction is to construct a set of new features $\{\phi_1, \ldots, \phi_m\}$ to improve the learning performance on a given dataset $\{\{x_1, y_1\}, \ldots, \{x_n, y_n\}\}$ rather than learning on the original features $\{x^1, \ldots, x^p\}$. In the feature construction domain, genetic programming (GP) based feature construction methods have gained significant success [4, 40]. GP has been widely used to automatically construct features because its flexible representation and gradient-free search mechanism make it suitable for constructing optimal features based on arbitrary datasets and machine learning algorithms, especially for enhancing non-differentiable machine learning algorithms.

Among all GP representations, tree-based GP is the most popular one for evolutionary feature construction tasks [34, 39, 40]. This is because features can be naturally represented by an expression tree ϕ. Based on the tree representation, variation operators in GP can iteratively cross two subtrees or mutate one subtree to a newly generated subtree to discover new features. Using the score provided by the fitness function and the selection pressure imposed by the selection operator, GP can gradually discover expressive features that improve the learning performance on a given dataset.

Although tree-based GP methods have become popular in the machine learning domain, bloat hinders their further development. Due to the bloat phenomenon in GP, it tends to generate large models, but the increase in model size does not improve performance. The explanations for bloat range from hitchhiking [33], to defense against crossover [3], removal bias [31] and the nature of a program search space [16]. The hitchhiking explanation [33] suggests that introns accidentally attach to a GP tree and survive during the selection process because they do not have an impact on fitness. The defense against crossover [3] and removal bias [31] hypotheses claim the variation operator on exons is often disruptive. Thus, bloated GP trees have a higher chance of survival because they can resist destructive variations. The nature of the program search space [16] explanation posits that there are more large and good programs than small and good programs in the GP search space because of the existence of various introns. Thus, it is easier for GP to find large and good programs.

Regardless of the reason for bloat, it is widely acknowledged that solving bloat can increase search efficiency and improves the interpretability of the final model [30]. In recent years, there are a lot of bloat control methods modifying variation operators, fitness functions and selection operators. Among these bloat control methods, selection-based bloat control methods received a lot of attention as it is easy to implement, and requires little domain knowledge.

In the GP domain, a relatively new operator named the lexicase selection [13] has gained wide attention in recent years. Different from the tournament selection operator which selects better-fitted parent individuals, the lexicase selection operator selects individuals based on the semantics of each individual. The semantics of a GP individual refers to the output or behavior of the individual. For regression problems, the semantics can be defined as $\{\hat{y_1}, \ldots, \hat{y_n}\}$. Then, based on the semantic information, fitness cases are defined as the mean squared error made on each data instance, i.e., $\{(\hat{y_1} - y_1)^2, \ldots, (\hat{y_n} - y_n)^2\}$. By considering the fitness cases of each individual, the lexicase selection operator constructs multiple filters to select GP individuals in a more diverse way, which avoids GP to get stuck at a local optimum. For regression problems, it is worth noting that the automatic ϵ lexicase selection operator [13] is used to replace the lexicase selection operator for better search effectiveness.

Although the lexicase selection operator has demonstrated better performance than the tournament selection operator in many cases, it still suffers from bloat and most existing bloat control strategies in selection operators are designed for the tournament selection operator [18, 19]. Consequently, it is worthwhile to develop a bloat control method for lexicase selection. Recently, a variant of the lexicase selection operator named Lexi^2 has been proposed to control bloat [9]. This selection operator uses the tree size as a criterion to break a tie when the lexicase selection operator still cannot select the best individual after examining all training cases. When facing a tie, the Lexi^2 operator prefers the smaller individual among the remaining individuals, thereby bloat can be reduced to a certain extent while not sacrificing the search effectiveness. Experimental results confirm its good performance in bloat control on Boolean problems and classification problems. However, such problems have discrete outputs, and thus devising a tie-breaking strategy to facilitate the generation of smaller GP individuals would take effect. For regression problems, existing studies show that only a few training cases are enough to select the parent individuals because of the continuous output [13]. Therefore, a tie-breaking strategy may not be effective in regression problems, necessitating the design of a bloat control strategy that is suitable for regression tasks.

1.1 Goals

The overall goal of this work is to develop a bloat control strategy to enhance the lexicase selection operator in tree-based evolutionary feature construction algorithms on regression problems [1]. Specifically, we aim to make GP prefer smaller models during the evolutionary process while not hurting search effectiveness, i.e., reducing the size of ineffective codes. To achieve this goal, we propose a double lexicase selection (DLS) operator in this paper. The general idea of the DLS operator is to split the selection process into two stages. In the first stage, the DLS operator selects multiple candidates based on fitness/semantics using the automatic ϵ-lexicase selection operator. In the second stage, the DLS operator selects the parent among all candidates using the roulette wheel selection operator to select an individual according to the model size. By employing such a two-stage selection process, we can take the model sizes into account as an optimization goal, while not reducing the selection pressure. In summary, this work covers the following three objectives:

- Introducing a two-stage selection mechanism to lexicase selection considering both fitness values and model size in the selection stage.
- Developing a size-based roulette wheel selection operator to have a high probability to select good individuals with smaller model sizes, while not completely ignoring good but big individuals.
- Investigating whether the proposed selection operator can surpass existing selection operators with bloat control methods.

1.2 Organization

The rest of this paper is organized as follows. Section 2 introduces the related work of bloat control methods and selection operators in GP. Section 3 shows the details of the proposed DLS operator. Section 4 and Section 5 present experimental settings and experimental results, respectively. Section 6 gives some further analysis of parameter settings and evolved models. The conclusion and future directions are presented in Section 7.

2 RELATED WORK

2.1 Bloat Control Methods

Research on bloat control in the GP domain has been developed over thirty years [20]. Generally speaking, bloat control strategies can be embedded into evaluation, selection or variation operators. For the bloat control strategy used in the selection operator, the most representative method is the depth limit method [12], which sets a hard limit to GP to avoid oversized models. Following up the depth limit method, researchers developed more advanced methods to dynamically determine the depth limit based on the distribution of good individuals [17, 30]. These methods estimate the size distribution of good individuals and try to generate or preserve individuals obeying the expected distribution.

As for the bloat control approach based on evaluation methods, the most intuitive way is to add model size as a part of the fitness evaluation, which is known as the parsimony pressure method [37]. However, it is not an easy task to determine the weight between the model performance and the model size. To avoid such a dilemma, Zhang and Rockett [41] developed a multi-objective method to balance the trade-off between model performance and model size. A problem of multi-objective-based GP (MOGP) is that some existing multi-objective evolutionary algorithms over-exploit small but low-performance solutions. For example, a recent study shows that over 30% GP trees in standard MOGP only have one node, which is not enough for achieving good performance [17]. To address this issue, an α-dominance relationship was introduced into MOGP [35]. Rather than balancing the trade-off between model performance

[1]Source Code:https://tinyurl.com/DLS-GPFC

and model size, the Tarpeian method [28] controls bloat by directly assigning bad fitness values to a part of individuals larger than the average tree size in the population.

In contrast to designing a bloat control strategy in selection operators and evaluation functions to passively control bloat, bloat control strategies in the variation operators actively control bloat by simplifying a GP tree. A representative example is size-fair crossover [15], which controls bloat based on the crossover bias theory [29]. Supposing the size of the first subtree to be crossed is s_a, the size-fair crossover operator restricts the size of the second subtree to be less than $2s_a + 1$. Another representative example of controlling bloat through variation operators is prune-and-plant [1], which randomly splits a GP tree into two separate trees and places them into the population. Compared to designing a bloat control strategy in selection operators, incorporating a bloat control method in variation operators may require more effort due to the existence of different representations in GP. For strongly typed genetic programming [3], linear genetic programming [5], stack-based genetic programming [32] and other variants of GP, bloat control strategies need to be redesigned to adapt to the change in variation operator caused by different representations [27].

For the bloat control strategies based on variation operators, there is a special kind of strategy named program simplification. Unlike traditional bloat control methods, program simplification methods require the simplified program to have equal or similar semantics to the original program. For the program simplification methods to guarantee exact equivalent semantics, representative examples include removing inactive codes or simplification by mathematical rules. The advantage of exact simplification is that it guarantees to not impair the model performance, but the magnitude of size reduction may be limited. As for approximate simplification methods, it can replace parent nodes with child nodes with similar semantics [11] or replace a subtree with a randomly generated semantically similar tree [22]. Compared to the exact simplification method, the approximate method can simplify programs even more but at the risk of compromising predictive performance.

2.2 Selection Operator

The selection operator is one driving force of GP because it determines which genetic materials will survive and reproduce. The most widely used selection operator in GP is the tournament selection operator. This operator has been widely studied in the GP domain in the past thirty years, e.g., enhancing it via taking multiple loss functions into consideration [38] or making it easy to use with an automatic tournament size control technique [36]. In recent years, researchers begin to take the semantic information of GP individuals in the selection process to encourage population diversity. The representative selection operators include: lexicase selection [13], Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [24] and GP with transformed semantics entropy-based diversity preserving framework (GPED) [6]. The key idea of the lexicase selection operator is to randomly construct filters based on semantics to filter out individuals until only one individual remains, where the simplest way is to filter out individuals which do not have the best semantic value on a randomly selected case i. However, this approach is too strict for regression problems, and thus La Cava et al. [13] propose an automatic ϵ-lexicase selection operator to allow individuals within the median absolute deviation to survive.

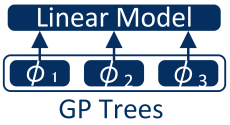


Figure 1: The multi-tree GP representation used in evolutionary feature construction.

From the perspective of bloat control, some researchers have already considered tree size as one criterion in the selection operator, creating several effective bloat control methods. Among these methods, the double tournament selection operator [19] is a representative one. The general idea of double tournament selection is to select two individuals based on the standard tournament selection operator and then select the smaller one with a higher probability. Existing literature shows that the double tournament selection operator is a strong competitor among bloat control methods [19]. As for a bloat control method for the lexicase selection operator, there is little related work. A representative algorithm is Lexi^2 [9] introduced in Section 1. However, Lexi^2 is designed for tasks with discrete output values. It should be useful developing a new bloat control method for lexicase selection, particularly for regression, which outputs continuous values.

3 ALGORITHM

In this section, a new variant of the automatic ϵ lexicase selection operator named double lexicase selection (DLS) is proposed and discussed in detail. We first present the model representation and the overall framework of a GP-based evolutionary feature construction method. Then, we describe the proposed double lexicase selection operator.

3.1 Model Representation

The proposed DLS operator is targeting a GP-based evolutionary feature construction scenario. In this paper, we employ a multi-tree GP representation for feature construction, as it is more flexible than the single-tree GP representation. As shown in Figure 1, each GP individual consists of m GP trees $\{\phi_1, \ldots, \phi_m\}$, representing m constructed features. Based on the constructed features $\{\phi_1(X), \ldots, \phi_m(X)\}$, a linear model is built to make predictions. The linear model is chosen because it is an efficient learning algorithm, and has shown superior performance in the evolutionary feature construction domain [14, 21].

3.2 Algorithm Overview

As shown in Figure 2, an evolutionary feature construction algorithm consists of four steps:

- Population Initialization: At the beginning of the evolution process, $|P|$ individuals are initialized randomly. Each individual consists of m trees initialized by the ramped half-and-half method to represent m randomly constructed features.
- Solution Evaluation: In the solution evaluation stage, the fitness cases and the fitness value of each individual are evaluated based on training data and a learning algorithm

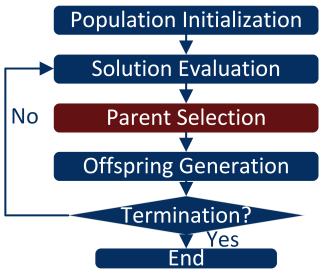


Figure 2: The workflow of DLS-based GP.

(i.e. a linear model). The fitness value is the sum of fitness cases, which is used to determine the best model, whereas the fitness cases are used specifically in lexicase selection.

- Fitness Cases: The whole model is evaluated on the training data through a leave-one-out cross-validation scheme. Suppose $\hat{y_k}$ represents the leave-one-out prediction made by the model on the data item k, and y_k represents the ground truth value, the vector of the square error on all data items $(\hat{y_k} - y_k)^2$ is defined as the fitness cases of each individual. The fitness cases are used for parent selection.
- Fitness Value: The fitness value is defined as the R^2 score over the data items, i.e., $1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$, where $\bar{y}$ represents the average target value of the training data. The fitness value is only used to determine the historically best individual, which is output as the final model. Fitness cases cannot be used for this purpose because it is hard to determine the best model based on a vector of square errors.

- Parent Selection: Parent selection refers to a stage where promising individuals are selected from the parent selection operator. In this paper, we apply a DLS operator to select parents based on the semantics/fitness and the model size of each individual.
- Offspring Generation: After selecting parents, random subtree crossover and random subtree mutation operators are applied to parents to generate offspring. For m-tree GP, the crossover and mutation operators are applied to m randomly selected GP trees to ensure sufficient variation.

Once population initialization is finished, the solution evaluation, parent selection and offspring generation steps are repeated until the stopping criterion is satisfied.

3.3 Double Lexicase Selection

Algorithm 1 presents the pseudocode of the double lexicase selection operator. The selection process consists of double stages, where lines 1-14 represent the first stage, and lines 15-19 represent the second stage.

3.3.1 First Stage Selection. For the first stage of selection, the DLS operator selects Cap individuals by using Cap rounds of the automatic ϵ-lexicase selection (ALS) operator, where Cap represents the capacity of a candidate pool. In each round, as shown in line 4, the ALS operator randomly picks one index $k \in \{1, 2, \cdots, n\}$ from the semantic vector of an individual ϕ according to the shuffled order.

Algorithm 1 Double Lexicase Selection

Input: Population P, Capacity of Candidates Pool Cap, Number of Data Items n
Output: Parent c_{final}

1: $C = \emptyset$ ▷ Candidates Pool
2: **while** $|C| < Cap$ **do** ▷ Select Cap individuals
3: $C' \leftarrow P$ ▷ Promising individuals
4: **for** $k \in$ shuffle($\{1, 2, \cdots, n\}$) **do** ▷ Random order
5: $C_k = \emptyset$
6: **for** $p \in C'$ **do**
7: **if** $\mathcal{L}_k(p) < min_{c' \in C'} \mathcal{L}_k(c') + \epsilon_k$ **then**
8: $C_k \leftarrow C_k \cup \{p\}$
9: $C' \leftarrow C_k$
10: **if** $|C'| = 1$ **then**
11: **break**
12: **if** $|C'| > 1$ **then**
13: $C' \leftarrow$ random select one from C'
14: $C \leftarrow C \cup C'$
15: $\{s_c | c \in C\} \leftarrow$ get the individual sizes(C)
16: **for** $c \in C$ **do**
17: $s_c \leftarrow \max_{c \in C} s_c + \min_{c \in C} s_c - s_c$
18: $c_{final} \leftarrow$ roulette wheel selection ($\{s_c | c \in C\}$)
19: **return** c_{final}

Then, in line 7, individuals in the current candidate pool $p \in C'$ that satisfy $\mathcal{L}_k(p) < min_{c' \in C'} \mathcal{L}_k(c') + \epsilon_k$ are preserved, where $\mathcal{L}_k(c') = (\hat{y_k} - y_k)^2$ represents the mean square error on the data item k. $min_{c' \in C'} \mathcal{L}_k(c')$ represents the minimal error for the data item k in the population and ϵ_k represents the median absolute deviation for the error of the data item k in the population [13]. The filtering process will continue until all indices $[1, n]$ have been traversed, or only one individual remains. If more than one individual remains after using all indices, an individual is randomly selected from the remaining individuals, as shown in line 13. In the end, as shown in line 14, the selected individual is added to a candidate pool C to wait for the next stage selection.

3.3.2 Second Stage Selection. After selecting Cap individuals to form a candidate pool C, the DLS operator first transforms the model size score of each individual via $s_c \leftarrow \max_{c \in C} s_c + \min_{c \in C} s_c - s_c$ in line 17. This transformation leads to the smaller individual having a larger score, forcing the selection operator to prefer smaller individuals among acceptable individuals. Then, in line 18, the DLS operator samples an individual proportional to the model size score s_i, i.e., as a roulette wheel selection. Based on this roulette wheel selection, a smaller individual has a higher chance to be selected, whereby bloat can be controlled. The reason for using roulette wheel selection is that greedily selecting the smallest individual from the candidate pool may force GP to focus on exploiting small individuals and thus hinders the algorithm from discovering complex but accurate models.

4 EXPERIMENTAL SETTINGS

To investigate the effectiveness of the double lexicase selection operator, a set of experiments has been conducted to compare the

predictive performance and the model size using different bloat control methods. This section describes the design of experiments, including datasets, baseline methods, parameter settings and the evaluation protocol.

4.1 Datasets

This paper uses the Penn Machine Learning Benchmark (PMLB) [25] to conduct experiments. PMLB is a curated list of datasets based on the OpenML repository. In this paper, 98 regression datasets are used in the experiments, which are all datasets in PMLB with less than 2000 data items. These 98 datasets are a variety of datasets with different properties. For example, the size of datasets range between 47 and 1059, and the dimension of datasets are between 2 and 124.

4.2 Baseline Methods

In this paper, we compare the DLS operator with the following seven bloat control methods on multi-tree GPs:

- Depth Limit: The depth limit method sets a strict depth limit for each GP tree. This method is used to control bloat in most GP algorithms.
- Double Tournament Selection (DTS) [19]: The double tournament selection operator first selects two individuals with a traditional tournament selection operator and then selects the smaller one among these two individuals with a higher probability.
- Tarpeian [28]: The idea of the Tarpeian method is to assign extremely poor values to a portion of individuals with a tree size greater than the average size.
- Prune and plant (PAP) [1]: Prune and plant is an active bloat control method that replaces a subtree with a random node and plants the pruned subtree as an individual in the population.
- α-MOGP [35]: α-MOGP uses NSGA-II to balance the trade-off between model performance and model size. Rather than using the traditional dominance relationship, α-MOGP defines an α-dominance relationship to avoid preserving small trivial solutions.
- TS-S [8]: Statistics tournament selection with size (TS-S) is a selection-based bloat control method proposed by Chu et al. [8]. The basic flow of TS-S is the same as the traditional tournament selection operator. The difference is that if the semantics of two individuals are not statistically different, the smaller individual is chosen. Otherwise, the individual with a higher fitness value is chosen.
- DSA [22]: Dynamic subtree approximation (DSA) randomly generates a small tree to replace a subtree in each GP tree that is larger than the average tree size. The generated subtree is linearly scaled to approximate the semantics of the replaced subtree.

4.3 Parameter Settings

Table 1 presents the parameter settings of GP. These parameter settings are common in the GP field [6]. In order to avoid the zero division problem, we use the analytical quotient (AQ) [23] to replace the division operator, which is defined as $\frac{a}{\sqrt{1+b^2}}$ for two input variables $\{a, b\}$. For all bloat control methods, we set a strict limit for the depth of GP trees to 10, as GP trees that exceed this limit are over-complex. The capacity of the candidate pool Cap is the only hyperparameter of the DLS operator, which is set as 10 based on the results shown in Section 6.1. For the hyperparameters of the baseline methods, tournament size and parsimonious size of the DTS method are set to 7 and 1.4, respectively, according to the suggestion provided by Luke and Panait [19]. Initial alpha value and step size in αMOGP are set to 0 and 90 respectively, which is recommended by the original paper [35]. Generally speaking, αMOGP will provide a set of non-dominated solutions. In this work, we select the solution with the best R^2 score as the final model, which is the same as in the original paper [35]. For the probability of applying the prune and plant operator, we set it to 1 as it is the optimal parameter for solving symbolic regression problems according to the existing literature [2]. For tournament size of the TS-S operator, it is set to 7 because this has shown good performance in the original paper [8].

Table 1: Parameter settings for all experiments.

Parameter	Value
Population Size	1000
Maximal Number of Generations	100
Crossover and Mutation Rates	0.9 and 0.1
Maximum Tree Depth	10
Maximum Initial Tree Depth	6
Number of Trees in An Individual	10
Functions	+, -, *, AQ, Sin, Cos, Abs, Max, Min, Negative

4.4 Evaluation Protocol

This paper applies a common evaluation protocol that is used in the existing literature. First, all experiments are repeated 30 times independently to ensure a reliable conclusion. In each independent run, each dataset is randomly split into a training set and a test set in a ratio of 80:20. To eliminate the magnitude differences between different features, we normalize all datasets before performing feature construction [26]. Wilcoxon signed-rank test results on R^2 and model sizes for each pair of bloat control methods with a significance level of 0.05 are reported.

5 EXPERIMENTAL RESULTS

In this section, we present and discuss the experimental results of GP with the proposed DLS operator on the 98 datasets by comparing them with the results obtained from seven bloat control methods. The comparison includes training performance, test performance and model sizes, demonstrating the effectiveness of the proposed method.

5.1 Comparisons on Test Performance

In this section, we present the test R^2 score of each bloat control method compared to show the predictive performance of the final model obtained by different bloat control methods. Table 2 shows the pair-wise statistical comparison results of the eight bloat control

Table 2: Statistical comparison of test R^2 scores for different bloat control methods. ("+","~", and "-" indicate using the method in a row is better than, similar to or worse than using the method in a column.)

	αMOGP	Tarpeian	DTS	PAP	TS-S	DSA	DepthLimit
DLS	13(+)/83(~)/2(-)	8(+)/90(~)/0(-)	37(+)/58(~)/3(-)	45(+)/49(~)/4(-)	46(+)/41(~)/11(-)	7(+)/91(~)/0(-)	14(+)/79(~)/5(-)
αMOGP	—	4(+)/87(~)/7(-)	35(+)/55(~)/8(-)	44(+)/45(~)/9(-)	38(+)/45(~)/15(-)	7(+)/88(~)/3(-)	4(+)/85(~)/9(-)
Tarpeian	—	—	35(+)/60(~)/3(-)	42(+)/50(~)/6(-)	39(+)/48(~)/11(-)	4(+)/94(~)/0(-)	7(+)/90(~)/1(-)
DTS	—	—	—	18(+)/76(~)/4(-)	7(+)/75(~)/16(-)	1(+)/71(~)/26(-)	13(+)/50(~)/35(-)
PAP	—	—	—	—	4(+)/71(~)/23(-)	1(+)/62(~)/35(-)	10(+)/48(~)/40(-)
TS-S	—	—	—	—	—	8(+)/65(~)/25(-)	19(+)/40(~)/39(-)
DSA	—	—	—	—	—	—	7(+)/82(~)/9(-)

methods. In short, only DLS, DSA and Tarpeian methods have similar or better performance on most datasets compared to the depth limit method. For the top three algorithms, the DLS method is better than the Tarpeian and DSA methods, as it outperforms the DSA and Tarpeian methods on 7 and 8 datasets respectively, while not worse on any dataset. Compared to the fourth-placed algorithm, αMOGP, the advantage of DLS is further verified, as it obtains significantly better test performance on 13 datasets and gets worse on only 2 datasets. The advantage of DLS over the DTS and TS-S methods is notable, where the DLS method outperforms them on 37 and 46 datasets, respectively, while worsening on only 3 and 11 datasets. The comparison on the test R^2 between DLS and PAP is the most notable where the DLS operator outperforms the PAP method on 45 datasets and is only worse on 4 datasets.

To further understand the behavior of the proposed method, we plot the convergence curve of test R^2 score on four datasets in Figure 3. The results show that the DLS operator has good effectiveness over the whole evolution process, and thus achieves good final accuracy. These results are consistent with the results presented in Table 2. One interesting observation worth noting is that the PAP, DTS and TS-S operators perform significantly worse than other bloat control methods in early generations, which is obvious in the "OpenML_618" dataset. The reason might be that these three bloat control operators aggressively reduce the model size, as shown in Figure 4 in the next section, restricting GP from finding more complex but better solutions. Therefore, GP with these bloat control methods cannot achieve comparable R^2 scores to GP with the DLS method.

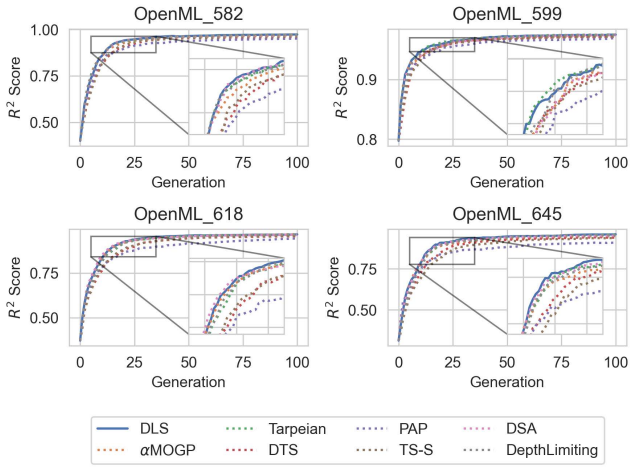


Figure 3: Evolutionary plots of test R^2 score for different bloat control methods.

5.2 Comparisons on Model Size

Table 3 presents pairwise comparisons of different bloat control methods for model sizes, referring to the average number of nodes of all trees. It shows that the DLS operator is a successful bloat control method, as it reduces model sizes on all datasets. When compared to Tarpeian, which is a method that has slightly worse R^2 scores, the final model size obtained with the DLS operator is significantly smaller on 19 datasets while only larger on one dataset, showing the advantage of using the DLS operator. Also, the DLS operator is significantly better than αMOGP on 86 datasets and not worse on any dataset. When comparing the PAP, DSA, DTS and TS-S operators, the DLS operator is worse at reducing model size, as the model size of the DLS operator is significantly larger than these four baseline algorithms on 42, 49, 87 and 88 datasets, respectively. However, keeping in mind that the DLS operator is better than the PAP, DSA, DTS and TS-S operators in terms of test R^2 scores, we can conclude that the DLS operator strikes the best balance between test accuracy and model size since test accuracy is the more important than model size in most machine learning tasks.

Besides showing numerical results, we also plot the convergence curve of average tree sizes in Figure 4 to gain a deeper understanding of the DLS operator. Figure 4 shows that all methods experience a reduction of model size in early generations, which is consistent with the results reported by [2] on the symbolic regression task. As evolution proceeds, the depth limit method cannot effectively control tree sizes, which leads to a rapid growth of tree sizes. In contrast, the DLS operator can effectively control tree size to a relatively low level, and thus achieves a better final result.

5.3 Overall Analysis

Based on the experimental results in the previous two subsections, we present the overall results in Table 4. The results in Table 4 show Friedman's rank of median test R^2 scores and tree sizes across all datasets. The number in parenthesis indicates the relative rank between the eight algorithms in terms of mean rank value. The

Table 3: Statistical comparison of model sizes for different bloat control methods. ("+","~", and "-" indicate using the method in a row is better than, similar to or worse than using the method in a column.)

	αMOGP	Tarpeian	DTS	PAP	TS-S	DSA	DepthLimit
DLS	86(+)/12(~)/0(-)	19(+)/78(~)/1(-)	2(+)/9(~)/87(-)	11(+)/45(~)/42(-)	0(+)/10(~)/88(-)	11(+)/38(~)/49(-)	98(+)/0(~)/0(-)
αMOGP	—	0(+)/11(~)/87(-)	0(+)/0(~)/98(-)	0(+)/27(~)/71(-)	0(+)/0(~)/98(-)	0(+)/1(~)/97(-)	74(+)/19(~)/5(-)
Tarpeian	—	—	0(+)/6(~)/92(-)	13(+)/41(~)/44(-)	0(+)/10(~)/88(-)	2(+)/43(~)/53(-)	98(+)/0(~)/0(-)
DTS	—	—	—	54(+)/28(~)/16(-)	31(+)/19(~)/48(-)	87(+)/11(~)/0(-)	98(+)/0(~)/0(-)
PAP	—	—	—	—	8(+)/22(~)/68(-)	43(+)/17(~)/38(-)	98(+)/0(~)/0(-)
TS-S	—	—	—	—	—	81(+)/14(~)/3(-)	98(+)/0(~)/0(-)
DSA	—	—	—	—	—	—	98(+)/0(~)/0(-)

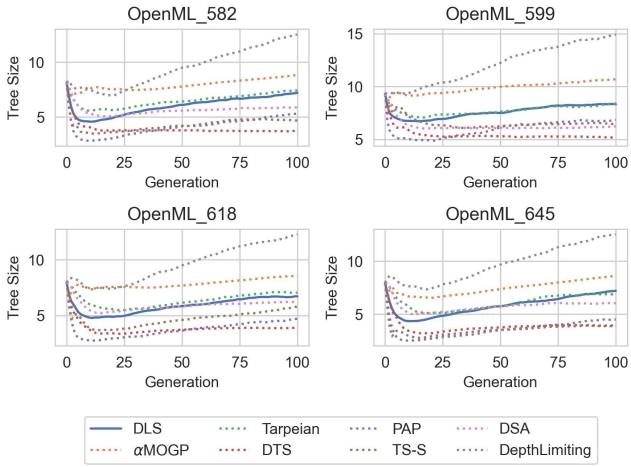


Figure 4: Evolutionary plots of average tree sizes for different bloat control methods.

results in Table 4 are similar to the results in the previous two subsections. For a good bloat control method, it should at least not get worse predictive performance than using depth-limited methods alone. Only four methods, DLS, Tarpeian, DSA and αMOGP meet this criterion. Further, when analyzing the bloat control effect of these four algorithms, it becomes clear that DLS achieves the best R^2 score rank and the second-best size rank among these four methods, indicating the DLS operator achieves a good trade-off between test R^2 scores and model size. It is worth noting that although some operators, like TS-S and DTS, are more successful in reducing model size, these operators significantly impair the predictive performance of the final model. Thus, these operators are less suitable for evolutionary feature construction tasks compared to the DLS operator, as the predictive performance of the model is often more important than model size in machine learning tasks.

6 FURTHER ANALYSIS

In this section, we further analyze the sensitivity of the only parameter in the DLS operator, i.e., the capacity of the candidate pool. Furthermore, the effectiveness of using the roulette wheel selection operator instead of selecting the smallest individual in the candidate pool of the DLS operator is also discussed.

Table 4: Friedman's rank of test R^2 scores and tree sizes on all datasets for different bloat control methods.

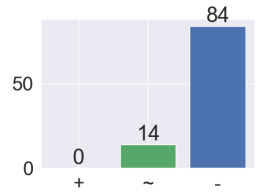
Algorithm	R^2 Score Rank	Size Rank
DLS	3.43 (1)	4.71 (5)
Tarpeian	3.93 (2)	4.92 (6)
DSA	4.16 (3)	3.9 (4)
αMOGP	4.22 (4)	7.06 (7)
DepthLimit	4.29 (5)	7.88 (8)
DTS	5.03 (6)	2.09 (2)
TS-S	5.15 (7)	1.7 (1)
PAP	5.79 (8)	3.74 (3)

6.1 Capacity of Candidate Pool

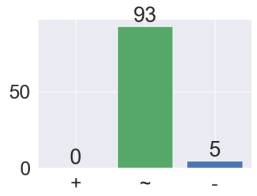
In the DLS operator, the capacity of the candidate pool is a crucial parameter, as it determines the number of candidates to be further selected by the roulette wheel selection operator. To study the impact of this parameter, we use the capacity of 10 as the baseline and compare three options {2,5,20} with the baseline and present the statistical comparison results of model sizes in Figure 5. The results of R^2 scores are shown in supplementary material, as the significant difference between different options is not much. Experimental results in Figure 5 show that model sizes decrease with an increase in capacity. One possible reason is that, by providing more choices to the roulette wheel selection operator, a better estimate can be obtained for the appropriate size of good individuals. Although increases in the capacity of the candidate pool can reduce final model size, marginal improvement decreases with increase in capacity. For example, increasing the pool size from 10 to 20 only improves results on 8 datasets and makes the results significantly worse on one dataset. Also, experimental results in Figure 5d show that training time will increase significantly when increasing pool capacity from 10 to 20. Thus, to achieve a good balance between time complexity and bloat control effect, setting capacity to 10 is the recommended option.

6.2 Roulette Wheel Selection

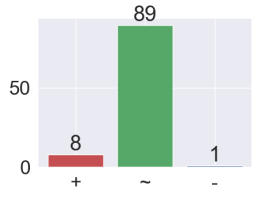
In the DLS operator, we use roulette wheel selection to select parents from candidate pools. One question is whether we can select the smallest individual from the candidate pool as parent, which would be easier to implement than the roulette wheel selection operator. In this section, we study this question by conducting an



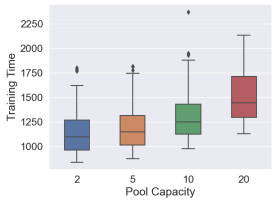
(a) Candidate Pool Capacity: 2.



(b) Candidate Pool Capacity: 5.



(c) Candidate Pool Capacity: 20.



(d) Distribution of training time versus candidate pool capacity.

Figure 5: Comparison of average tree sizes and training time using a capacity of 10 as the baseline. ("+","~", and "-" indicate using the compared capacity is significantly better than, similar to or worse than using the capacity of 10.)

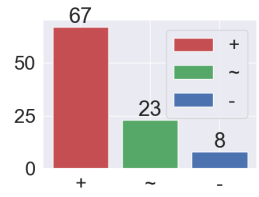


Figure 6: Statistical comparison of R^2 scores using roulette instead of minimum as the selection strategy.

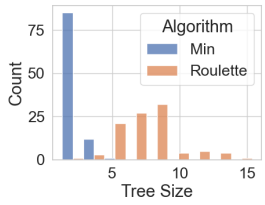


Figure 7: Distribution of tree sizes when using roulette or minimum as the selection strategy.

ablation study, which replaces the roulette wheel selection operator with the smallest selection operator. A pair-wise statistical comparison of using the roulette wheel selection operator to replace the smallest selection operator is presented in Figure 6. The experimental results in Figure 6 show that using the roulette wheel selection operator instead of the minimum selection operator to select a parent from a candidate pool will significantly improve the test R^2 score on 67 out of the 98 datasets, while only worsening it on 8 datasets. To further investigate the reason for causing this phenomenon, we plot the distribution of the average tree sizes in Figure 7. Figure 7 shows that the roulette wheel selection operator controls model sizes within a reasonable range. In contrast, the minimum selection operator favors very small individuals, thus creating final models that have a small model size but very poor predictive performance.

7 CONCLUSIONS

The goal of this paper was to develop a bloat control strategy to reduce the tree size of GP in the evolutionary feature construction scenario. This is accomplished by introducing a double-stage selection mechanism to the lexicase selection operator to consider both the model performance and the model size during the selection process. Moreover, we propose to use the roulette wheel selection operator in the second selection stage of the DLS operator to avoid the overuse of very small individuals. The performance of the proposed DLS operator has been examined on 98 regression datasets with different properties. Experimental results confirm that the DLS operator obtains a better trade-off between predictive accuracy and model size than the seven baseline bloat control methods. Further studies illustrate that the DLS operator can perform better by increasing the capacity of the candidate pool, and the effectiveness of using the roulette wheel selection operator in the DLS operator has been validated. This paper has demonstrated that using a double selection mechanism can boost the lexicase selection operator to reduce the model size. In the future, it is worthwhile to investigate whether such a mechanism can control the model complexity, such as the Rademacher complexity [7], to obtain a model with better generalization performance.

REFERENCES

[1] Eva Alfaro-Cid, Anna Esparcia-Alcázar, Ken Sharman, and Francisco Fernández de Vega. 2008. Prune and plant: a new bloat control method for genetic programming. In *2008 Eighth International Conference on Hybrid Intelligent Systems*. IEEE, 31–35.
[2] Eva Alfaro-Cid, JJ Merelo, F Fernández de Vega, Anna Isabel Esparcia-Alcázar, and Ken Sharman. 2010. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation* 18, 2 (2010), 305–332.
[3] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. 1998. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc.
[4] Ying Bi, Bing Xue, and Mengjie Zhang. 2022. Genetic Programming-Based Evolutionary Deep Learning for Data-Efficient Image Classification. *IEEE Transactions on Evolutionary Computation* (2022). https://doi.org/10.1109/TEVC.2022.3214503
[5] Markus Brameier, Wolfgang Banzhaf, and Wolfgang Banzhaf. 2007. *Linear genetic programming*. Vol. 1. Springer.
[6] Qi Chen, Bing Xue, and Mengjie Zhang. 2020. Preserving Population Diversity Based on Transformed Semantics in Genetic Programming for Symbolic Regression. *IEEE Transactions on Evolutionary Computation* 25, 3 (2020), 433–447.
[7] Qi Chen, Bing Xue, and Mengjie Zhang. 2022. Rademacher Complexity for Enhancing the Generalization of Genetic Programming for Symbolic Regression. *IEEE Transactions on Cybernetics* 52, 4 (2022), 2382–2395.
[8] Thi Huong Chu, Quang Uy Nguyen, and Michael O'Neill. 2018. Semantic tournament selection for genetic programming based on statistical analysis of error vectors. *Information Sciences* 436 (2018), 352–366.
[9] Allan de Lima, Samuel Carvalho, Douglas Mota Dias, Enrique Naredo, Joseph P Sullivan, and Conor Ryan. 2022. Lexi2: lexicase selection with lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 929–937.
[10] Stephen Dignum and Riccardo Poli. 2007. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. 1588–1595.
[11] David Kinzett, Mark Johnston, and Mengjie Zhang. 2009. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence* 2, 4 (2009), 151–168.
[12] John R Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing* 4, 2 (1994), 87–112.
[13] William La Cava, Thomas Helmuth, Lee Spector, and Jason H Moore. 2019. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation* 27, 3 (2019), 377–402.
[14] William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason H Moore. 2018. Learning concise representations for regression by evolving networks of trees. In *International Conference on Learning Representations*.
[15] William B Langdon. 2000. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines* 1, 1/2 (2000), 95–119.

[16] William B Langdon and Riccardo Poli. 1998. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*. Springer, 13–22.

[17] Dazhuang Liu, Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. 2022. Evolvability Degeneration in Multi-Objective Genetic Programming for Symbolic Regression. *arXiv preprint arXiv:2202.06983* (2022).

[18] Sean Luke and Liviu Panait. 2002. Fighting bloat with nonparametric parsimony pressure. In *International Conference on Parallel Problem Solving from Nature*. Springer, 411–421.

[19] Sean Luke and Liviu Panait. 2006. A comparison of bloat control methods for genetic programming. *Evolutionary Computation* 14, 3 (2006), 309–344.

[20] Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. 2022. Explainable Artificial Intelligence by Genetic Programming: A Survey. *IEEE Transactions on Evolutionary Computation* (2022).

[21] Kaustuv Nag and Nikhil R Pal. 2019. Feature extraction and selection for parsimonious classifiers with multiobjective genetic programming. *IEEE Transactions on Evolutionary Computation* 24, 3 (2019), 454–466.

[22] Quang Uy Nguyen and Thi Huong Chu. 2020. Semantic approximation for reducing code bloat in genetic programming. *Swarm and Evolutionary Computation* 58 (2020), 100729.

[23] Ji Ni, Russ H Drieberg, and Peter I Rockett. 2012. The use of an analytic quotient operator in genetic programming. *IEEE Transactions on Evolutionary Computation* 17, 1 (2012), 146–152.

[24] Kyle Nickerson, Antonina Kolokolova, and Ting Hu. 2022. Creating Diverse Ensembles for Classification with Genetic Programming and Neuro-MAP-Elites. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 212–227.

[25] Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData mining* 10, 1 (2017), 1–13.

[26] Caitlin A Owen, Grant Dick, and Peter A Whigham. 2022. Standardisation and Data Augmentation in Genetic Programming. *IEEE Transactions on Evolutionary Computation* 26, 6 (2022), 1596–1608.

[27] Michael Defoin Platel, Manuel Clergue, and Philippe Collard. 2003. Maximum homologous crossover for linear genetic programming. In *European Conference on Genetic Programming*. Springer, 194–203.

[28] Riccardo Poli. 2003. A simple but theoretically-motivated method to control bloat in genetic programming. In *European Conference on Genetic Programming*. Springer, 204–217.

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

[30] Sara Silva and Ernesto 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 (2009), 141–179.

[31] Terence Soule and James A Foster. 1998. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation Proceedings*. IEEE, 781–786.

[32] Lee Spector, Jon Klein, and Maarten Keijzer. 2005. The push3 execution stack and the evolution of control. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. 1689–1696.

[33] Walter Alden Tackett. 1994. *Recombination, selection, and the genetic construction of computer programs*. Ph. D. Dissertation. University of Southern California Los Angeles.

[34] Binh Tran, Bing Xue, and Mengjie Zhang. 2019. Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition* 93 (2019), 404–417.

[35] Shaolin Wang, Yi Mei, and Mengjie Zhang. 2022. A Multi-Objective Genetic Programming Algorithm with α dominance and Archive for Uncertain Capacitated Arc Routing Problem. *IEEE Transactions on Evolutionary Computation* (2022). https://doi.org/10.1109/TEVC.2022.3195165

[36] Huayang Xie and Mengjie Zhang. 2012. Parent selection pressure auto-tuning for tournament selection in genetic programming. *IEEE Transactions on Evolutionary Computation* 17, 1 (2012), 1–19.

[37] Byoung-Tak Zhang and Heinz Mühlenbein. 1995. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation* 3, 1 (1995), 17–38.

[38] Hu Zhang, Hengzhe Zhang, and Aimin Zhou. 2020. A Multi-metric Selection Strategy for Evolutionary Symbolic Regression. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 585–591.

[39] Hengzhe Zhang, Aimin Zhou, and Xin Lin. 2020. Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. *Complex & Intelligent Systems* 6, 3 (2020), 741–753.

[40] Hengzhe Zhang, Aimin Zhou, and Hu Zhang. 2022. An Evolutionary Forest for Regression. *IEEE Transactions on Evolutionary Computation* 26, 4 (2022), 735–749.

[41] Yang Zhang and Peter Rockett. 2007. A Comparison of three evolutionary strategies for multiobjective genetic programming. *Artificial Intelligence Review* 27, 2 (2007), 149–163.