A Semantic-Based Hoist Mutation Operator for Evolutionary Feature Construction in Regression

Hengzhe Zhang, *Member, IEEE*, Qi Chen, *Member, IEEE*, Bing Xue, *Senior Member, IEEE*, Wolfgang Banzhaf, *Member, IEEE*, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—In recent years, genetic programming (GP) has achieved impressive results on evolutionary feature construction tasks. To increase search effectiveness, researchers have developed many semantic-based crossover and mutation operators to guide GP searches toward the target semantics. However, semantics has not yet been explored for the hoist mutation operator, which is an operator designed for controlling the bloat effect. Although the hoist mutation operator can significantly reduce model sizes, the most informative subtree may be disrupted by the randomness in mutation. To address this issue, we develop a semantic-based hoist mutation (SHM) operator in this article to preserve the most informative subtree that has the largest cosine similarity between its semantics and the target semantics. Experimental results on 98 regression datasets from the Penn Machine Learning Benchmark show that using this operator not only significantly reduces model size but also improves the test accuracy of features constructed by GP. A comparison with seven bloat control methods shows that the proposed operator achieves the best tradeoff between accuracy and model size. Moreover, an experiment on the state-of-the-art symbolic regression benchmark shows that GP with the SHM operator achieves the best test accuracy and competitive model sizes compared with 22 symbolic regression and machine learning algorithms.**

***Index Terms*—Bloat control, evolutionary feature construction, evolutionary machine learning, genetic programming (GP).**

I. Introduction

FEATURE construction is an important task in the machine learning pipeline. For a dataset $\{X, Y\}$, feature construction constructs m high-order features $\Phi = \{\phi_1(X), \ldots, \phi_m(X)\}$ to improve the prediction accuracy of the machine learning algorithm on unseen data. Genetic programming (GP)-based feature construction methods have become popular in recent years and have achieved many impressive results [1], [2] due to their flexible representation and gradient-free search mechanism.

GP is an evolutionary algorithm that focuses on evolving variable-length solutions for solving complex optimization problems [2]. For evolutionary feature construction tasks, tree-based GP is a dominant method because the constructed features can be easily represented as expression trees [1], [2]. During GP evolution, genetic operators randomly cross two subtrees of two selected GP trees or randomly mutate a subtree of a selected GP tree to generate new GP trees. With the fitness value obtained using an evaluation function and the selection pressure introduced by the selection operator, GP can iteratively discover expressive features that enhance a specific machine-learning algorithm on a given dataset.

In tree-based GP, there is a phenomenon known as "bloat," where increasing the size of GP trees does not lead to better fitness values [3]. Bloat may be due to hitchhiking [4], defense against crossover [5], removal bias [6], or the nature of program search space [7]. The hitchhiking hypothesis [3] suggests that redundant building blocks are accidentally attached to good individuals and can survive in selection as they do not worsen fitness. The defense against crossover hypothesis [5] suggests that larger GP trees are less susceptible to destructive crossover, hence they have a higher chance to survive. The removal bias hypothesis [6] suggests that removing large subtrees is more detrimental than removing small subtrees, leading to an increase in the average tree size of the population. The nature of the program search space hypothesis [7] posits that large and good individuals are more abundant than small and good individuals, making it easier for GP to find large and good individuals.

Regardless of why bloat occurs, it is widely recognized that bloat can trap GP in local optima and impact the interpretability of the final model [8], [9], [10]. Many methods have been proposed to address this issue, including parsimony pressure [11], [12], dynamic depth limit [13], prune and plant (PAP) [14], multiobjective method [15], [16], and program simplification [17]. These methods have successfully reduced the size of GP trees. Among these methods, the PAP method has been shown to be effective for symbolic regression [9] as it actively prunes GP trees to reduce their size. PAP is a variant of the hoist mutation method [5], which is a bloat control

Manuscript received 2 April 2023; revised 24 July 2023 and 26 September 2023; accepted 26 October 2023. Date of publication 8 November 2023; date of current version 3 December 2024. This work was supported in part by the Marsden Fund of New Zealand Government under Contract VUW1913, Contract VUW1914, and Contract VUW2016; in part by the Science for Technological Innovation Challenge (SfTI) Fund under Grant E3603/2903; in part by the MBIE Data Science SSIF Fund under Contract RTVU1914; in part by Huayin Medical under Grant E3791/4165; and in part by the MBIE Endeavor Research Programme under Contract C11X2001 and Contract UOCX2104. *(Corresponding author: Qi Chen.)*

Hengzhe Zhang, Qi Chen, Bing Xue, and Mengjie Zhang are with the Centre for Data Science and Artificial Intelligence and the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: hengzhe.zhang@ecs.vuw.ac.nz; qi.chen@ecs.vuw.ac.nz; bing.xue@.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Wolfgang Banzhaf is with the Department of Computer Science and Engineering, College of Engineering, and the BEACON Center, Michigan State University, East Lansing, MI 48824 USA (e-mail: banzhafw@msu.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2023.3331234.

Digital Object Identifier 10.1109/TEVC.2023.3331234

method that randomly chooses a subtree from an individual and replaces it with a smaller subtree from within itself. PAP takes this further by hoisting a subtree to the root node as a new individual and randomly replacing the hoisted subtree in the original individual with a terminal variable. For both PAP and the hoist mutation operator, selecting the subtree to hoist is the most critical step. The quality and size of the selected subtree determine the changes in fitness values, as well as changes in individual sizes. However, the traditional hoist mutation operator, including the PAP variant, selects a subtree randomly from a GP tree. It may disrupt the informative components in a GP tree and generate many less informative trees in the population that are not beneficial to evolution.

To overcome the limitations of the current hoist mutation operator, we propose a semantic-based hoist mutation (SHM) operator in this article. In recent years, semantic GP has become popular in the GP domain [18], [19], [20], [21]. The general idea of semantic GP is to use the output values of GP trees to guide the search process in addition to the fitness value [18]. In an evolutionary feature construction scenario, for a given dataset $\{X, Y\}$, the semantics of a GP tree ϕ is $\phi(X) = \{\phi(X_1), \ldots, \phi(X_n)\}$. Semantic GP optimizes GP based on both behavior and objective spaces [22], resulting in better search performance [18] and population diversity [22].

The general idea of our new hoist mutation operator, SHM, in this work is to hoist the subtree with the largest semantic similarity to the target semantics to form a new GP tree.[1] By preserving the most informative part of the tree, SHM can significantly reducing tree size while improving fitness values. However, in multitree GP, we observed that some important subtrees frequently appear in different GP trees of a single individual, and hoisting only the most informative subtree resulted in the same tree appearing multiple times in a GP individual. To overcome this issue, we embed a hash-based checking strategy in the SHM operator to skip identical subtrees and maintain diversity in GP individuals. In summary, the key objectives of this article are as follows.

1) Develop an SHM operator. The new mutation operator aims to hoist the most informative subtree, thus controlling the bloat effect and improving the fitness values of GP models.
2) Design a hash-based checking strategy to avoid hoisting identical subtrees. This strategy assists GP in maintaining diverse features within each GP individual.
3) Provide a theoretical analysis of the generalization bound for the evolved GP models. This analysis aims to demonstrate the rationale behind using the proposed SHM operator.

The remainder of this article is organized as follows. Section II reviews related work on semantic GP and bloat control methods. Section III presents details of the proposed hoist mutation operator. Experimental settings and results are given in Sections IV and V, respectively. Section VI shows additional analysis, including ablation studies and model visualization. Finally, Section VII provides conclusions and future directions. The key notations used in this article are listed in Table I.

[1]Source code: https://tinyurl.com/SHM-GP.

TABLE I
NOTATIONS USED IN THIS ARTICLE

Symbol	Description
Φ	GP Individual
ϕ	GP Tree
$\phi(X)$	Constructed Feature
n	Population Size
m	Number of Trees in Each Individual
X	Original Feature
Y	Target Label
N	Number of Training instances

II. Related Work

A. Bloat Control

In GP, bloat refers to the tendency of solutions to become more complex over time without improving the fitness value [5]. Several hypotheses have been proposed to explain the reason for bloat, including hitchhiking [4], defense against crossover [5], removal bias [6], and the nature of the program search space [7]. While the explanations for bloat remain an ongoing subject of research, the benefits of bloat control in GP have been widely recognized [8], [10], [13]. For evolutionary feature construction tasks, controlling the size of GP-constructed features can make the final features more interpretable, improve search effectiveness, and mitigating overfitting [10]. Although researchers in machine learning have developed numerous regularization techniques to reduce functional complexity and prevent overfitting, it is important to note that bloat control techniques differ from overfitting control techniques, as bloat control techniques primarily aim to reduce structural complexity for searching interpretable models. Thus, researchers have developed many bloat control methods for GP.

In GP, bloat control methods can be applied in different stages: evaluation, selection, and variation. During the evaluation stage, the parsimonious pressure method penalizes model complexity within the fitness function to avoid evolving oversized models [23]. However, it is hard to determine the weight of the model complexity term [12]. To address this, multiobjective techniques have been widely used to balance fitness with complexity automatically [16], [24], [25]. Regrettably, the standard multiobjective GP (MOGP) approach may spend many resources on searching for trivial individuals. For example, a recent study on symbolic regression shows that more than 30% of GP trees in the final population in standard MOGP have only one node, which hinders GP from discovering good solutions [26]. To address this limitation, the adaptive α-dominance strategy [27] and some improved variants of NSGA-II [26] have been developed to encourage multiobjective techniques to discover more good solutions. Rather than considering fitness and complexity as a tradeoff, the Tarpeian method directly assigns extremely poor fitness values to a portion of larger-than-average individuals [28]. By doing so, larger individuals have a smaller chance to survive, thereby curbing bloat.

For the bloat control methods used in the selection stage, a representative example is lexicographic parsimony pressure [11]. This operator selects two individuals and takes the fitness value as the first objective to select the best parent,

meanwhile considering the individual size as the second objective in case of tied fitness. Other representative bloat control methods in the parent selection stage include proportional tournament selection and double tournament selection (DTS) methods [8]. One comparative study shows that the DTS method achieves superior performance among several selection-based bloat control methods on multiplexer and symbolic regression problems [8]. In addition to modifying the tournament selection operator, other selection operators, like the lexicase selection operator, can also be used to control the bloat effect [29]. Finally, several techniques in GP can automatically control the depth limit based on the fitness distribution [13], [30]. The dynamically controlled depth limit is used to determine which individuals to preserve, and thus can also be categorized as a bloat control method used in the selection stage.

Both bloat control methods in the evaluation and selection stages passively control bloat. In contrast, bloat control methods based on variation operators, e.g., crossover and mutation operators, actively control bloat [3], [9], [20], [31]. For example, the size fair crossover operator [3] first randomly selects a subtree a from the first parent with a size of s_a, then it limits the size of the second subtree, s_b, to less than $s_a * 2 + 1$, thereby controlling bloat. The hoist mutation operator [31] was proposed to explicitly reduce the model size by replacing a randomly chosen subtree with a subtree in itself. Based on the idea of the hoist mutation operator, Alfaro-Cid et al. [14] proposed the PAP operator to plant the pruned subtree to the population as a new individual instead of replacing the original individual. This operator avoids the loss of genetic material and achieves impressive results in a comparison of several bloat control methods [9]. However, existing variation-based bloat control methods rarely consider semantic information, and important genetic material may be disrupted after pruning. Thus, these algorithms still have a lot of room for improvement in terms of search effectiveness.

In addition to incorporating bloat control techniques in genetic operators, there is another kind of variation-operator-based bloat control method named program simplification [32]. Different from traditional bloat control methods, simplification-based methods require the simplified GP tree to have exactly or approximately equal semantics with the original GP tree. For the exact program simplification techniques, they simplify GP trees by removing inactive code [33] or using mathematical rules [34]. These methods ensure the semantics of the simplified GP tree is the same as that of the original GP tree, but designing these methods requires rich domain knowledge. Also, requiring exactly the same semantics could bring difficulties in finding smaller individuals, like simplifying $x + 10^{-10}$ to x. Thus, many approximate program simplification techniques are proposed to simplify GP trees. These methods include replacing parent nodes with semantically similar child nodes [35] and replacing a subtree with a randomly generated tree with similar semantics [20].

B. Semantic Genetic Programming

In the GP domain, semantics refers to the outputs or behavior of a GP individual [36]. Semantic GP refers to GP algorithms that use the semantics of GP individuals to guide evolution. Many semantic-based crossover and mutation operators have been proposed in [37]. A representative example is geometric semantic crossover (GSX) [38]. It guarantees the effectiveness of GP search by generating offspring with desired semantics. However, GSX faces the problem of exponential growth in model size, leading to expensive evaluations and uninterpretable models [39]. To address this, semantic approximation techniques have been developed. Some researchers [18], [40] developed semantic genetic operators that search for a subtree in an external library to make offspring approximate the target semantics without suffering from exponential growth.

As for selection operators, lexicase selection [41] is a representative example that uses semantics instead of fitness values to select parents. The key idea of the lexicase selection operator is to iteratively filter out less fit GP individuals in the current population P by a filter $\min_{p\in P}\mathcal{L}_k(P) + \epsilon_k$, where $k \in [1, n]$ is a random index within the n training instances/cases, $\min_{p\in P}\mathcal{L}_k(P)$ represents the minimum fitness value of the population P on fitness case k and ϵ_k indicates the mean absolute deviation of fitness values on case k. The filtering process is repeated until only one individual remains, or all cases are traversed. By examining different cases in each iteration, lexicase selection significantly enhances population diversity compared to fitness-based selection operators, which ultimately improves the quality of final results. Inspired by lexicase selection, several other semantic-based operators, like GPED [21] and MAP-Elites [42], have also been proposed to increase population diversity and search efficiency. Although both semantic-based variation and selection operators have gained wide attention, SHM operators remain underexplored. Given that hoist mutation is a powerful technique to control code bloat, it is worthwhile to investigate how to design an effective SHM operator.

C. Genetic Programming for Feature Construction

Evolutionary feature construction is a key technique in machine learning and has received wide attention [2], [16], [43]. Among these methods, GP-based feature construction methods have achieved superior performance to tackle regression [44], classification [1], [45], and clustering tasks [46]. Depending on the evaluation methods used for GP-constructed features, GP-based feature construction methods can be categorized as filter-based, wrapper-based, or embedded methods. Filter-based methods do not use any machine learning algorithm to evaluate the quality of constructed features. Instead, they use information gain [47], Pearson correlation [48], or other information-theoretic measures [45] to evolve expressive features to enhance arbitrary machine learning algorithms. However, since filter-based feature construction methods do not rely on a specific machine learning algorithm, they may not be able to discover features that work best for a particular algorithm. In comparison, wrapper-based feature construction methods evaluate features on a specific machine learning algorithm and thus can often have better performance than filter-based methods [44]. For instance, features can be evaluated using a decision tree (DT),

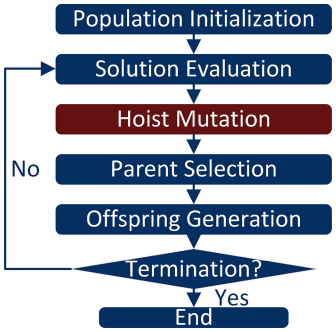


Fig. 1. Workflow of SHM-GP.

a linear regression (LR), a support vector machine (SVM), and an extreme gradient boosting (XGBoost) algorithm [2], [44]. Embedded feature construction methods refer to a kind of method that constructs features during the model learning process [49]. In general, the predictive performance and time cost of embedded methods lie between filter-based and wrapper-based methods. They are faster than wrapper-based methods because they only need to train the learning algorithm once. However, performing feature construction and model fitting simultaneously might be too difficult for GP, especially given that GP is not good at fitting coefficients [50].

Previous studies have successfully applied GP-based feature construction algorithms in various domains [44], [46], [51]. However, the issue of bloat in GP increases the program size, thus reducing search effectiveness and impairing the interpretability of discovered features. This article seeks to alleviate such a problem.

III. New Algorithm

In this section, a new SHM operator for GP-based feature construction is proposed and described in detail. We first introduce the overall algorithm of GP with the SHM operator (SHM-GP). Then, a semantic similarity calculation method, a hoist mutation operator, and some additional strategies are presented. Finally, we provide a generalization bound based on the Vapnik–Chervonenkis (VC) dimension to show the rationale for using the proposed SHM operator in GP.

A. Overall Framework

The SHM operator is proposed to reduce the size of GP trees in GP-based feature construction methods. Like many existing evolutionary feature construction methods [2], the proposed algorithm is based on a tree-based GP framework. As presented in Fig. 1, the overall algorithm consists of five steps: 1) population initialization; 2) solution evaluation; 3) hoist mutation; 4) parent selection; and 5) offspring generation. A brief introduction to these five steps is as follows.

1) *Population Initialization:* In SHM-GP, each GP individual has a multitree representation that consists of m trees to represent m constructed features. Thus, during the initialization stage, we randomly generate $n * m$ GP trees with the ramped half-and-half method to fill a population with n individuals.

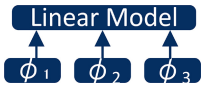


Fig. 2. Example of GP individual in SHM-GP.

2) *Solution Evaluation:* During the evaluation stage, we first transform the training data from the original features X to the constructed features $\{\phi_1(X), \ldots, \phi_m(X)\}$ using the m GP trees in a GP individual. Then, a linear model is trained on the constructed features to make predictions. As shown in Fig. 2, in this example, a linear model is trained on three features $\{\phi_1(X), \phi_2(X), \phi_3(X)\}$ constructed using the three GP trees $\{\phi_1, \phi_2, \phi_3\}$. The linear model is chosen because it is an effective and efficient machine learning model and is also easy to analyze with theoretical tools. To ensure that the constructed features can generalize well, the coefficients of the linear model are learned by a ridge regression method with the leave-one-out cross-validation algorithm. The leave-one-out errors on each data instance $\{1, 2, \ldots, N\} \rightarrow \{(\hat{y_1} - y_1)^2, \ldots, (\hat{y_N} - y_N)^2\}$ are recorded to form a fitness vector of individual Φ, which is used for lexicase selection. We use leave-one-out errors instead of training errors to form the fitness vector because we aim to discover features that can generalize well on unseen data. When determining the final model for predicting unseen data, the model with the minimum mean squared error is chosen because it is hard to choose the best model based on a fitness vector.
3) *Hoist Mutation:* After evaluation, the semantic hoist mutation operator is applied to each tree $\phi \in \Phi$ to extract the most informative subtree ψ_{best} and replace the original tree ϕ with the subtree ψ_{best}. The criterion for determining the most informative subtree ψ_{best} is introduced in Section III-B1.
4) *Parent Selection:* In order to fully exploit semantic information, SHM-GP utilizes the lexicase selection operator [41] to select parents. Unlike tournament selection that compares individuals based on the mean error on all cases, lexicase selection compares individuals case by case. By considering fitness values case-by-case, lexicase selection can select specialists that perform exceptionally well in a few cases, regardless of their overall fitness value. The lexicase selection operator is described in detail in Section II-B.
5) *Offspring Generation:* SHM-GP modifies two parents using random subtree crossover and random subtree mutation to generate two offspring. Unlike the traditional random mutation operator that randomly generates a new subtree, SHM-GP generates a subtree using a guided subtree generation (GSG) operator [43]. The general idea is to sample terminal variables according to the frequency of each terminal variable in good individuals, weighted by the importance value of each GP tree. Details of the GSG operator are introduced in Section A of the supplementary material. Since SHM-GP uses an m-tree representation, we perform m rounds of random-index crossover and random-index mutation on each

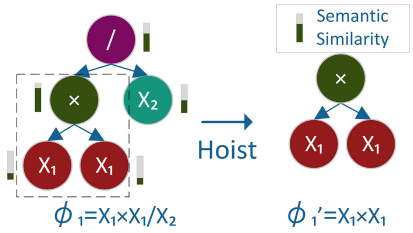


Fig. 3. Example of the SHM operator.

pair of parents to encourage exploration. In each round, all trees have an equal probability of being chosen for crossover and mutation.

B. Semantic-Based Hoist Mutation

1) Cosine Semantic Similarity: First, we need to define an indicator to measure the quality of each subtree to determine which subtree to hoist to the root. Since we train an LR model on the GP-constructed features $\{\phi_1(x), \ldots, \phi_m(x)\}$, the magnitude of each feature $\phi(X)$ is not important because the LR algorithm can automatically determine the optimal coefficient of each constructed feature. Therefore, using square error $(Y - \phi(X))^2$ to measure the semantic similarity between a feature $\phi(X)$ and the target semantics Y is not ideal because it does not consider the effect of LR. Instead, using cosine similarity

$$\theta = \cos(\phi(X), Y) = \frac{\sum_{i=1}^{n} \phi(\mathbf{x_i}) \cdot \mathbf{y_i}}{\sqrt{\sum_{i=1}^{n} \phi(\mathbf{x_i})^2} \sqrt{\sum_{i=1}^{n} \mathbf{y_i^2}}} \quad (1)$$

as the semantic similarity score is a better choice as it reflects the real quality of a feature with the LR technique. In SHM-GP, we use the absolute value of the cosine similarity since the sign issue can be addressed by LR later. It is worth noting that the cosine similarity score is sensitive to the shift. However, when using LR as the learning algorithm, it is desirable to let $\phi(X)$ and $\phi(X) + c$ have the same importance value since the shift term c can be automatically canceled out by adding a bias term in LR. In order to eliminate the impact of shift, we subtract the mean value $\overline{\phi(X)}$ from $\phi(X)$ before calculating the similarity score, i.e., $\phi(X) \leftarrow \phi(X) - \overline{\phi(X)}$. In SHM-GP, we calculate the cosine semantic similarity of each subtree ψ in a GP tree ϕ during the fitness evaluation phase, which can be computed during feature construction. Thus, it only increases the fitness evaluation time linearly.

2) Semantic Hoist Mutation Operator: Once the cosine similarity of each subtree $\psi \in \phi$ is obtained, SHM-GP enumerates all possible trees to get the optimal tree to be hoisted. Fig. 3 presents an example of the semantic hoist mutation operator. In this figure, a subtree $x_1 * x_1$ is hoisted to be a new tree as it has the highest cosine semantic similarity among all subtrees in $x_1 * x_1/x_2$, where the cosine semantic similarity is indicated by the percentage of the green bar. The pseudocode of the hoist mutation operator is presented in Algorithm 1. In this pseudocode, θ_{best} records the best similarity score in the traversal, and ψ_{best} stores the optimal subtree with the best similarity score. After traversing all subtrees, SHM-GP adds ψ_{best} to be the new individual Φ'.

Algorithm 1 Hoist Mutation

Input: A GP Individual $\Phi = \{\phi_1, \ldots \phi_m\}$, Training Data $\{X, Y\}$, Semantic Similarity of Each Node $\{\theta_\psi | \psi \in \phi\}$
Output: Mutated GP Individual $\{\phi'_1, \ldots, \phi'_m\}$

1: $\Phi' = \{\}$ ▷ Mutated GP Individual
2: **while** $|\Phi'| < |\Phi|$ **do**
3: *Success* $\leftarrow$ **false**
4: **for** $\phi \in \Phi$ **do**
5: $\theta_{best} \leftarrow 0$ ▷ Current best semantic similarity
6: $\psi_{best} \leftarrow \phi$ ▷ The optimal subtree
7: **for** $\psi \in \phi$ **do** ▷ Enumerate all subtrees
8: **if** $\theta_\psi > \theta_{best}$ and $\psi \notin \Phi'$ **then**
9: $\theta_{best} \leftarrow \theta_\psi$
10: $\psi_{best} \leftarrow \psi$
11: **if** $\theta_{best} > 0$ **then**
12: $\Phi' \leftarrow \Phi' \cup \{\psi_{best}\}$
13: *Success* $\leftarrow$ **true**
14: **if** $|\Phi'| == |\Phi|$ **then**
15: **break**
16: **if** !*Success* **then**
17: **break**
18: **return** Φ'

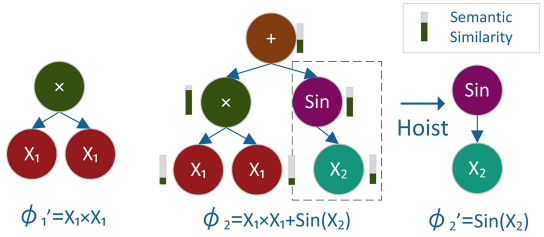


Fig. 4. Example of the semantic check.

After processing all trees $\phi \in \Phi$, the pruned individual Φ' is returned for generating offspring.

In SHM-GP, diversity may be significantly reduced with the proposed hoist mutation operator because an identical informative subtree ϕ' may appear in several trees of an individual Φ. Then, after applying the hoist mutation operator, ϕ' may appear multiple times in an individual Φ. This is a serious problem for evolutionary feature construction since redundant features produce no additional benefits but waste computational resources. For example, in Fig. 4, it is unreasonable to hoist the subtree $x1 * x2$ to be the second tree since it is the same as the first tree. To solve this issue, we propose a hash-based checking strategy in the hoist mutation operator. As shown in lines 4–17 of Algorithm 1, we sequentially evaluate all GP trees ϕ in an individual Φ. For each proposed optimal subtree ψ, we check whether ψ already exists in the individual Φ'. Each individual maintains a hash table, and thus the time complexity of redundancy checking is $O(1)$. If the subtree ψ does not exist in Φ', the best candidate tree ϕ can be replaced by the best subtree ψ. After iterating through all subtrees, if there is a subtree ϕ_{best} that satisfies $\theta_{\text{best}} > 0$, then ϕ_{best} is added to the individual Φ'. The check and hoist process will execute repeatedly until enough trees are hoisted or there is no subtree that can be hoisted, which is

shown in lines 14 and 16 of Algorithm 1. In Fig. 4, the best subtree apart from $x_1 * x_2$ is $\sin(x_2)$, which is hoisted to be the second tree because it is the best tree that can be hoisted without incurring redundancy.

3) Additional Strategies: Also, to further reduce model sizes and improve feature diversity and search effectiveness, we apply two strategies that accompany the SHM operator.

1) *Equivalent Node Prune [52]:* If a subtree ϕ'' has identical semantics to its upper-level subtree ϕ', it indicates that the upper-level subtree has redundant parts, and thus the upper-level subtree ϕ'' is replaced with ϕ'.
2) *Constant Node Prune [37]:* If the semantics of a subtree is equal to a constant value, then the subtree is replaced by this constant value.

C. Learning Guarantees

In this section, we focus on proving the rationale of developing the above SHM operator with the LR model.

Theorem 1: Consider a GP-constructed feature ϕ which has been standardized with zero mean and unit variance before training a linear model. Let θ be the cosine similarity between feature $\phi(x)$ and target variable y with n indicating the number of instances. Define h as the VC dimension [53] of a learning model with a GP tree ϕ and a linear model, that measures the complexity of functions that can be discovered by a GP tree ϕ with a linear model. Abbreviate h/n as p. The generalization loss of the constructed model can be bounded by

$$L_{\exp}(\phi) \leq \left(1-\theta^2\right)\left(1-\sqrt{p-p\ln p+\frac{\ln n}{2n}}\right)_+^{-1}$$

with the probability of at least $1-q$.[2]

Proof: Since the data are standardized to zero mean and unit variance, $\bar{y}=0$ and $\overline{\phi(x_i)}=0$, $([\sum_{i=1}^n(\phi(x_i)-\overline{\phi(x)})^2]/n)=1$ and $([\sum_{i=1}^n(y_i-\bar{y})^2]/n)=1$ are true. It means $\sum_{i=1}^n(\phi(x_i))^2=n$ and $\sum_{i=1}^n(y_i)^2=n$, and cosine similarity can be simplified as $\theta=(1/n)\sum_{i=1}^n y_i\phi(x_i)$. Given that the coefficient of a feature in the linear model is determined by the equation $\alpha=([\sum_{i=1}^n(y_i-\bar{y})(\phi(x_i)-\overline{\phi(x_i)})]/[(\sqrt{\sum_{i=1}^n(y_i-\bar{y})^2})(\sqrt{\sum_{i=1}^n(\phi(x_i)-\overline{\phi(x)})^2})]$ [19], the relationship between the sum squared error and θ can be established

$$\begin{aligned}
&\sum_{i=1}^n(y_i-\alpha\phi(x_i))^2\\
&\quad=\sum_{i=1}^n\left(y_i-\frac{\sum_{i=1}^n(y_i-\bar{y})(\phi(x_i)-\overline{\phi(x)})}{\sum_{i=1}^n(\phi(x_i)-\overline{\phi(x)})^2}\phi(x_i)\right)^2\\
&\quad=\sum_{i=1}^n\left(y_i-\frac{\sum_{i=1}^n y_i\phi(x_i)}{\sum_{i=1}^n\phi(x_i)^2}\phi(x_i)\right)^2\\
&\quad=\sum_{i=1}^n\left(y_i-\frac{n\theta}{n}\phi(x_i)\right)^2\\
&\quad=\sum_{i=1}^n\left(y_i^2-2y_i\theta\phi(x_i)+\theta^2\phi(x_i)^2\right)\\
&\quad=\sum_{i=1}^n y_i^2-2n\theta^2+\theta^2 n\\
&\quad=\sum_{i=1}^n y_i^2-n\theta^2=n\left(1-\theta^2\right).
\end{aligned}$$

Then, the following inequality holds [54]:

$$\begin{aligned}
L_{\exp}(\phi)&\leq\frac{1}{n}\sum_{i=1}^n(y_i-\alpha\phi(x_i))^2\left(1-\sqrt{p-p\ln p+\frac{\ln n}{2n}}\right)_+^{-1}\\
&=\frac{1}{n}\left(n-n\theta^2\right)\left(1-\sqrt{p-p\ln p+\frac{\ln n}{2n}}\right)_+^{-1}\\
&=\left(1-\theta^2\right)\left(1-\sqrt{p-p\ln p+\frac{\ln n}{2n}}\right)_+^{-1}.
\end{aligned}$$

Theorem 1 shows that the generalization bound is negatively correlated with θ^2. Therefore, it is reasonable to use θ^2 as a criterion to select the hoist point to ensure that the semantic similarity of the new tree ψ_{best} is no less than that of the old tree ϕ, i.e., $1-(\theta^2_{\psi_{\text{best}}})\leq 1-(\theta^2_{\phi})$. Additionally, since the hoisted subtree ψ_{best} is a part of the tree ϕ, the VC-dimension of ψ_{best} is not larger than the VC-dimension of ϕ, i.e., $h_{\psi_{\text{best}}}\leq h_\phi$. Thus, the generalization bound of the hoisted subtree ψ_{best} is not larger than that of the original feature ϕ, which means that the hoisted subtree ψ_{best} generalizes at least as well as the original tree ϕ, showing an advantage of using the SHM operator for machine learning tasks.

It is worth noting that this generalization bound only applies to a single-tree GP. For a multitree GP, feature interaction will make the above analysis more complicated, and the generalization bound will be more complex. Nonetheless, the generalization bound on the single-tree GP shows that hoist mutation has a solid theoretical foundation for a special case of multitree GP, and experimental results in Section V show that the SHM operator can have good generalization performance for a multitree GP.

IV. Experimental Settings

A. Datasets

In this article, experiments are conducted on regression datasets from Penn Machine Learning Benchmark (PMLB) [55], which is a curated list of datasets based on the OpenML repository. Due to limited computational resources, experiments are conducted on 98 datasets with fewer than 2000 instances. Detailed information about these datasets can be found in the supplementary material. In the section comparing SHM-GP with other SR and ML algorithms, all 120 datasets in PMLB are used to be consistent with the state-of-the-art symbolic regression benchmark (SRBench) [56]. The number of instances for these datasets ranges from 47 to 1 000 000, while the number of features varies between 2 and 124.

[2] $x_+=\max(x,0)$ represents the rectifier function.

TABLE II
PARAMETER SETTINGS FOR SHM-GP

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 One Individual	10
Elitism (Number of Individuals)	10
Functions	Add, Sub, Mul, AQ, Sin, Cos, Abs, Max, Min, Neg

B. Evaluation Protocol

This article follows the conventional evaluation protocol of evolutionary machine learning. Specifically, each dataset is divided into a training set and a test set at a ratio of 80:20. In order to eliminate the influence of different scales of datasets, all datasets are standardized before training [57]. To remove the influence of scale on target labels, test scores are reported using the R^2 score metric, which has a range of [0, 1]. Formally, R^2 is defined as $1-([\sum_i(y_i-\hat{y}_i)^2]/[\sum_i(y_i-\bar{y})^2])$, where $\hat{y}_i$ represents the prediction value on data item i, and $\bar{y}$ indicates the average of ground truth values y. To get a reliable conclusion, each algorithm is tested on each dataset with 30 independent runs with different random seeds, where each dataset is randomly shuffled at the beginning of each independent run to ensure randomness of the training data. After finishing all experiments, a Wilcoxon signed-rank test with a significance level of 0.05 is applied to verify the significance of the performance difference between the proposed method and baseline methods.

C. Baseline Methods

In this article, we implement seven GP approaches with bloat control methods as a baseline for comparison.

1) *Standard GP With Depth Limit:* The depth limiting method sets a depth limit for each GP tree. Most existing GP approaches for feature construction adopt this method to control bloat and improve generalization performance.
2) *DTS [8]:* The DTS method was proposed by Luke and Panait [8]. It first selects two individuals using tournament selection based on fitness values. Then, it selects the smaller individual as a parent with a higher probability.
3) *Tarpeian [28]:* The general idea of Tarpeian is to directly assign poor fitness values to a fraction of individuals that exceed the average tree size.
4) *PAP [14]:* The PAP operator prunes a subtree from the population by replacing it with a random node, and plants the pruned subtree in the population as a new tree.
5) *α-MOGP [27]:* α-MOGP designs an α adaptation scheme to automatically balance fitness and complexity under the NSGA-II framework [58]. Such a strategy is proposed to avoid MOGP spending a lot of resources on exploiting trivial solutions.

TABLE III
PARAMETER SETTINGS FOR BLOAT CONTROL IN DIFFERENT ALGORITHMS

Algorithm	Parameter Settings
DTS [8]	Tournament size = 7, Parsimony size = 1.4
Tarpeian [8]	Reduced fraction = 0.3
PAP [9]	Pruning Probability = 0.5
αMOGP [27]	Initial alpha value = 0, Step size = 90
TS-S [59]	Tournament size = 7

6) *Tournament Selection With Size (TS-S) [59]:* Statistics TS-S was proposed by Chu et al. [59]. The key idea of TS-S is to keep the smaller solution when the semantics of the two GP individuals are not significantly different in the tournament selection process. If the semantics of the two individuals are significantly different, TS-S keeps the solution with the better fitness value.
7) *Dynamic Subtree Approximation DSA [20]:* DSA is a bloat control method based on the mutation operator. For every GP tree larger than the average tree size, DSA randomly replaces a subtree with a randomly generated smaller subtree. A linear scaling technique is used to approximate the semantics of the original subtree with the semantics of the generated subtree.

For a fair comparison, all these bloat control methods are applied in the same GP-based feature construction framework as SHM-GP.

D. Parameter Settings

In the following experiments, all GP with bloat control methods are tested using the same parameter settings. Table II presents the parameter settings of GP. These parameter settings are common in the GP field [21]. The number of GP trees is set to 10, as it has shown good performance on evolutionary feature construction tasks [2]. In this article, we use the analytical quotient (AQ) [60] instead of the division operator to avoid zero division error. To avoid generating over-complex features, the depth limit of GP trees is set to 10. All bloat control parameters are set according to the existing literature, as shown in Table III. Hyperparameters for SR and ML methods in SRBench are tuned using the successive-halving grid search method according to the parameter grid defined in SRBench [56]. In the comparison experiments of bloat control methods, the SHM operator does not include the GSG strategy to examine if the improvement is made by the hoist mutation operator. In the comparison experiments with other SR and ML methods, SHM-GP uses the GSG strategy to enhance search efficiency.

V. EXPERIMENTAL RESULTS

This section presents and compares the experimental results of the SHM operator on 98 datasets with seven benchmark methods, including training accuracy, test accuracy, and model size, showing the effectiveness of the proposed method. A comparison between GP with the SHM operator and the state-of-the-art regression methods is also shown. This article primarily focuses on regression problems, but experiments

TABLE IV
STATISTICAL COMPARISON OF TEST R^2 SCORE 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	DepthLimiting
SHM	29(+)/68(∼)/1(-)	13(+)/81(∼)/4(-)	35(+)/57(∼)/6(-)	46(+)/49(∼)/3(-)	27(+)/65(∼)/6(-)	19(+)/74(∼)/5(-)	13(+)/80(∼)/5(-)
αMOGP	—	2(+)/78(∼)/18(-)	11(+)/79(∼)/8(-)	40(+)/48(∼)/10(-)	5(+)/76(∼)/17(-)	1(+)/86(∼)/11(-)	3(+)/75(∼)/20(-)
Tarpeian	—	—	22(+)/73(∼)/3(-)	44(+)/52(∼)/2(-)	11(+)/80(∼)/7(-)	7(+)/89(∼)/2(-)	4(+)/91(∼)/3(-)
DTS	—	—	—	32(+)/60(∼)/6(-)	4(+)/87(∼)/7(-)	1(+)/73(∼)/24(-)	6(+)/64(∼)/28(-)
PAP	—	—	—	—	2(+)/57(∼)/39(-)	1(+)/58(∼)/39(-)	4(+)/51(∼)/43(-)
TS-S	—	—	—	—	—	6(+)/85(∼)/7(-)	12(+)/62(∼)/24(-)
DSA	—	—	—	—	—	—	8(+)/81(∼)/9(-)

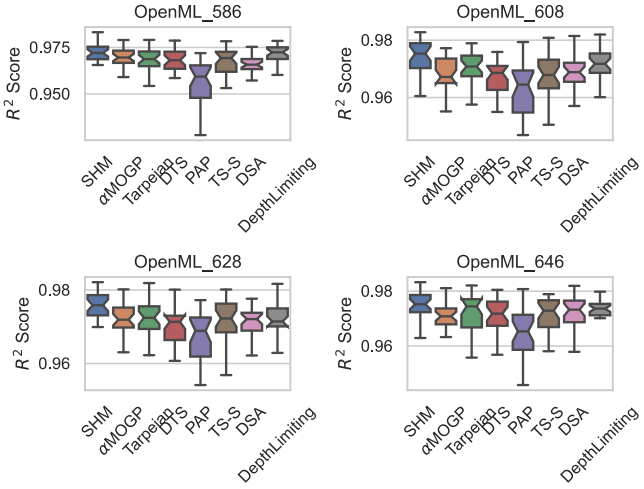


Fig. 5. Boxplots of test R^2 score for different bloat control methods.

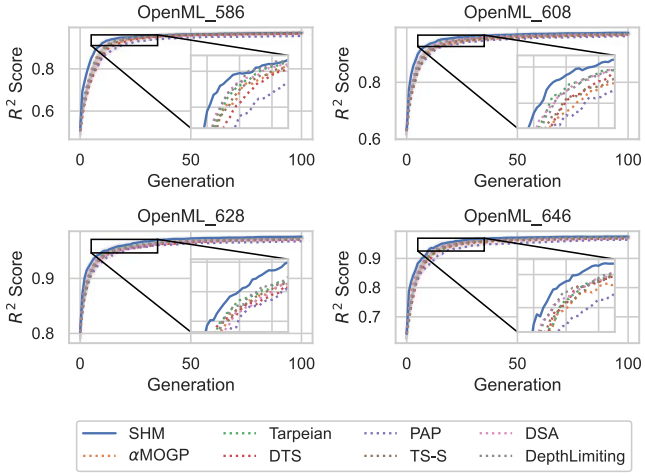


Fig. 6. Evolutionary plots of test R^2 score for different bloat control methods.

on classification also demonstrate the effectiveness of the proposed method. These results are presented in the supplementary material.

A. Comparisons on Predictive Performance

First, we present the test losses of different bloat control methods. Table IV presents the statistical comparison of test losses for different algorithms. Generally speaking, among the eight bloat control methods, the SHM operator is the best in terms of R^2 scores. Specifically, experimental results in Table IV show that the SHM operator significantly outperforms the second-best method, Tarpeian, on 13 datasets. Meanwhile, the SHM operator has a similar performance to the Tarpeian method on 81 datasets. Compared to the depth limiting method, the SHM operator has similar advantages, as it also outperforms the depth limiting method on 13 datasets. The DSA and TS-S are fourth and fifth-rank methods, and they have worse performance than SHM on 19 and 27 datasets, respectively. There is a large gap between the performance of SHM and that of αMOGP, DTS, and PAP, where SHM outperforms these three methods on 29, 35, and 46 datasets, respectively. Based on these results, it is clear that the SHM operator is the best bloat control method in terms of predictive performance, validating the effectiveness of using semantic information for bloat control. Fig. 5 shows the distribution of test R^2 scores from 30 independent runs on 4 exemplar datasets. The results further confirm that the SHM operator keeps comparable performance to the depth limiting method and has superior performance over other bloat control methods.

In order to fully understand the behavior difference of different bloat control methods, we plot the convergence curve of median test R^2 scores over the 30 independent runs in Fig. 6. The figure shows that GP with SHM achieves very good performance in early generations, which is not seen in other bloat control methods. This is because the most informative subtree, like ψ, may be submerged in the complex nonlinear transformation of GP, like $\psi^2 - \psi^2$, and become an uninformative GP tree. If we use traditional genetic operators, we may need several generations to pull up the most informative feature ψ to the root, which hinders GP to perform well in early generations. In contrast, the SHM operator can directly identify the most important subtree at the end of each generation, and modify the GP tree with a guarantee that the generalization upper bound will not become worse, thereby significantly improving the predictive performance in early generations. Moreover, from the perspective of long-term change of the convergence curve, Fig. 6 shows that SHM advantages can last throughout evolution, thus further verifying the effectiveness of the SHM operator.

B. Comparisons of Tree Size

The main objective of the SHM operator is to control bloat. Thus, this section compares the final number of nodes obtained with different bloat control methods. For evolutionary feature

TABLE V
STATISTICAL COMPARISON OF *Tree Size* 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	DepthLimiting
SHM	98(+)/0(∼)/0(-)	98(+)/0(∼)/0(-)	61(+)/37(∼)/0(-)	55(+)/40(∼)/3(-)	58(+)/35(∼)/5(-)	94(+)/4(∼)/0(-)	98(+)/0(∼)/0(-)
αMOGP	—	0(+)/5(∼)/93(-)	0(+)/0(∼)/98(-)	0(+)/2(∼)/96(-)	0(+)/1(∼)/97(-)	0(+)/0(∼)/98(-)	39(+)/51(∼)/8(-)
Tarpeian	—	—	0(+)/20(∼)/78(-)	0(+)/38(∼)/60(-)	0(+)/34(∼)/64(-)	5(+)/45(∼)/48(-)	98(+)/0(∼)/0(-)
DTS	—	—	—	32(+)/32(∼)/34(-)	29(+)/37(∼)/32(-)	47(+)/47(∼)/4(-)	98(+)/0(∼)/0(-)
PAP	—	—	—	—	25(+)/44(∼)/29(-)	49(+)/27(∼)/22(-)	98(+)/0(∼)/0(-)
TS-S	—	—	—	—	—	60(+)/30(∼)/8(-)	98(+)/0(∼)/0(-)
DSA	—	—	—	—	—	—	98(+)/0(∼)/0(-)

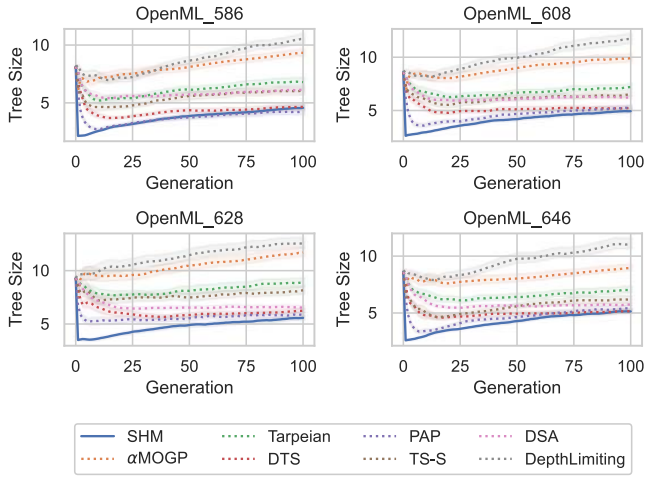


Fig. 7. Evolutionary plots of average tree sizes for different bloat control methods ("OpenML_X" represents the dataset with the id of X in OpenML).

construction tasks, each GP individual contains ten trees. For simplicity, the tree size of each individual in this section is defined as the average tree size of all trees in an individual. Experimental results in Table V show that the SHM operator can significantly reduce tree size on all 98 datasets compared to using the depth limiting method. Compared to Tarpeian, which is only slightly worse than the SHM operator on test R^2 scores, the SHM operator finds significantly smaller solutions on all datasets. In comparison with TS-S and DSA, which are fourth-ranked and fifth-ranked in terms of test R^2 scores, the SHM operator outperforms them on 58 and 94 datasets, respectively, while being worse on no more than five datasets. The αMOGP and PAP are effective bloat control methods, as they significantly reduce model size on 39 and 98 datasets compared to the depth limiting method. However, the model size produced by these methods is significantly worse than the model size produced by the SHM operator on 98 and 55 datasets. In general, all advanced bloat control methods can reduce the tree size better than the depth-limiting method. However, these methods have a relatively limited effect on bloat control compared to the SHM method.

Fig. 7 shows average tree size as evolution progresses. The results indicate that the average tree size decreases at the beginning for all methods and then gradually grows. Among the eight methods, the SHM operators can reduce tree size in early generations more than other bloat control methods. As iterations increase, the bloat control strategy based on DSA and DTS can maintain average tree size at a stable level. In

TABLE VI
FRIEDMAN'S RANK OF R^2 TEST SCORES AND AVERAGE TREE SIZES ON ALL DATASETS FOR DIFFERENT BLOAT CONTROL METHODS. (THE RELATIVE RANKS ARE PRESENTED IN PARENTHESES)

Algorithm	R^2 Rank	P-Value	Size Rank	P-Value
SHM	3.05 (1)	-	1.48 (1)	-
Tarpeian	3.88 (2)	1.9e-02	5.42 (6)	0.0e+00
DepthLimiting	3.96 (3)	1.9e-02	7.83 (8)	0.0e+00
DSA	4.07 (4)	1.1e-02	4.48 (5)	0.0e+00
TS-S	4.55 (5)	7.3e-05	3.12 (2)	4.0e-06
αMOGP	5.01 (6)	1.1e-07	7.17 (7)	0.0e+00
DTS	5.11 (7)	2.3e-08	3.14 (3)	4.0e-06
PAP	6.37 (8)	0.0e+00	3.35 (4)	2.6e-07

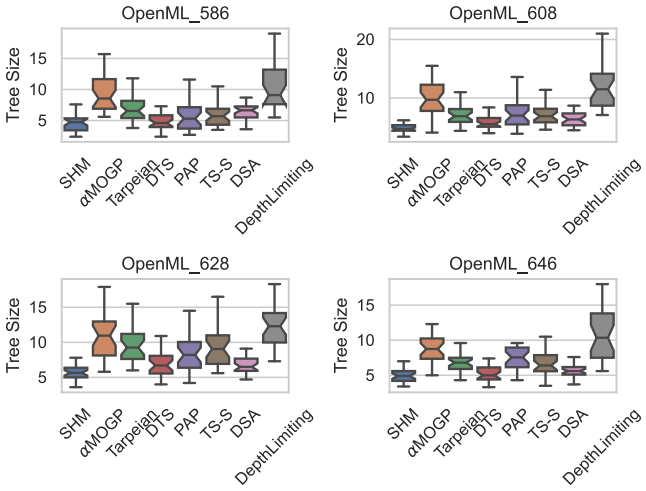


Fig. 8. Boxplots of average tree sizes for different bloat control methods.

contrast, the average tree size goes up during evolution if using SHM as a bloat control technique. It is worth noting that the goal of a bloat control technique is not to limit average tree size at a stable level as the flexible length representation is one vital advantage of GP. Instead, bloat control techniques should be able to allow GP trees to grow within a reasonable range. From this point of view, SHM is more appropriate than other bloat control techniques, and this can explain why the SHM method can achieve the best R^2 scores among eight bloat control methods as shown in Table IV. The tree size distribution on 4 representative datasets over 30 independent runs is presented in Fig. 8, verifying that the SHM method is robust to reduce model size. Fig. 9 presents the distribution of average tree size on each dataset. We can find that the average GP tree size on all problems based on the SHM operator is spread around a mean value of 4.5. In comparison, the

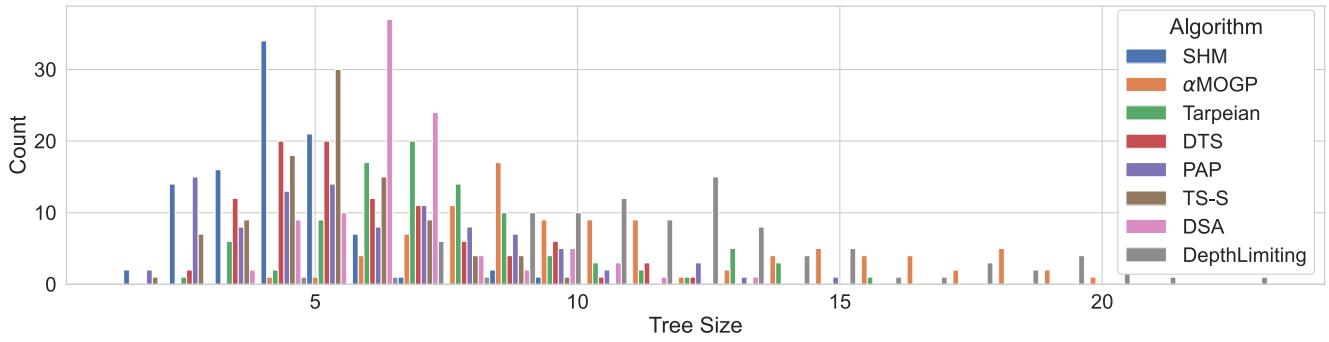


Fig. 9. Distribution of tree size for different bloat control methods.

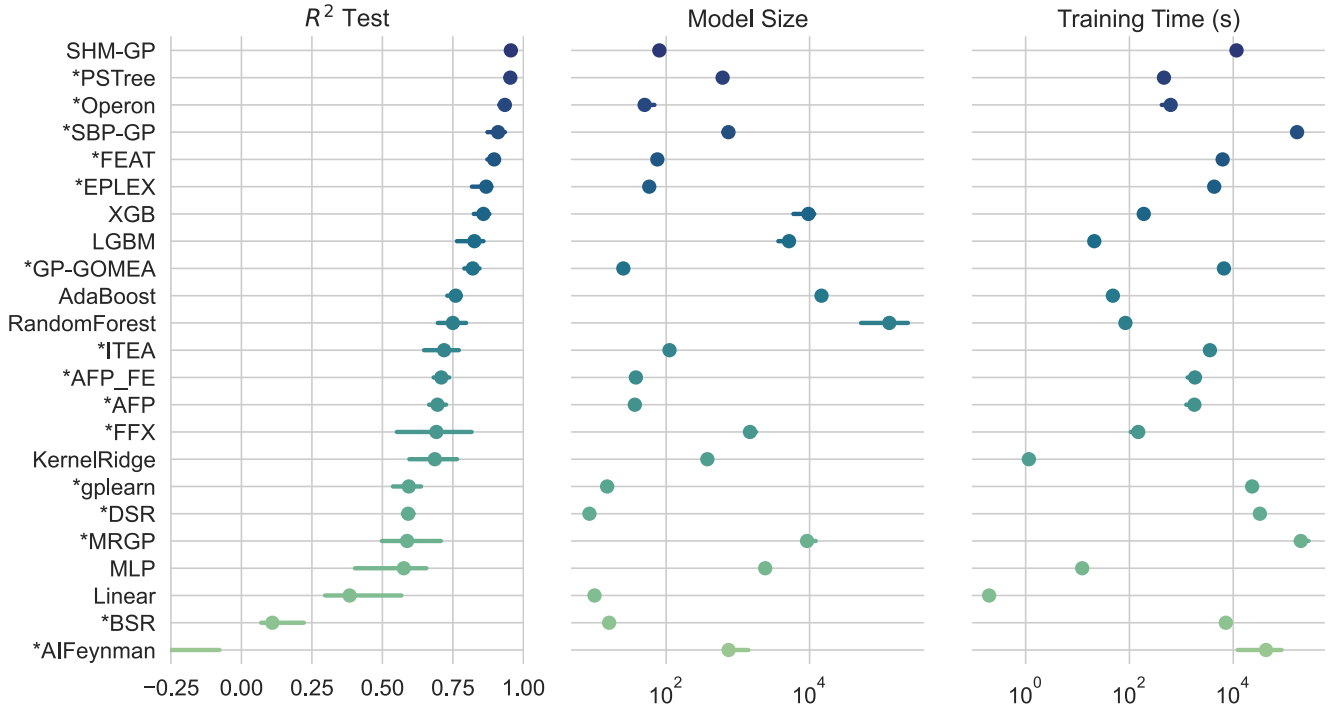


Fig. 10. R^2 scores, model sizes, and training time of 23 algorithms on 120 regression problems.

average tree size induced by the depth limiting method and the αMOGP method can be up to around 20. This suggests that for some datasets, the depth limiting method and the αMOGP method may yield large solutions, while the SHM operator has a relatively stable behavior on all 98 datasets. Based on these results, we can conclude that the SHM operator is a stable method that can reduce model size on a wide range of problems without impeding the search progress.

C. Overall Comparisons

Summary results combining the test R^2 scores and average tree size are shown in Table VI. The SHM operator ranks first in both test R^2 scores and average tree size. For an ideal bloat control method, it should perform as well as the bloat control method in test R^2 scores and minimize model size. Among the bloat control methods, only SHM and Tarpeian have better mean ranks of R^2 scores than the depth limiting method. In addition, DSA and TS-S are slightly worse than the depth-limiting method. Further comparing the R^2 scores and the model size between these four methods, it can be seen that the SHM operator has a substantial advantage over Tarpeian and DSA in model size and a big advantage over DSA and TS-S in R^2 scores indicating that the SHM operator strikes the best balance between accuracy and complexity.

D. Comparisons With Other Symbolic Regression Algorithms

This section follows the evaluation protocol of SRBench [56], to compare the proposed SHM-GP algorithm with 22 algorithms on all 120 datasets in PMLB. Fig. 10 presents the median test R^2 score, model size, and training time distribution for all the 23 algorithms. The model size of SHM is defined by the number of nodes in Sympy format in the final model, including linear coefficients and constructed features, which strictly follow the requirement of SRBench. The model size of other methods is defined by their respective developers. From this figure, it is clear that SHM-GP achieves superior performance compared to other baseline methods while having a similar level of complexity compared to Operon [61]. It is worth noting that the actual number of nodes in the final model of SHM-GP is lower than the reported model size because we

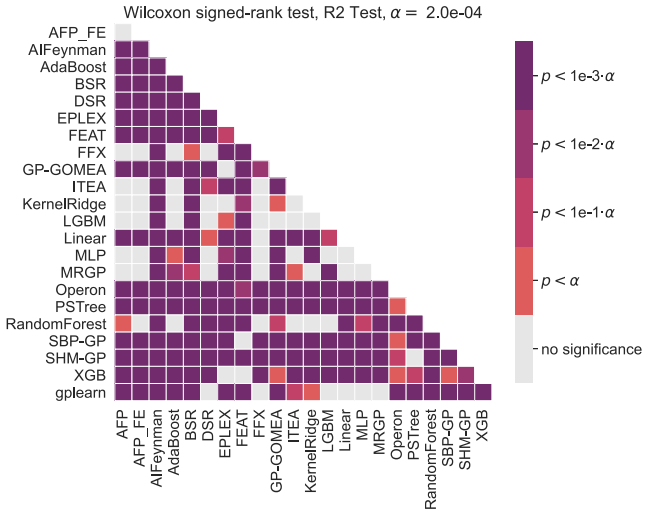


Fig. 11. Pairwise statistical comparisons of R^2 test scores on regression problems.

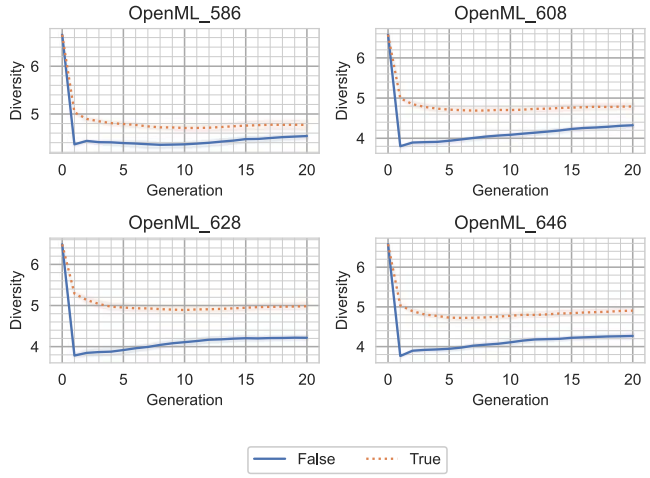


Fig. 12. Phenotype entropy for using hash-based checking or not.

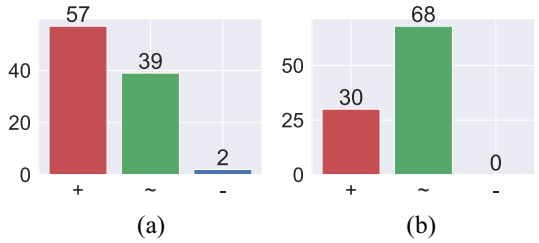


Fig. 13. Statistical comparison of best fitness values and R^2 scores using hash checking or not. ("+," "∼," and "-" indicate using hash-based checking is better than, similar to, or worse than not using hash-based checking.) (a) Best fitness values. (b) Test R^2 scores.

count the size of the AQ ($a/[\sqrt{1+b^2}]$) as 10 to follow the rule of SRBench.[3] In contrast, several methods in SRBench consider the AQ function as a node AQ with two variables a and b, thus considering its size as 3. In terms of training time, SHM-GP has a comparable training time to FEAT [44] and is slightly slower than Operon. However, both FEAT and Operon are implemented in C++, while SHM-GP is implemented in Python. The running time of SHM-GP still has a lot of room for improvement. To obtain a reliable conclusion, we perform a pairwise statistical comparison on test R^2 score using the Wilcoxon signed-rank test at a significance level of 0.05 with Bonferroni correction. The experimental results are presented in Fig. 11 and show that SHM-GP is significantly better than Operon in the test R^2 score and has similar performance to PS-Tree [62]. Considering that the model size obtained by the SHM operator is approximately one order of magnitude smaller than the PS-Tree, it is clear that the SHM operator is a successful bloat control method for GP-based feature construction methods.

VI. Further Analysis

A. Analysis of Hash-Based Checking

To investigate whether the hash-based checking strategy takes effect in SHM-GP, we plot a curve of the average phenotype entropy over the 30 independent runs in Fig. 12 on four example datasets. The phenotype entropy is defined as $-\sum_k p_k \ln p_k$, where p_k represents the ratio of trees with distinct semantics k in the population [63]. These four curves show that the phenotype entropy is increased by employing the hash-based checking strategy. For example, on the "OpenML_608" dataset, the entropy first drops to around 5 in the first few generations, then stays at this level. In comparison, when not using the hash-based checking strategy, entropy will decrease to lower than 4 at first, and then gradually increase to around 4.5. From the perspective of the whole evolution process, phenotype entropy when using the hash-based checking strategy develops always better than without using it. It is worth noting that the hash-based checking strategy is more useful when the evaluation budget is insufficient, as the phenotype entropy can gradually recover in the later stage of evolution. Fig. 13 presents a statistical comparison of results on best fitness values and test R^2 scores when limiting the evaluation budget to 20 generations. The results show that the hash-based checking strategy can significantly improve search effectiveness on 57 datasets, while only performing worse on 2 datasets. As for the test R^2 scores, the checking strategy can improve performance on 30 datasets and not degrade performance on any dataset. Based on these results, we can conclude that the hash-based checking strategy should be incorporated into the SHM operator to preserve population diversity and to get better search results.

B. Analysis of Simplification Strategies

This section investigates the impact on tree size made by the proposed simplification strategies under the condition of using the GSG operator. Comparing the test R^2 of eliminating either two simplification strategies, Table VII shows that the simplification strategies do not significantly change predictive performance on most datasets. This is reasonable because simplification strategies do not change the semantics of GP trees, and thus only have a minor impact on the evolution process.

[3]Sympy automatically converts the division operator to a power operator. Thus, the AQ becomes $a(\sqrt{1+b^2})^{-1}$. Additionally, it is worth noting that b^2 is counted as three nodes in Sympy because the power operator itself is counted as one node.

TABLE VII
STATISTICAL COMPARISON OF TEST R^2 WHEN REMOVING DIFFERENT SIMPLIFICATION STRATEGIES

	Equivalent Prune	No Ablation
Constant Prune	1(+)/94(∼)/3(-)	0(+)/95(∼)/3(-)
Equivalent Prune	—	2(+)/93(∼)/3(-)

TABLE VIII
STATISTICAL COMPARISON OF MODEL SIZES WHEN REMOVING DIFFERENT SIMPLIFICATION STRATEGIES

	Equivalent Prune	No Ablation
Constant Prune	21(+)/76(∼)/1(-)	1(+)/93(∼)/4(-)
Equivalent Prune	—	0(+)/77(∼)/21(-)

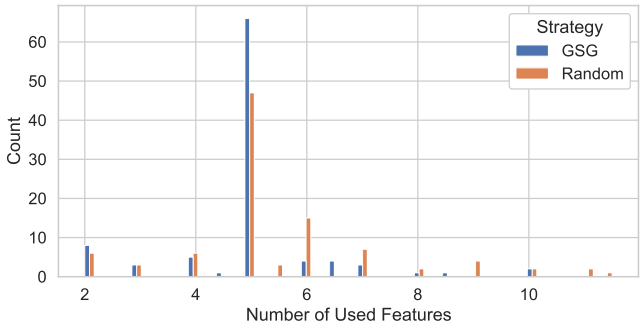


Fig. 14. Distribution for the number of selected features using GSG or not.

Further, we compare the average tree size by removing either one of the two prune strategies and present the significance comparison results in Table VIII. The results demonstrate that removing the constant prune strategy and the equivalent prune strategy will lead to worse results on 4 and 21 datasets, respectively. Thus, both strategies help the SHM operator reduce model size.

However, the SHM operator can significantly reduce model size on all datasets, and thus the main reason why the SHM operator outperforms other bloat control methods is not because of these two strategies. Due to the page limit, more experimental results supporting this claim are provided in the supplementary material.

C. Analysis of Guided Subtree Generation

In SHM-GP, the GSG operator is used to reduce the number of selected features to improve interpretability and predictive performance. In this section, we verify whether the GSG operator is helpful for SHM-GP. We present the distribution of the number of selected features in the final tree concerning using GSG or not in Fig. 14. Fig. 14 shows that the GSG operator can largely reduce the number of selected features. The average number of selected features before using the GSG operator is 5.5, whereas the number after using the GSG operator is 5. Therefore, the GSG operator is an effective method to simplify the number of used features in constructed features.

Further, since the irrelevant original features will have a low probability to be sampled in the GSG operator, the GSG operator may improve the predictive performance of the final

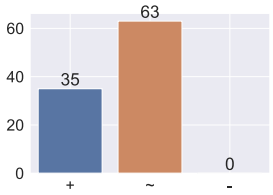


Fig. 15. Statistical comparison of test R^2 scores using GSG or not.

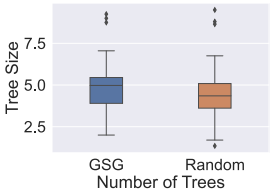


Fig. 16. Distribution of tree size with respect to using GSG or not.

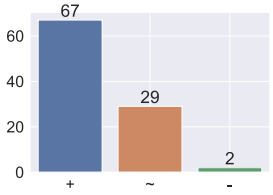


Fig. 17. Statistical comparison of test R^2 scores using GP with ten trees instead of using a single-tree GP.

model. To verify this, we plot the significance comparison for using the GSG operator or not in Fig. 15. Experimental results in Fig. 15 show that the GSG operator significantly improves the performance on 35 datasets, meanwhile having a similar performance on 63 datasets. These results indicate that the GSG operator not only reduces the number of selected features but also increases the predictive performance on unseen data. Moreover, experimental results in Fig. 16 show that the increase in predictive performance is not accompanied by a large increase in tree sizes, indicating that the SHM operator works well with the GSG operator.

D. Multitree GP Versus Single-Tree GP

In this article, we propose to use multitree GP instead of single-tree GP for evolutionary feature construction. In this section, we conduct experiments to verify the advantage of using multitree GP over single-tree GP. Fig. 17 presents the statistical comparison of test R^2 scores for using multitree GP or single-tree GP. The results show that using multitree GP can significantly improve the predictive performance of single-tree GP on 67 datasets, and only degrade the performance on 2 datasets. In fact, multitree GP can be viewed as multiple base learners in an ensemble learning model, and the LR model in SHM-GP can be viewed as a model combination technique. In the machine learning domain, there have been theories to prove the superiority of an ensemble of weak learners. Thus, it is not surprising that multitree GP can achieve better results than single-tree GP on test R^2 scores.

In addition to the test R^2 scores, tree size is another factor that needs to be considered. Fig. 18 presents the tree size using multitree GP and single-tree GP. For a fair comparison,

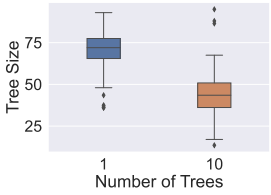


Fig. 18. Distribution of the sum of tree size with respect to different numbers of trees in GP.

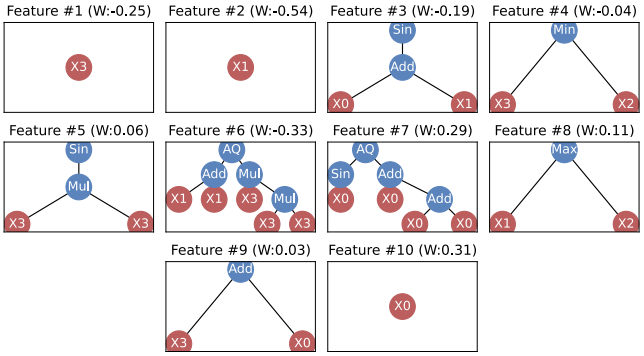


Fig. 19. Example of constructed features based on the SHM operator.

we present the sum of tree sizes instead of the mean of tree sizes in each individual. The experimental results show that although multitree GP has ten times the number of GP trees compared to single-tree GP, the sum of tree sizes is even smaller. As previously mentioned, multitree GP is similar to an ensemble learning model, which can make an accurate prediction by combining several weak GP trees. For tree-based GP, the random subtree crossover operator tends to generate a lot of tiny GP trees following a Lagrange distribution [64]. For single-tree GP, this is a problem because tiny GP trees are usually inferior to large GP trees, thus leading to an increase in the average tree size. In contrast, for multitree GP, a combination of tiny GP trees can still produce accurate prediction and thus mitigate bloat based on the crossover bias theory. In general, we can conclude that multitree GP is more appropriate than single-tree GP for evolutionary feature construction tasks from the perspective of predictive performance and tree size.

E. Example Models

The previous experimental results show that the SHM operator can significantly reduce tree size while maintaining the fitness value at the same level or even performing better. For a more intuitive understanding, we provide an illustrative example that demonstrates trees generated by SHM-GP on a galaxy visualization dataset [65] in Fig. 19. Additionally, two examples of trees generated by standard GP with limited tree depth are presented in Fig. 20. Due to their complexity, the complete examples of trees generated by standard GP with limited tree depth are included in the supplementary material. The weight of these features in the linear model is presented in parentheses. The first model achieves an R^2 score of 0.974, while the second model achieves an R^2 score of 0.971 on the test data, indicating similar accuracy between the two models. However, when comparing the complexity

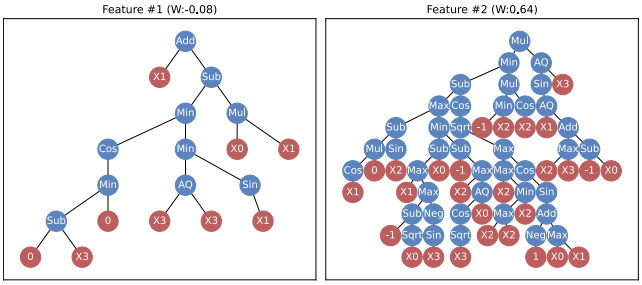


Fig. 20. Example of two out of ten features constructed using the depth-limiting method.

of the two models, it is clear that the constructed features in Fig. 19 are simpler than those features in Fig. 20. For example, Feature #2 in Fig. 20 is hard to read, while the largest feature in Fig. 19 only has nine nodes. It is worth noting that although judging the rationale of the final model needs domain knowledge, the obtained model shows great consistency with the model discovered by other evolutionary feature construction methods [44], which reports that features $\{X0, X1, X3\}$ and their interactions correspond to large weights in the final model.

VII. Conclusion

The goal of this article is to propose a genetic operator to control bloat for GP-based feature construction algorithms without sacrificing the predictive accuracy of the final model. This has been successfully achieved by proposing a new SHM operator that retains the most informative subtree during the evolution process while discarding other parts. Moreover, we have developed a hash-based checking strategy to prevent generating repetitive features in an individual, which increases population diversity.

Extensive experiments have been conducted to validate the effectiveness and efficiency of the SHM operator. The results on 98 datasets show that the proposed method not only significantly reduces tree size but also improves the test accuracy compared to the other seven bloat control methods. In addition, the comparison with 22 machine learning and symbolic regression algorithms on all 120 datasets in PMLB confirms the superiority of using the SHM operator in GP. Further analysis of mutation probability in the supplementary material shows that the proposed operator can be parameter-free. The ablation study on the hash-based checking strategy indicates that it is an effective way to increase population diversity. Therefore, the proposed bloat control method can significantly reduce tree size without sacrificing or even improving regression performance.

In the future, there are three promising research directions worth exploring. First, the generalization bound in this article only considers the hoist mutation operator in each generation. It would be better to prove that the generalization error is also bounded when combining the hoist mutation operator with crossover, mutation, and selection operators. Second, the SHM operator proposed in this article is designed for small-scale and medium-scale problems. In the future, it would be worthwhile to investigate whether the proposed operator

is also applicable to large-scale problems. One promising direction is to apply the proposed operator in GP for neural architecture search. However, defining the semantics in a neural architecture search scenario is an issue that still needs exploration. Finally, different features in each individual may share the same building blocks. In the future, it is worth investigating how to design a modular GP system to further reduce tree size.

REFERENCES

[1] B. Tran, B. Xue, and M. Zhang, "Genetic programming for multiple-feature construction on high-dimensional classification," *Pattern Recognit.*, vol. 93, pp. 404–417, Sep. 2019.

[2] H. Zhang, A. Zhou, and H. Zhang, "An evolutionary forest for regression," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 735–749, Aug. 2022.

[3] W. B. Langdon, "Size fair and homologous tree genetic programming crossovers," *Genet. Program. Evol. Mach.*, vol. 1, nos. 1–2, pp. 95–119, 2000.

[4] W. A. Tackett, "Recombination, selection, and the genetic construction of computer programs," Ph.D. dissertation, Dept. Comput. Sci., Univ. Southern California, Los Angeles, CA, USA, 1994.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, USA: Morgan Kaufmann Publ., Inc., 1998.

[6] T. Soule and J. A. Foster, "Removal bias: A new cause of code growth in tree based evolutionary programming," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1998, pp. 781–786.

[7] W. B. Langdon and R. Poli, "Fitness causes bloat: Mutation," in *Proc. 1st Eur. Conf. Genet. Program.*, 1998, pp. 37–48.

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

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

[10] Y. Mei, Q. Chen, A. Lensen, B. Xue, and M. Zhang, "Explainable artificial intelligence by genetic programming: A survey," *IEEE Trans. Evol. Comput.*, vol. 27, no. 3, pp. 621–641, Jun. 2023.

[11] S. Luke and L. Panait, "Lexicographic parsimony pressure," in *Proc. GECCO*, 2002, pp. 829–836.

[12] S. Luke and L. Panait, "Fighting bloat with nonparametric parsimony pressure," in *Proc. 7th Int. Conf. Parallel Probl. Solv. Nat.*, 2002, pp. 411–421.

[13] S. Silva and E. Costa, "Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories," *Genet. Program. Evol. Mach.*, vol. 10, no. 2, pp. 141–179, 2009.

[14] E. Alfaro-Cid, A. Esparcia-Alcázar, K. Sharman, and F. F. de Vega, "Prune and plant: A new bloat control method for genetic programming," in *Proc. HIS*, 2008, pp. 31–35.

[15] A. Ekart and S. Z. Nemeth, "Selection based on the Pareto nondomination criterion for controlling code growth in genetic programming," *Genet. Program. Evol. Mach.*, vol. 2, no. 1, pp. 61–73, 2001.

[16] K. Nag and N. R. Pal, "Feature extraction and selection for parsimonious classifiers with multiobjective genetic programming," *IEEE Trans. Evol. Comput.*, vol. 24, no. 3, pp. 454–466, Jun. 2020.

[17] M. Zhang and P. Wong, "Genetic programming for medical classification: A program simplification approach," *Genet. Program. Evol. Mach.*, vol. 9, no. 3, pp. 229–255, 2008.

[18] T. P. Pawlak, B. Wieloch, and K. Krawiec, "Semantic backpropagation for designing search operators in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 326–340, Jun. 2015.

[19] M. Virgolin, T. Alderliesten, and P. A. Bosman, "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression," in *Proc. GECCO*, 2019, pp. 1084–1092.

[20] Q. U. Nguyen and T. H. Chu, "Semantic approximation for reducing code bloat in genetic programming," *Swarm Evol. Comput.*, vol. 58, Nov. 2020, Art. no. 100729.

[21] Q. Chen, B. Xue, and M. Zhang, "Preserving population diversity based on transformed semantics in genetic programming for symbolic regression," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 433–447, Jun. 2021.

[22] E. Dolson, A. Lalejini, and C. Ofria, "Exploring genetic programming systems with MAP-elites," in *Genetic Programming Theory and Practice XVI*. Cham, Switzerland: Springer, 2019, pp. 1–16.

[23] B.-T. Zhang and H. Mühlenbein, "Balancing accuracy and parsimony in genetic programming," *Evol. Comput.*, vol. 3, no. 1, pp. 17–38, Mar. 1995.

[24] E. D. De Jong and J. B. Pollack, "Multi-objective methods for tree size control," *Genet. Program. Evolv. Mach.*, vol. 4, no. 3, pp. 211–233, 2003.

[25] M. Kommenda, G. Kronberger, M. Affenzeller, S. M. Winkler, and B. Burlacu, "Evolving simple symbolic regression models by multi-objective genetic programming," in *Genetic Programming Theory and Practice XIII*. Cham, Switzerland: Springer, 2016, pp. 1–19.

[26] D. Liu, M. Virgolin, T. Alderliesten, and P. A. N. Bosman, "Evolvability degeneration in multi-objective genetic programming for symbolic regression," in *Proc. GECCO*, 2022, pp. 973–981.

[27] S. Wang, Y. Mei, and M. Zhang, "A multi-objective genetic programming algorithm with α dominance and archive for uncertain capacitated arc routing problem," *IEEE Trans. Evol. Comput.*, early access, Jul. 29, 2022, doi: 10.1109/TEVC.2022.3195165.

[28] R. Poli, "A simple but theoretically-motivated method to control bloat in genetic programming," in *Proc. 6th Eur. Conf. Genet. Program.*, 2003, pp. 204–217.

[29] A. de Lima, S. Carvalho, D. M. Dias, E. Naredo, J. P. Sullivan, and C. Ryan, "Lexi2: Lexicase selection with lexicographic parsimony pressure," in *Proc. GECCO*, 2022, pp. 929–937.

[30] S. Silva, S. Dignum, and L. Vanneschi, "Operator equalisation for bloat free genetic programming and a survey of bloat control methods," *Genet. Program. Evol. Mach.*, vol. 13, no. 2, pp. 197–238, 2012.

[31] K. E. Kinnear, "Evolving a sort: Lessons in genetic programming," in *Proc. Proc. IEEE Int. Conf. Neural Netw.*, 1993, pp. 881–888.

[32] N. Javed, F. Gobet, and P. Lane, "Simplification of genetic programs: A literature survey," *Data Min. Knowl. Discov.*, vol. 36, no. 4, pp. 1279–1300, 2022.

[33] A. Song, D. Chen, and M. Zhang, "Contribution based bloat control in genetic programming," in *Proc. IEEE Congr. Evol. Comput.*, 2010, pp. 1–8.

[34] P. Wong and M. Zhang, "Algebraic simplification of GP programs during evolution," in *Proc. GECCO*, 2006, pp. 927–934.

[35] D. Kinzett, M. Johnston, and M. Zhang, "Numerical simplification for bloat control and analysis of building blocks in genetic programming," *Evol. Intell.*, vol. 2, no. 4, pp. 151–168, 2009.

[36] L. Vanneschi, M. Castelli, and S. Silva, "A survey of semantic methods in genetic programming," *Genet. Program. Evol. Mach.*, vol. 15, pp. 195–214, Jan. 2014.

[37] M. A. Haeri, M. M. Ebadzadeh, and G. Folino, "Statistical genetic programming for symbolic regression," *Appl. Soft Comput.*, vol. 60, pp. 447–469, Nov. 2017.

[38] M. Castelli, S. Silva, and L. Vanneschi, "A C++ framework for geometric semantic genetic programming," *Genet. Program. Evolvable Mach.*, vol. 16, pp. 73–81, Mar. 2015.

[39] J. F. B. Martins, L. O. V. Oliveira, L. F. Miranda, F. Casadei, and G. L. Pappa, "Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming," in *Proc. GECCO*, 2018, pp. 1151–1158.

[40] Q. Chen, B. Xue, and M. Zhang, "Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 488–502, Jun. 2019.

[41] W. La Cava, T. Helmuth, L. Spector, and J. H. Moore, "A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection," *Evol. Comput.*, vol. 27, no. 3, pp. 377–402, 2019.

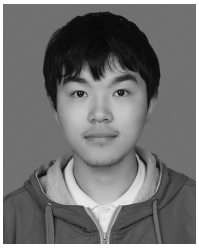
[42] K. Nickerson, A. Kolokolova, and T. Hu, "Creating diverse ensembles for classification with genetic programming and neuro-MAP-elites," in *Proc. 25th Eur. Conf. Genet. Program.*, 2022, pp. 212–227.

[43] H. Zhang, A. Zhou, Q. Chen, B. Xue, and M. Zhang, "SR-forest: A genetic programming based heterogeneous ensemble learning method," *IEEE Trans. Evol. Comput.*, early access, Feb. 7, 2023, doi: 10.1109/TEVC.2023.3243172.

[44] W. La Cava, T. R. Singh, J. Taggart, S. Suri, and J. H. Moore, "Learning concise representations for regression by evolving networks of trees," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–16. [Online]. Available: https://openreview.net/forum?id=Hke-JhA9Y7

[45] J. Ma and X. Gao, "A filter-based feature construction and feature selection approach for classification using genetic programming," *Knowl.-Based Syst.*, vol. 196, May 2020, Art. no. 105806.

[46] A. Lensen, B. Xue, and M. Zhang, "Genetic programming for evolving similarity functions for clustering: Representations and analysis," *Evol. Comput.*, vol. 28, no. 4, pp. 531–561, 2020.
[47] M. Muharram and G. D. Smith, "Evolutionary constructive induction," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1518–1528, Nov. 2005.
[48] I. Arnaldo, U.-M. O'Reilly, and K. Veeramachaneni, "Building predictive models via feature synthesis," in *Proc. GECCO*, 2015, pp. 983–990.
[49] Q. Chen, M. Zhang, and B. Xue, "Genetic programming with embedded feature construction for high-dimensional symbolic regression," in *Proc. 20th Asia–Pacific Symp. Intell. Evol. Syst.*, 2017, pp. 87–102.
[50] C. A. Owen, G. Dick, and P. A. Whigham, "Standardization and data augmentation in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1596–1608, Dec. 2022.
[51] Q. Fan, Y. Bi, B. Xue, and M. Zhang, "Genetic programming for image classification: A new program representation with flexible feature reuse," *IEEE Trans. Evol. Comput.*, vol. 27, no. 3, pp. 460–474, Jun. 2023.
[52] A. Song, D. Chen, and M. Zhang, "Bloat control in genetic programming by evaluating contribution of nodes," in *Proc. GECCO*, 2009, pp. 1893–1894.
[53] Q. Chen, M. Zhang, and B. Xue, "Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 703–717, Aug. 2019.
[54] V. Cherkassky and F. M. Mulier, *Learning from Data: Concepts, Theory, and Methods*. Hoboken, NJ, USA: Wiley, 2007.
[55] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "PMLB: A large benchmark suite for machine learning evaluation and comparison," *BioData Min.*, vol. 10, no. 1, pp. 1–13, 2017.
[56] W. L. Cava et al., "Contemporary symbolic regression methods and their relative performance," in *Proc. NeurIPS Datasets Benchmarks*, 2021, pp. 1–16. [Online]. Available: https://openreview.net/forum?id=xVQMrDLyGst
[57] C. A. Owen, G. Dick, and P. A. Whigham, "Standardisation and data augmentation in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1596–1608, Dec. 2022.
[58] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[59] T. H. Chu, Q. U. Nguyen, and M. O'Neill, "Semantic tournament selection for genetic programming based on statistical analysis of error vectors," *Inf. Sci.*, vol. 436–437, pp. 352–366, Apr. 2018.
[60] J. Ni, R. H. Drieberg, and P. I. Rockett, "The use of an analytic quotient operator in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 146–152, Feb. 2013.
[61] B. Burlacu, G. Kronberger, and M. Kommenda, "Operon C++ an efficient genetic programming framework for symbolic regression," in *Proc. GECCO*, 2020, pp. 1562–1570.
[62] H. Zhang, A. Zhou, H. Qian, and H. Zhang, "PS-tree: A piecewise symbolic regression tree," *Swarm Evol. Comput.*, vol. 71, Jun. 2022, Art. no. 101061.
[63] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 47–62, Feb. 2004.
[64] S. Dignum and R. Poli, "Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat," in *Proc. GECCO*, 2007, pp. 1588–1595.
[65] W. S. Cleveland, *Visualizing Data*. Thousand Oaks, CA, USA: Hobart Press, 1993.



Hengzhe Zhang (Member, IEEE) received the B.Sc. degree in software engineering from Xiangtan University, Xiangtan, Hunan, China, in 2019, and the M.Sc. degree in computer science from East China Normal University, Shanghai, China, in 2022. He is currently pursuing the Ph.D. degree in computer science with the Victoria University of Wellington, Wellington, New Zealand.

His current research interests include symbolic regression, genetic programming, evolution computation, and statistical machine learning.

Qi Chen (Member, IEEE) received the B.E. degree in automation from the University of South China, Hengyang, Hunan, China, in 2005, the M.E. degree in software engineering from the Beijing Institute of Technology, Beijing, China, in 2007, and the Ph.D. degree in computer science from the Victoria University of Wellington (VUW), Wellington, New Zealand, in 2018.

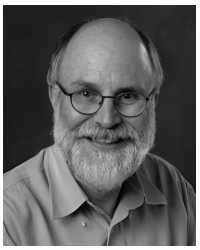
She is currently a Senior Lecturer of Artificial Intelligence with the School of Engineering and Computer Science, VUW. Her research interests include machine learning, evolutionary computation, feature selection, feature construction, transfer learning, domain adaptation, and statistical learning theory.

Dr. Chen serves as a Reviewer for international conferences, including AAAI and IJCAI, and international journals, including IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and IEEE TRANSACTIONS ON CYBERNETICS.

Bing Xue (Senior Member, IEEE) received the Ph.D. degree in computer science from the Victoria University of Wellington, Wellington, New Zealand, in 2014.

She is currently a Professor of Artificial Intelligence, the Deputy Director of the Centre for Data Science and Artificial Intelligence, and the Deputy Head of School with the School of Engineering and Computer Science, Victoria University of Wellington. She has over 300 papers published in fully refereed international journals and conferences and her research focuses mainly on evolutionary computation and machine learning.

Dr. Xue is currently the Chair of IEEE CIS Evolutionary Computation Technical Committee and an Editor of IEEE CIS Newsletter. She has also served as an Associate Editor for several international journals, such as *IEEE Computational Intelligence Magazine*, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, and *ACM Transactions on Evolutionary Learning and Optimization*. She is also a Fellow of Engineering New Zealand.



Wolfgang Banzhaf (Member, IEEE) received the Dr.rer.nat (Ph.D.) degree from the Department of Physics, Technische Hochschule Karlsruhe (currently, Karlsruhe Institute of Technology), Karlsruhe, Germany, in 1985.

He was the University Research Professor with the Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada, where he served as the Head of Department from 2003 to 2009 and from 2012 to 2016. He is the John R. Koza Chair of Genetic Programming with the Department of Computer Science and Engineering and a member of the BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, MI, USA. He is also interested in self-organization studies and in artificial life field. He has become more involved with network research as it applies to natural and man-made systems. His research interests are in the field of bioinspired computing, notably evolutionary computation, and complex adaptive systems.

Mengjie Zhang (Fellow, IEEE) received the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2000.

He is currently a Professor of Computer Science and the Director of the Centre for Data Science and Artificial Intelligence, Victoria University of Wellington, Wellington, New Zealand. He has published over 800 research papers in refereed international journals and conferences. His current research interests include genetic programming, image analysis, feature selection and reduction, job-shop scheduling, and evolutionary deep learning and transfer learning.

Prof. Zhang is a Fellow of the Royal Society of New Zealand and Engineering New Zealand, and an IEEE Distinguished Lecturer.