# Chapter 8
# Sharpness-Aware Minimization in Genetic Programming


**Illya Bakurov, Nathan Haut, and Wolfgang Banzhaf**

**Abstract** Sharpness-Aware Minimization (SAM) was recently introduced as a regularization procedure for training deep neural networks. It simultaneously minimizes the fitness (or loss) function and fitness sharpness. The latter serves as a measure of the nonlinear behavior of a solution guiding toward solutions that lie in neighborhoods with uniformly similar loss values across all fitness cases. In this contribution we adapt SAM for tree Genetic Programming (TGP) by exploring the semantic neighborhoods of solutions using two simple approaches. By perturbing input and output of program trees, sharpness can be estimated and used as a second optimization criterion during the evolution. To better understand the impact of this variant of SAM on TGP, we collect numerous indicators of the evolutionary process, including generalization ability, complexity, diversity, and a recently proposed genotype-phenotype mapping to study the amount of redundancy in trees. The experimental results demonstrate that using any of the two proposed SAM adaptations in TGP allows (i) a significant reduction of tree sizes in the population and (ii) a decrease in redundancy of the trees. When assessed on real-world benchmarks, the generalization ability of solutions does not deteriorate.

## 8.1 Introduction

The automatic discovery of mathematical expressions to describe phenomena captured in data is an extremely valuable tool for accelerating scientific discovery since the mathematical expressions can be used to make predictions about the systems that

I. Bakurov (✉) · W. Banzhaf
Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA
e-mail: bakurov1@msu.edu

W. Banzhaf
e-mail: banzhaf@msu.edu

N. Haut
Department of Computational Mathematics, Science, and Engineering, Michigan State University, East Lansing, MI, USA
e-mail: hautnath@msu.edu



S. M. Winkler et al. (eds.), *Genetic Programming Theory and Practice XXI*, Genetic and Evolutionary Computation, https://doi.org/10.1007/978-981-96-0077-9_8

generated the data and the expressions can be directly studied to extract new insights into the system. There are many approaches for finding equations that fit data: linear regression, polynomial regression, SINDy [7], neural-symbolic regression [6], symbolic regression [19], etc. Genetic programming (GP) is a popular method for finding equations that fit data since it allows great flexibility for the discovery of nonlinear behaviors in data while also being effective in small data scenarios, unlike deep learning (DL) approaches which generally require large training datasets. This ability of GP to be effective in small data scenarios is likely related to an evolutionary algorithm's bias for simple solutions, as simple solutions are less likely to overfit [5]. Even so, in small data scenarios, the models are naturally underconstrained in the interstitial spaces between the training data points, which means that surprising and unexpected behavior can occur when interpolating. Ideally, we would want the models to be at least stable (smooth) when interpolating, otherwise the generalization ability of a model can be severely reduced.

Some GP methods have been proposed to help stabilize model predictions to improve the robustness against overfitting in small data scenarios, such as order of nonlinearity [38], model curvature [35], random sampling technique (RST) [13], RelaxGP [8], and overfit repulsors [36]. Order of nonlinearity and model curvature are approaches that attempt to consider model properties to predict overfitting [35, 38]. Random sampling attempts to reduce the risk of overfitting by ensuring that no model sees the whole dataset in a single generation [13]. RelaxGP minimizes the risk of overfitting by assigning no additional fitness for getting closer than some threshold to the response data [8]. The use of overfit repulsors stores known overfit models and promotes the evolution of individuals that are different in semantic space [36].

Here we propose to use model properties to predict and discourage overfitting while using synthetic data to catch models that are unstable in the interstitial spaces around the known training data. This approach is inspired by the Sharpness-Aware Minimization (SAM) technique recently developed in DL for neural networks [10] and also applied in a different way in [40] for feature construction. In [10], the authors demonstrate that SAM can promote generalization in neural networks. This is achieved by searching for neural network model parameters that lie in neighborhoods of low sharpness.

We introduce two new methods for selecting against sharpness in genetic programming: input-based SAM (SAM-In) and output-based SAM (SAM-Out). In this work, we aim to:

- Adapt sharpness-aware minimization from a deep learning regularization method to be compatible with genetic programming in an efficient way;
- Reduce the risk of overfitting in GP by rewarding the smoothness of models rather than just the accuracy;
- Improve the stability of models by penalizing models that vary significantly in their response surfaces in neighborhoods in the fitness landscape.

## 8.2 Related Work

### *8.2.1 Sharpness-Aware Minimization in Deep Learning*

DL models commonly used in computer vision are often comprised of hundreds of megabytes and millions of parameters, whereas large language models are made of billions of parameters and their size is measured in gigabytes. Training such complex systems is a big challenge for deep learning, but the high demand for computational resources and data availability are not the only constraints. DL models are particularly prone to memorizing the training data rather than generalizing the learned patterns to unseen data (i.e., overfitting). Several techniques are commonly used to mitigate overfitting in DL and foster convergence: penalizing loss/fitness functions for disproportional high weights (such as L1 and L2 penalties), setting the weights connecting randomly selected neurons in some layer(s) to zero during a given training iteration, usage of varied data augmentation techniques to expose models to more variation, normalization of data batches that pass through the network, etc. Some DL researchers focus their attention on how a variety of neural architecture choices (such as depth, width, network architecture, optimizer selection, connectivity patterns, and batch size) affect the geometry of fitness landscapes and relate this to network generalization. For instance, Li et al. concluded [21] that landscape geometry significantly affects the generalization ability of a system and that larger network depth, batch size, and usage of sequential connections without shortcut connections produce sharper fitness landscapes and, consequently, negatively impact the generalization ability of networks.

Motivated by the connection between the fitness landscape and generalization, P. Foret et al. proposed a novel procedure that improves model generalization by simultaneously minimizing the loss value and sharpness of the loss landscape by promoting parameters that lie in neighborhoods having uniformly low loss value [10]. Specifically, the authors add a regularization term to the loss function that is a measure of how training loss can be increased by *moving* weights $W$ to a *nearby* parameter value $W + \varepsilon$, where $\varepsilon$ represents the perturbation. The authors demonstrate that SAM improves model generalization ability across a range of widely studied computer vision tasks and provides robustness to label noise on par with that provided by SOTA procedures that specifically target learning with noisy labels. This procedure was called Sharpness-Aware Minimization (SAM) and constitutes the main inspiration for the approach proposed in this manuscript.

### *8.2.2 Semantic Awareness in Genetic Programming*

Traditional crossover and mutation operators used in GP rely on structural (i.e., genotypic) transformations of parent individuals, without knowledge of their effects

on the offspring's behavior. But a mere replacement of one node can lead to arbitrarily large changes in the behavior of a program (e.g., replacing—with $*$ in $x_1 + x_2 - x_3$).

Since 2012 GP researchers have incorporated semantic awareness into GP.[1] Moraglio et al. [24] proposed geometric semantic operators (GSOs) that allow to transform genotypes of parents in such a way that the effect on the semantics of the offspring is known, unlike what happens with standard GP operators [18]. A remarkable characteristic of GSOs is that the error surface on the training data for any supervised machine learning problem is unimodal and therefore easy to search. This holds independently of the size and complexity of the dataset [34]. GSOs allowed GP to achieve better accuracy in numerous real-world applications, even when compared with other ML approaches [2, 20, 37].

Vanneschi et al. [37] demonstrated that using a sigmoid-bounding function in geometric semantic mutation (GSM) helps to stabilize the learning process and increase the generalization ability of solutions. In this sense, the semantics of the offspring resulting from a GSM will surround the semantics of the parent within a user-controlled range $[-ms,\ ms]$, which can be interpreted as a step in any direction of the semantic space in a box of side $ms$. Recently, Bakurov et al. [3] proposed a normalization procedure for GSM that reduces the size of the programs and overcomes some saturation issues associated with the sigmoid function adopted in [37], which resulted in more accurate and simple solutions. This is the GSM variant used in our study. Gonçalves et al. introduced the concept of the semantic neighborhood—the set of neighbors reachable from a given solution when a GSM is applied to it—and proposed a stopping criteria based on neighborhood properties [12]. Competitive generalization was achieved within significantly fewer generations, and consequently in smaller solutions. This suggests that information collected from the semantic neighborhood can bring notable value for the evolutionary process.

### *8.2.3 Noisy Data and Fitness Functions*

A common problem in machine learning is overfitting, where the ML method focuses too much on chasing perfect accuracy and the model only learns to memorize the training data but fails to identify generalizable patterns. One method to avoid overfitting is to utilize a noisy fitness function, which is a fitness function that has noise introduced so that it is not exact and thus prevents the machine learning algorithm from chasing a stationary zero error solution in the search space. In evolutionary computation it has been shown that noisy fitness evaluations can be beneficial in rugged landscapes by allowing the search to more easily escape local optima and navigate toward a global optimum [31]. Also, evolutionary computation methods have been

[1] The term *semantics* defines the vector of output values of a candidate solution (program), calculated on the training observations [24]. Following this notion, a candidate solution in GP is a point in a multidimensional *semantic space*, where the dimensionality is equal to the number of observations in the training set.

found to be robust in "uncertain environments" where noise is naturally present in the environment [16, 29]. While somewhat different from a noisy fitness function where noise is intentionally injected into the fitness function, it demonstrates that noise is not necessarily a problem for evolutionary computation and may sometimes even be a benefit.

In GP, a similar approach has been to find a "good-enough" fit such that the error is within some threshold rather than trying to find a perfect fit with zero error. One such approach is RelaxGP [8], where the authors change the fitness to instead consider an upper and lower bound for each response value rather than trying to perfectly fit each response value. In GP, random sampling has also been shown to have similar benefits to noisy fitness evaluations since it improves the generalization of model and discourages memorization of the whole training set [13, 26].

### *8.2.4 SAM in GP*

Very recently, sharpness was also incorporated in GP [40] within the scope of feature construction. In that work, sharpness was measured by introducing Gaussian noise to each node at all layers of the feature trees. The sharpness of an individual was then determined using the difference in loss after the noise was introduced. The results demonstrate that sharpness can indeed be used to reduce the size of GP trees and mitigate their tendency to overfit. However, one potential drawback of this approach is that it is computationally expensive since it computes sharpness using noise at each layer of the trees.

## 8.3 The Proposed Approach

Here we evaluate two approaches for estimating the sharpness of a given tree in GP. Because of the fundamental difference between artificial neural networks (ANN) and TGP, a direct transfer of SAM from the former to the latter is impracticable. Typically, ANNs have a predefined (fixed) architecture (e.g., number of layers and nodes per layer) and the search for an optimal solution consists of finding an optimal set of weights (a collection of real values) that minimizes the loss; therefore, for ANNs, the loss landscape is continuous.

Although there exist some successful attempts to include learnable weights in TGP [27, 28, 32], TGP typically is comprised of discrete structures (program elements), commonly divided into two groups: (i) terminals, which represent the input features of the problem and numeric constant values, and (ii) functions, which represent the operations to be performed on terminals. The search for an optimal solution is thus conducted in a discrete space made up of combinations of these tree data structures, that can grow or shrink dynamically in size. Given these differences, we use two approaches to adapt SAM for TGP. The first acts on the input of a tree by

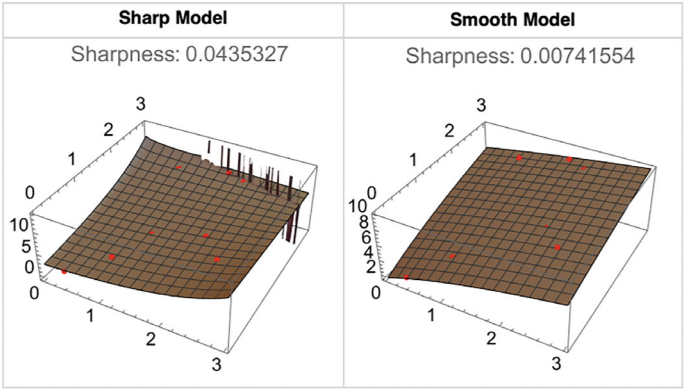


**Fig. 8.1** A sharp and a smooth model are compared along with the values that were returned by the SAM-IN metric, showing that the sharper model is clearly identified by the metric

randomly perturbing both constants and input features of the model; in other words, it adds noise to a tree's terminal nodes. The second approach consists of randomly perturbing the output of a tree. The subsections below explain each approach in more detail.

The goal of these methods is to identify and penalize those models of the population that exhibit strongly nonlinear behavior in the fitness landscape (sharpness). An example of the behavioral difference between two models is shown in Fig. 8.1, where we depict two models with similar accuracy on the training data. However, one is not stable in some regions of the fitness landscape, while the other is very smooth and stable. Training points are shown as red dots on the model response surfaces and sharpness, calculated by our metric, is listed for each model in the labels above the plots. The values show that the sharpness metric would be able to identify this unstable model and remove it from a population. Unknowingly selecting and using a sharper model would be detrimental since it has very unstable behavior in certain regions of input space.

### *8.3.1 SAM on Input (SAM-IN)*

Our first approach, called SAM-IN, quantifies the sensitivity of each tree in the population with respect to random perturbations in the terminals, both in variables and constants. This sensitivity is calculated by measuring the absolute fitness differential after applying random noise of magnitude $\varepsilon$ to a subset of input training data instances $n$ and all of the model constants. The noise introduced to the variables is a function of the standard deviation of the data in that variable. Then, the trees are executed a second time in order to obtain the fitness on the perturbed training cases. This quantity is then used as a second selection criterion during an evolutionary run. Selection is

performed by using randomized double tournament selection, where the order of objectives is randomized for every selection. We expect that high sensitivity toward input noise indicates that a given tree is likely overfit and thus should have a lower selection preference when compared to a tree with a smaller sensitivity. Algorithm 8.1 is an implementation of SAM-IN. Mathematically, the SAM-IN metric is represented by Eq. (8.1), where $n$ is the number of perturbations (in this work we just use just one perturbation), $x$ is the input data, $y$ is the response data, $\varepsilon$ is the noise introduced to the input data, $F$ is the model, $\tilde{F}$ is the model with perturbed constants, and Corr is a function for computing Pearson's correlation coefficient depicted in Eq. (8.2). In Eq. (8.2), $y$ is the target response, $\hat{y}$ is the model response, and $N$ is the number of data points.

$$S_{in} = \frac{\sum_{i=1}^{n} |\text{Corr}(F(x), y)^2 - \text{Corr}(\tilde{F}_i(x + \varepsilon_i), y)^2|}{n} \tag{8.1}$$

$$Corr(\hat{y}, y) = \frac{\sum_{i=1}^{N}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2 \times \sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}})^2}} \tag{8.2}$$

**Algorithm 8.1** SAM on Input *Revision in progress*

**Require:** perturbations number $n > 0$, perturbation magnitude $\varepsilon > 0$, training dataset $D$
$\overrightarrow{\sigma} \leftarrow$ Compute the standard deviation of each feature in $X$
**for** each generation in the evolutionary run **do**
  $X_s \leftarrow$ Select a random sample of size $n$ from $X$
  $X_{s+\varepsilon} \leftarrow$ Make a copy of $X_s$
  **for** $i = 1$ to LENGTH($X_{s+\varepsilon}$) **do**
    $\varepsilon_i \leftarrow$ generate a random value in $\left[-\varepsilon\overrightarrow{\sigma},\ \varepsilon\overrightarrow{\sigma}\right]$
    $X_{s+\varepsilon}[i] = X_{s+\varepsilon}[i] + \varepsilon_i$
  **end for**
  **for** each tree $t$ in the population **do**
    $t_\varepsilon \leftarrow$ make a copy of $t$
    **for** each constant $c$ in $t_\varepsilon$ **do**
      $\varepsilon_c \leftarrow$ generate a random value in $[-\varepsilon,\ \varepsilon]$
      $c = c + \varepsilon_c$
    **end for**
    $f^t \leftarrow$ compute the fitness of $t$ on $X_s$
    $f^{t+\varepsilon} \leftarrow$ compute the fitness of $t_\varepsilon$ on $X_{s+\varepsilon}$
    $S^t_{SAM-IN} \leftarrow$ compute sharpness of $t$ as $|f_t - f_{t+\varepsilon}|$
  **end for**
**end for**

Unlike the original SAM in DL, or the SAM variant introduced in [40], SAM-IN introduces noise in the terminals rather than all of the connections (red arrows in Fig. 8.2). This is computationally less expensive than the SAM-GP introduced in [40] and better reflects the idea behind generalization ability: the ability to provide accurate and stable predictions for new, previously unseen data, drawn from the same

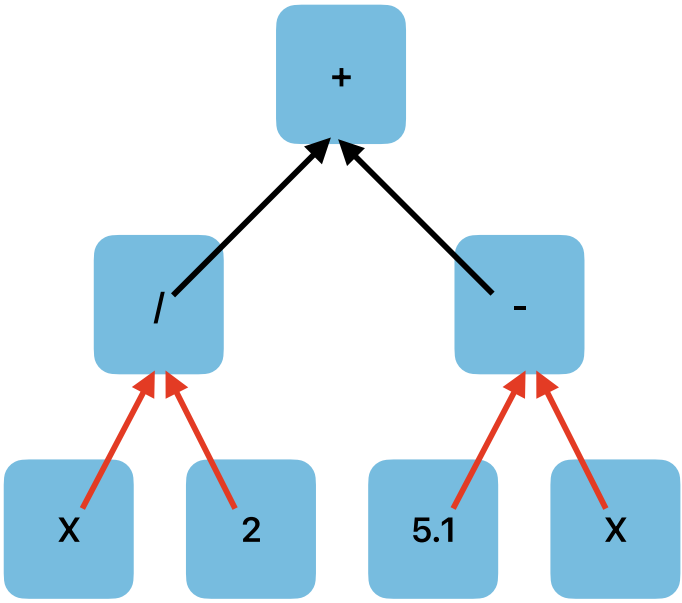


**Fig. 8.2** An example of a GP tree. The red arrows indicate the location of injected noise

distribution as the one used for training. The small noise will propagate through all layers of a tree and helps to measure the potential for accurate and stable predictions.

### *8.3.2 SAM on Output (SAM-OUT)*

Our second approach, called SAM-OUT, builds upon the mechanics of the GSM operator and the notion of semantic neighborhood introduced by [11] and described in Sect. 8.2.2. Given a reference individual (tree), it generates $n$ semantic neighbors using GSM with $ms = \varepsilon$, where $n$ and $\varepsilon$ are hyper-parameters of SAM-OUT. The fitness of each semantic neighbor is calculated and the sharpness of the reference individual is estimated as the variance in the semantic neighborhood. A large variance/instability in the semantic neighborhood indicates that a given reference individual is likely to overfit and, thus, should have a lower selection preference when compared to a tree with a smaller variance. Algorithm 8.2 contains an implementation of SAM-OUT. We use the normalized-GSM as proposed in [3] given its notable stability and simplicity. Mathematically, SAM-OUT is represented in Eq. (8.3), where $x$ represents input data, $y$ the response data, $\varepsilon$ is the noise introduced to the model output, $F$ is the model, and Corr is a function for computing Pearson's correlation coefficient shown in Eq. (8.2).

$$S_{out} = \frac{\sum_{i=1}^{n} |Corr(F(x), y)^2 - Corr(F(x) + \varepsilon_i, y)^2|}{n} \tag{8.3}$$

Note that we do not need to construct and execute a random tree to sample from the semantic neighborhood of a reference individual, as the semantic neighbors are only used for sharpness assessment. Because semantic neighbors will not be used anywhere else in the process, SAM-OUT (unlike SAM-IN) can fully operate in semantic space, which renders it significantly more efficient. Thus one can generate a semantic neighbor by summing the semantics of the reference individual with a randomly generated vector of values centered at zero and bounded in $[-\varepsilon,\ \varepsilon]$. It is important to note that there is no need for additional execution of the reference individual to extract its semantics, as it is already done when performing fitness evaluation.

**Algorithm 8.2** SAM on Output

**Require:** perturbations number $n > 0$, perturbation magnitude $\epsilon > 0$, training dataset $X$

**for** each generation in the evolutionary run **do**
  **for** each tree $t$ in the population **do**
    $\widehat{y}_t \leftarrow$ Compute the output of the tree $t$ on $X$
    $\sigma_{\widehat{y}_t} \leftarrow$ Compute the standard deviation of $\widehat{y}_t$
    $\overrightarrow{f_n^t} \leftarrow$ empty vector for storing fitness of $n$ semantic neighbors of $t$
    **for** each perturbation $i$ in $n$ **do**
      $\widehat{y}^i_{t+\epsilon} \leftarrow$ Make a copy of $\widehat{y}_t$
      **for** $j = 1$ to LENGTH($\widehat{y}^i_{t+\epsilon}$) **do**
        $\epsilon^t_{(j,i)} \leftarrow$ generate a random vector in $\left[-\epsilon\sigma_{\widehat{y}_t},\ \epsilon\sigma_{\widehat{y}_t}\right]$
        $\widehat{y}^i_{t+\epsilon}[j] = \widehat{y}^i_{t+\epsilon}[j] + \epsilon^t_{(j,i)}$
      **end for**
      $\overrightarrow{f_n^t}[i] \leftarrow$ compute the fitness of the $i^{th}$ semantic neighbor $\widehat{y}^i_{t+\epsilon}$
    **end for**
  **end for**
  $S^t_{SAM-OUT} \leftarrow$ compute sharpness of $t$ as $\sigma^2(\overrightarrow{f_n^t})$
**end for**

A subtlety arises when using $R^2$ as fitness function as opposed to RMSE fitness. In this case SAM-OUT is not functionally identical to introducing noise to the target vector, as would occur when using noisy fitness functions. This is shown in Table 8.1 where the same error with RMSE is observed when adding the noisy vector, $\varepsilon$, to the predicted target, $\hat{y}$, as when subtracting $\varepsilon$ from the target $y$. This is, however, not the case when employing $R^2$ as fitness function, which means that adding noise to the output is different here from adding noise to the target.

**Table 8.1** Comparison of SAM-OUT versus Noisy target for RMSE and $R^2$. $y$ represents the target variable, $\hat{y}$ represents the model predicted target, and $\varepsilon$ represents a normally distributed noisy vector. $\varepsilon$ was randomly initialized and then used in the same state for each evaluation in the table

| Noise location | RMSE value | $R^2$ value |
|---|---|---|
| $y + \varepsilon$ | 2.931 | 0.431 |
| $y - \varepsilon$ | **2.831** | 0.294 |
| $\hat{y} + \varepsilon$ | **2.831** | 0.817 |
| No noise | 1.957 | 0.847 |

**Table 8.2** Details about the four real-world datasets used

| Dataset | #Instances | #Features |
|---|---|---|
| Boston [14] | 506 | 13 |
| Heating [33] | 768 | 8 |
| Diabetes [9] | 442 | 10 |
| Concrete [39] | 1005 | 8 |

## 8.4 Experimental Settings

We assess our method on data from four real-world regression problems and four popular synthetic functions. The real-world problems are described in Table 8.2. As for the latter, given a synthetic function $f(X)$, we randomly sample 100 data points in a 2-dimensional grid $(x)$, under uniform distribution, where each dimension corresponds to an input feature ($x_1$ and $x_2$). The third dimension corresponds to the target of the prediction $y$ and is obtained as $f(x) = y$. The synthetic problems are described in Table 8.3. In each run, we use different train-test partitions of the data. For real-world problems, we randomly sample 70% observations for training, while the remaining 30% are held out for testing. For synthetically generated problems, we use 50% for training and 50% for testing.

Table 8.4 lists the hyper-parameters (HPs) used in this study, along with cross-validation settings. The HPs were selected following common practices found across the literature to avoid a computationally demanding tuning phase. $R^2$ was used as fitness function [15, 17, 22] as it was found to converge faster, generalize better, even when only a few data points are available. Programs allowing invalid operations (like dividing by 0 or taking a square of a negative value) were automatically assigned a low fitness value. Although operator protection ensures a valid numerical output for any input, it can also produce unexpectedly high/low (i.e., sharp) predictions as shown in [17]. This would inevitably bias our sharpness-aware approach to avoid programs containing such operators. Thus, to compare sharpness-aware minimization and standard GP on a fair basis, none of the operators is protected in this study. Two rounds of tournament selection are used to select individuals, with sizes 6 and

**Table 8.3** Details about the four synthetic datasets used. In our experiments, $D = 2$

| Name | Function | $x_i \in$ |
|---|---|---|
| Levy | $f(\mathbf{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{D-1}(w_i - 1)^2 \left(1 + 10\sin^2(\pi w_i + 1)\right) + (w_d - 1)^2(1 + \sin^2(2\pi w_d))$, $w_i = 1 + \frac{x_i - 1}{4}$ | $[-10, 10]$ |
| Ackley | $f(\mathbf{x}) = -a\ \exp\left(-b\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(c\ x_i)\right) + a + \exp(1), a = 20, b = 0.2, c = 2\pi$ | $[-32.768, 32.768]$ |
| Rastrigin | $f(\mathbf{x}) = 10D + \sum_{i=1}^{D}\left(x_i^2 - 10cos(2pi\, x_i)\right)$ | $[-5.12, 5.12]$ |
| Rosenbrock | $f(\mathbf{x}) = \sum_{i=1}^{D-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ | $[-2.048, 2.048]$ |

**Table 8.4** Summary of the hyper-parameters. Note that $P(C)$ and $P(M)$ indicate the crossover and the mutation probabilities, respectively

| Parameters | Values |
|---|---|
| Nº Train/test split | Real-world problems: 70/30%; synthetic problems: 50–50% |
| Cross-validation | Monte-Carlo (repeated random subsampling) |
| Nº runs | 60 |
| Nº generations | 50 |
| Population's size | 100 |
| Functions ($F$) | {+, -, x, / sin(x), cos(x), tanh(x), $x^2$, $x^{-1}$, $\sqrt{x}$, $e^x$, $log(x)$} |
| Initialization | Ramped Half&Half (RHH) with max depth of 5 |
| Selection | double tournament with sizes 6 and 3, respectively |
| Genetic operators | {swap crossover, subtree mutation} |
| $P(C)$ | 0.8 |
| $P(M)$ | 0.2 |
| Maximum depth limit | Not applied |
| Stopping criteria | Maximum Nº generations |
| SAM noise magnitude ($\varepsilon$) | {0.1, 0.2, 0.5, 1.0} |
| SAM number of noisy perturbations ($n$) | {10, 20, 50} |

3, respectively. At each selection event, a criterion for selection is chosen at random (either training fitness or sharpness). The returned individual is the one that exhibits the highest $R^2_{\text{train}}$ and the smallest sharpness.

Recently, Banzhaf and Bakurov [4] proposed an effective approach to map a genotype to its phenotype in TGP by removing semantically ineffective code from the former. This genotype-phenotype mapping (GPM) is highly efficient as it is built upon the mechanics of the fitness evaluation (which inevitably needs to take place), avoiding, therefore, redundant function calls and calculations. The authors show that smaller phenotypes are hidden within larger genotypic trees, easily extracted to facilitate interpretability. They also studied the population dynamics of both genotypes and phenotypes and concluded that population behavior is normally based on a scarce number of unique and small phenotypes, which—curiously—happens even when evolution is more explorative. The authors observed that the growth rate of phenotype size is notably smaller than that of genotypes. All of this suggests that using the GPM approach presented in [4], one can extract valuable insights about evolutionary dynamics. To make the best use of these results and allow an unbiased assessment of the impact of SAM in terms of bloat, no limit to tree depth was applied during evolution. Initially, we did not have any assumption about SAM's impact on solution size; instead, we wanted to allow for more degrees of freedom for SAM to manifest its features.

To conduct our experiments, we use the General Purpose Optimization Library [1]. GPOL is a flexible and efficient multi-purpose optimization library in Python that covers a wide range of stochastic iterative search algorithms, including GP. Its modular implementation allows for solving optimization problems, like the one in this study, and easily incorporates new methods. The library is open-source and can be found by following this link. The implementation of the proposed approach can be found there.

## 8.5 Experimental Results

Table 8.5 reports the ranks on the generalization ability obtained by different SAM variants and baseline TGP. To build the table, the $R^2$ fitness values on the test data observed in the last generation were ranked in ascending order. Then, the ranks were averaged by algorithm type (standard GP and SAM variants). From the table, one can observe that several of the proposed SAM variants tend to rank the best more often than standard GP in terms of generalization ability. When comparing two distinct SAM approaches, SAM-IN ranks above its SAM-OUT counterpart. Additionally, from this table, we can extract the reference parameters for the two SAM variants. Both tend to produce better ranks using mild perturbations (0.1) and small neighborhood (10 and 20 for SAM-IN and SAM-OUT, respectively). From the table, these are $\text{SAM-IN}^{0.1}_{10}$ and $\text{SAM-OUT}^{0.1}_{20}$. These SAM configurations were used in all figures of this section as the default SAM-IN and SAM-OUT, depicted as dashed dark blue and dark red lines.

**Table 8.5** Generalization ability ranks achieved by each version of SAM and standard GP. Large values correspond to high fitness ranks. Colors reflect fitness: the green end of the green-yellow-red color scale highlights the highest (best) ranks observed in an experiment; the red end of the scale highlights the smallest (worse) ranks. The last row averages results of each column (SAM experiment)

| Algorithm | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SAM | GP | $SAM_{10}^{0.10}$ | $SAM_{20}^{0.10}$ | $SAM_{50}^{0.10}$ | $SAM_{10}^{0.20}$ | $SAM_{20}^{0.20}$ | $SAM_{50}^{0.20}$ | $SAM_{10}^{0.50}$ | $SAM_{20}^{0.50}$ | $SAM_{20}^{1.0}$ | $SAM_{10}^{1.0}$ | $SAM_{50}^{0.50}$ | $SAM_{50}^{1.0}$ |
| IN | 7.38 | 9.88 | 6.5 | 8.88 | 8.63 | 3.75 | 8 | 7 | 6.13 | 5.38 | 6.13 | 7.25 | 6.13 |
| OUT | | 4.63 | 8.5 | 6.13 | 8.13 | 4.75 | 6.5 | 6.5 | 7 | 6.88 | 7.13 | 6.63 | 5.25 |
| AVG | | 7.25 | 7.50 | 7.50 | 8.38 | 4.25 | 7.25 | 6.75 | 6.56 | 6.13 | 6.63 | 6.94 | 5.69 |

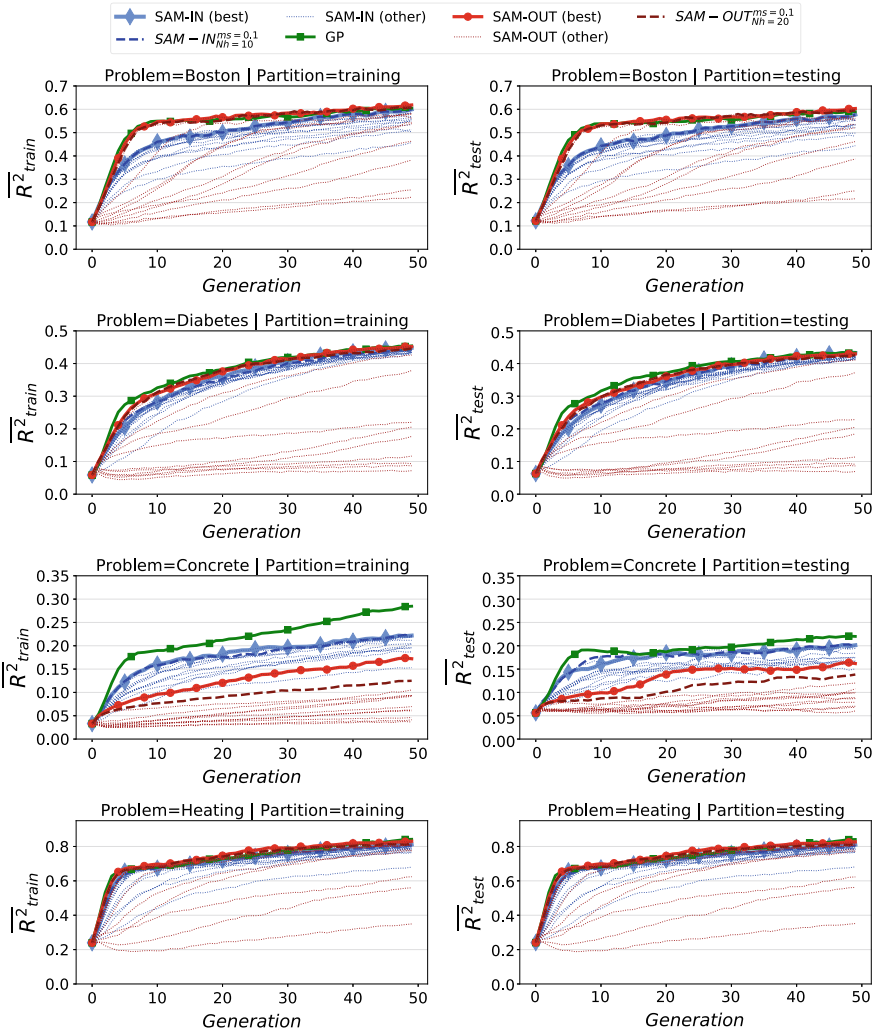


**Fig. 8.3** Average population training (left) and test (right) fitness of SAM approaches versus standard GP on the 4 real-world datasets

Figures 8.3 and 8.4 show the average fitness in the population for real-world and synthetic problems, respectively. Each problem is depicted in a given row, whereas each column represents a given partition: training on the left, and test on the right. The solid green lines represent standard GP (in green). The solid red and royal blue lines represent the SAM-IN and SAM-OUT methods that achieved the best generalization ability on a given problem. The SAM-IN and SAM-OUT configurations that were found to score the best more often across all the problems are depicted as dashed blue and dark red lines (these are the recommended SAM configurations and were

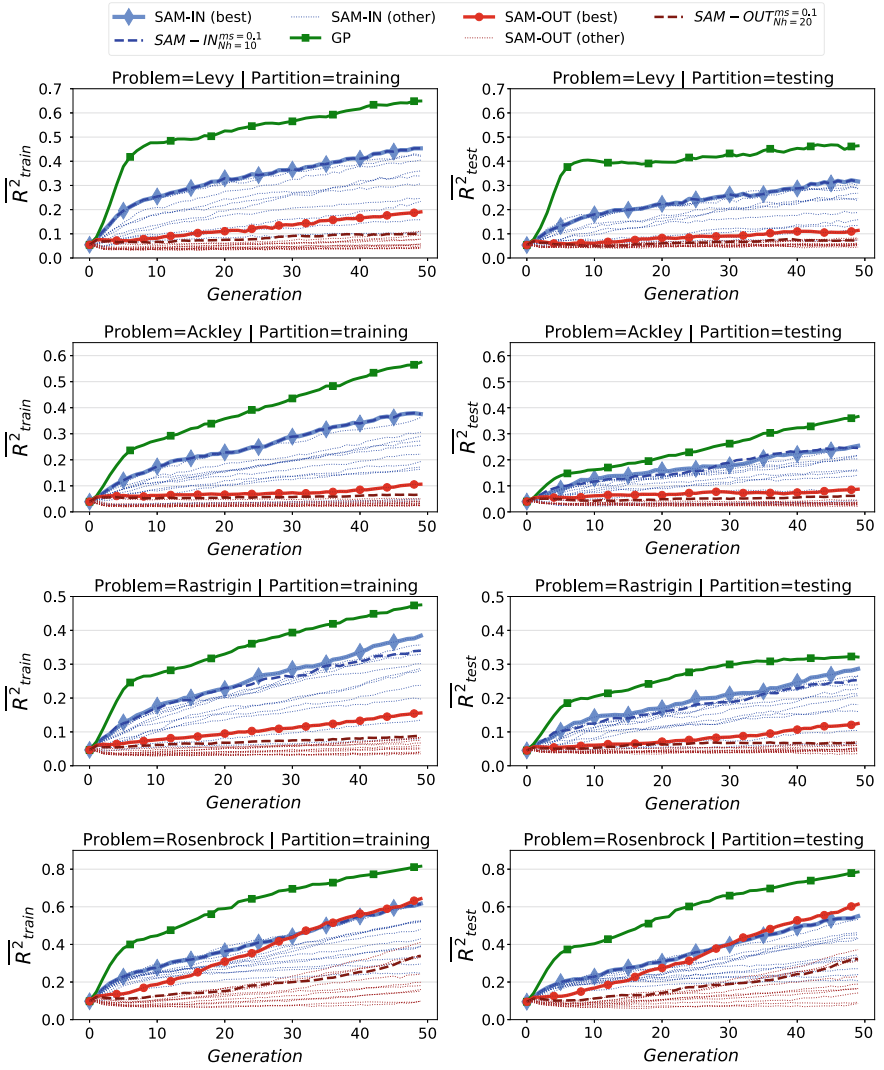


**Fig. 8.4** Average population training (left) and test (right) fitness of SAM approaches versus standard GP on the 4 synthetic datasets

obtained from Table 8.5). The remaining SAM-IN and SAM-OUT configurations are depicted as thin dotted lines, in blue and dark red, respectively. When looking at Fig. 8.3, the average population fitness is roughly the same for standard GP and the SAM approaches on Boston, Diabetes, and Heating. On the Boston problem, however, SAM-IN shows a slower convergence. On the Concrete problem, the highest fitness values are observed for standard GP, followed by SAM-IN and ultimately SAM-OUT. This particular problem seems more challenging to generalize on test

data for all the GP variants. It is also relevant to point out that the recommended SAM configurations tend to overlap with the best SAM configuration for particular problems, highlighting their robustness across problems.

When looking at Fig. 8.4, we observe a higher average fitness in the population when using standard GP on all the problems, across both partitions. SAM-IN follows standard GP and SAM-OUT achieves the worst average fitness values on both partitions. This is not the case, however, on the Rosenbrock problem where the best SAM-IN and SAM-OUT achieve competitive performance. In conclusion, the proposed SAM approaches tend to produce similar population training and test fitness on real-world problems when compared to standard GP, while on synthetic low dimensional problems they report worse fitness.

Figures 8.5 and 8.6 show the training and test fitness of the elite individuals, averaged across the evolutionary runs. The figure arrangement conforms to that of Figs. 8.3 and 8.4. When looking at Fig. 8.5, SAM-OUT converges faster and achieves better generalization when compared to both standard TGP and SAM-IN. Specifically, it achieves better test fitness than TGP on Boston and Heating problems, and similar fitness on Diabetes. Only on the Concrete problem, which seems to be challenging to generalize for all the algorithms, SAM-OUT achieves worse generalization. SAM-IN produces similar results to TGP across three problems (Boston, Diabetes, and Concrete) and scores the worst on Heating. For the elite fitness on synthetic problems, depicted in Fig. 8.6, standard TGP provided the best generalization, followed by SAM-IN which achieved the highest test scores on Rastrigin and similar scores on the Levy problem. SAM-OUT achieved the worst generalization ability, except on the Rosenbrock problem where it outperformed SAM-OUT but remained below TGP. In conclusion, the proposed SAM approaches tend to generate elites with equivalent or superior training and test fitness on real-world problems, while on synthetic low dimensional problems they report worse fitness.

We also recorded the average length of trees (node count) in the population across the evolutionary runs for all problems. Additionally, we measure the average difference between the whole genotype and the behavioral determinants (phenotype) of the individuals in the population using the GPM proposed in [4]. The latter was used to compute the ratio of redundant code (introns) in the trees. We display those results in Figs. 8.7 and 8.8 for the real-world and synthetic problems, respectively. The left column represents the average length, whereas the column on the right represents the average proportion of redundancy. Each problem is depicted in a given row. The results show that both SAM methods tend to produce notably smaller trees and, within those trees, the amount of nodes that do not contribute to the behavior is significantly smaller (i.e., there is a higher utilization of the code). Usually, SAM-OUT produces smaller trees with less redundancy than SAM-IN. Although this happens in real and synthetic problems, it is particularly notable for the latter.

Figure 8.9 reports the number of seconds per generation (first vertical axis) and the population size (second vertical axis) averaged across 10 runs on two problems: Levy and Concrete. The experiment was performed on an Intel(R) Core(TM) i7-8750H CPU processor with 2.20 GHz, and 16 GB of RAM. Solid lines represent the time, and dashed lines represent the size. One can notice that both SAM approaches

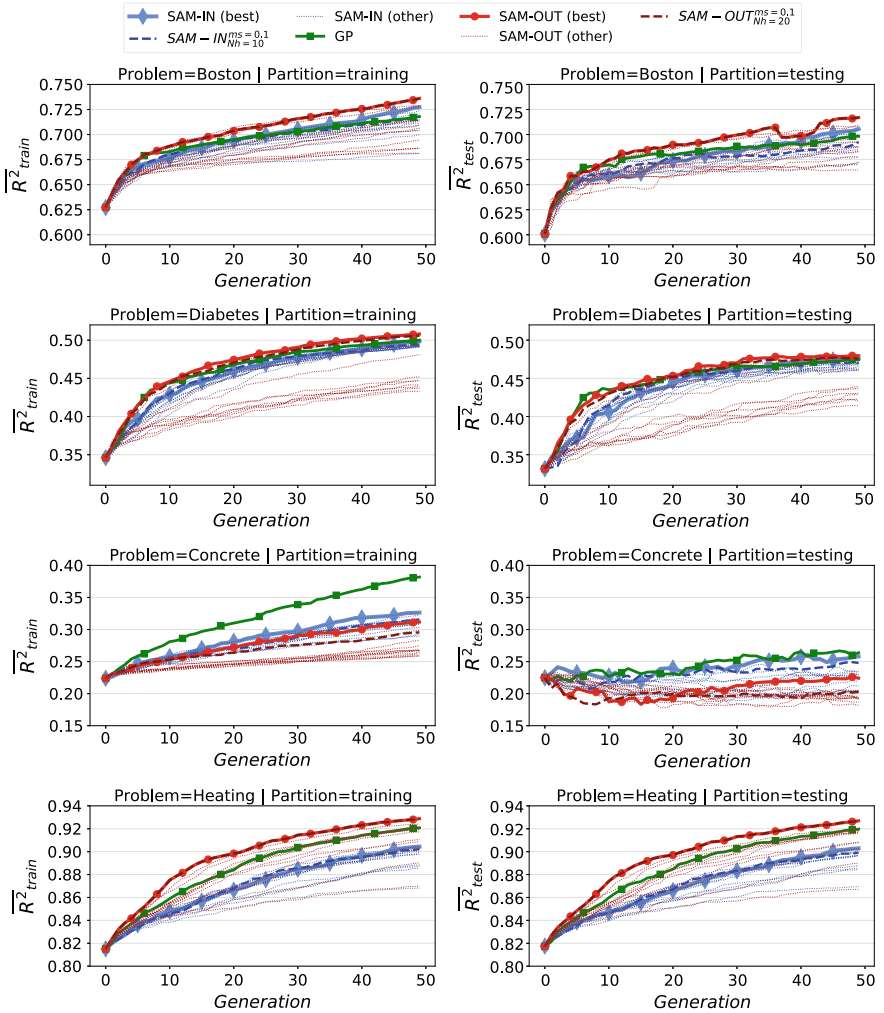


**Fig. 8.5** Average best-performing model fitness of SAM approaches versus standard GP on the 4 real-world datasets

report smaller runtimes and average lengths of individuals than standard GP, with SAM-OUT notably more efficient. Smaller average length was already observed in Figs. 8.7 and 8.8. The efficiency of SAM-OUT is expected as it operates exclusively on the semantics of trees, without executing them; moreover, it provides notably smaller trees, which gives additional benefits in terms of runtime. SAM-IN is shown to outperform standard GP in terms of both runtime and average length.

We would like to point out a further potential benefit of optimizing against sharpness: In scenarios where the goal is to use the model to predict a global optimum, such as in an active design of experiment context. In those scenarios, developing a model

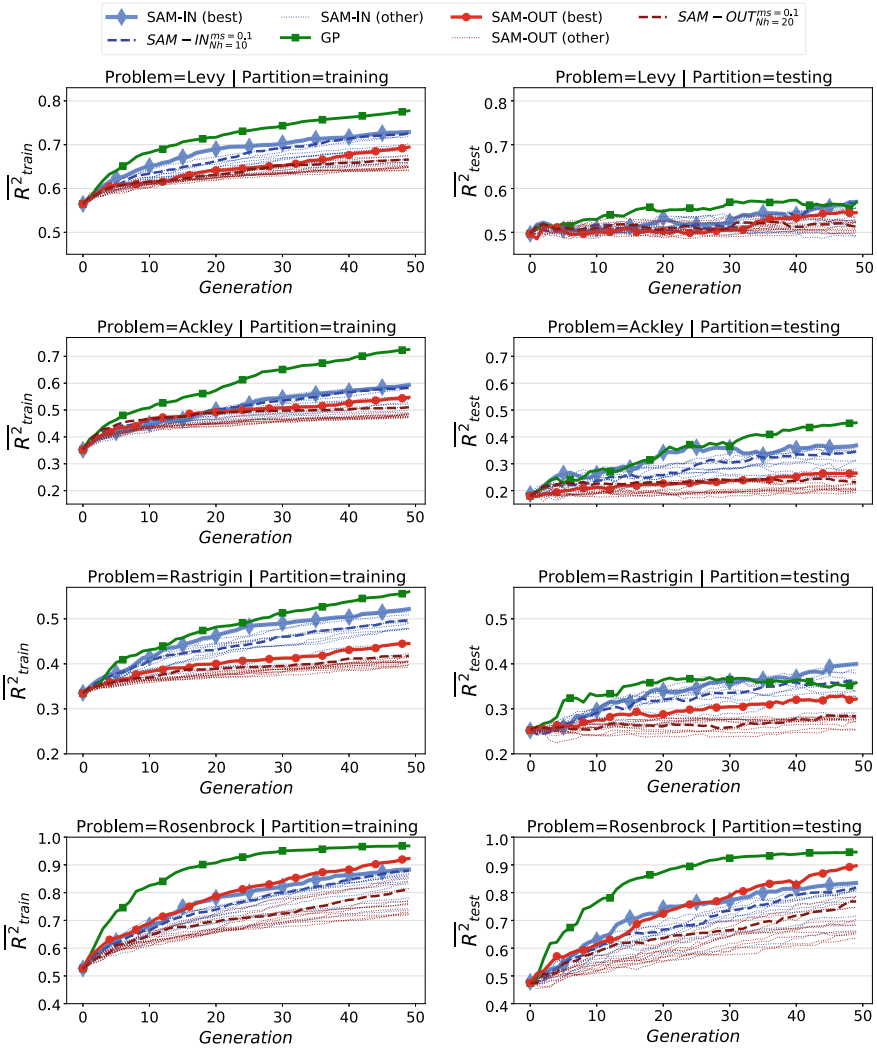


**Fig. 8.6** Average best-performing model fitness of SAM approaches versus standard GP on the 4 synthetic datasets

with a smoother surface would make optimizing on the surface simpler and could lead to a convex surface which would make finding the global optima even trivial. This is demonstrated in Fig. 8.10, where a sparse training dataset of 50 points was gathered from the 2D Rastrigin function (left-most subplot), which is notoriously rugged, and then TGP along with the two SAM approaches were used to generate approximation models of the dataset. The resulting SAM models are remarkably smoother than that provided by TGP (second subplot from the left). Both of them are convex and exhibit

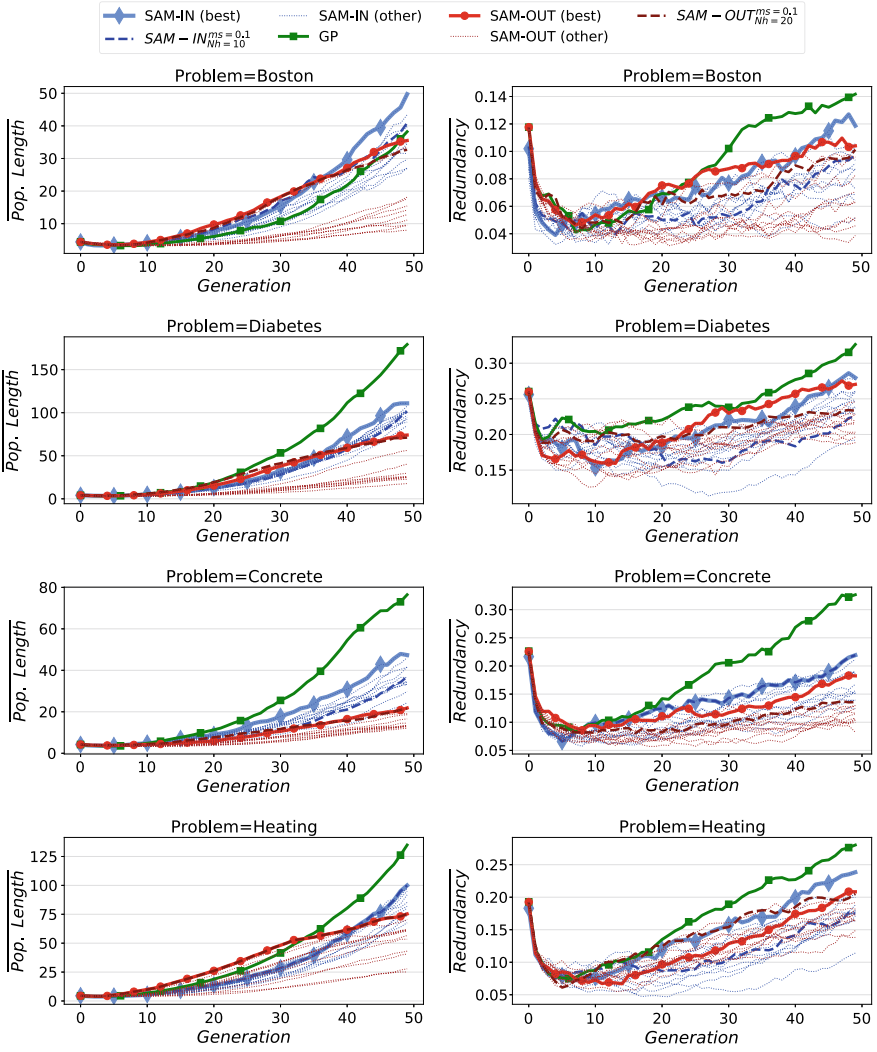


**Fig. 8.7** Average length trees in the population (left) and the average amount of redundancy (right). The latter was calculated as the difference between the genotype and phenotype using the method proposed in [4]. The figure regards 4 real-world datasets

a global minimum in the same location as the target function. Notice, however, that SAM-IN generated a better approximation of the target surface, which is supported by the analysis of aggregated performances on Synthetic problems in Fig. 8.6.

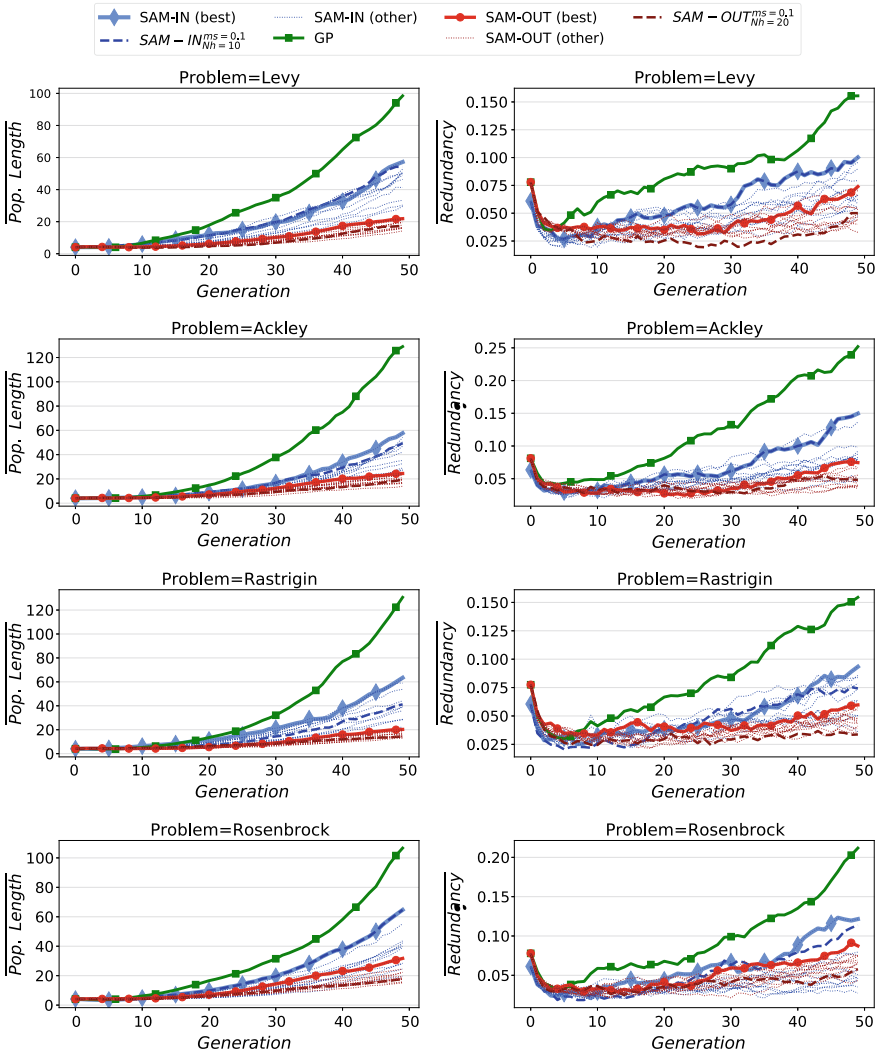


**Fig. 8.8** Average length trees in the population (left) and the average amount of redundancy (right). The latter was calculated as the difference between the genotype and phenotype using the method proposed in [4]. The figure regards 4 synthetic datasets

## 8.6 Conclusions

In Machine Learning, the term *learning* often refers to the task of inducing a general pattern from a provided set of examples, instead of *memorizing* them. This ability is known as *generalization* and represents one of the main battlegrounds in the field.

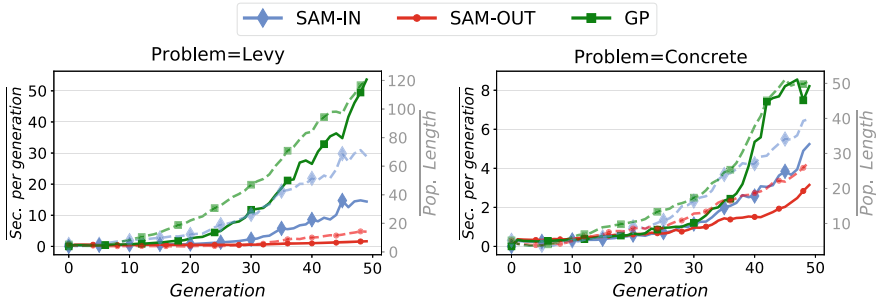


**Fig. 8.9** Average number of seconds per generation (first vertical axis) and average population size (second vertical axis) on two problems: Levy and Concrete. The figure aggregates data from 10 runs executed on an Intel(R) Core(TM) i7-8750H CPU processor with 2.20GHz, and 16GB of RAM. Solid lines represent the time and dashed lines represent the size

In this study, we adapt the Sharpness-Aware Minimization (SAM), previously introduced in Deep Learning (DL) to improve model generalization by simultaneously minimizing the loss and sharpness of the loss landscape through promoting parameters that lie in neighborhoods of uniformly low loss values. To accommodate SAM in GP, we propose two methods. The first, called SAM-IN, measures sharpness by adding slight perturbations to the terminals of the programs (i.e., to both constants and input features). The second, called SAM-OUT, measures sharpness by applying slight perturbations to the output of the programs (i.e., the semantics). Both penalize individuals for large fitness changes upon perturbation. In the latter case, sharpness is assessed without the need to execute the program, which makes it computationally more efficient.

Comparing the two SAM approaches to standard GP, we observe that co-selecting individuals by fitness and SAM leads to evolution to find smaller and less redundant models. On top of that, it allows for faster training times. For real-world problems which are characterized by high data dimensionality and noisy data, this was achieved without degradation of generalization ability. These findings are complementary with those observed in [40], who analyzed tree size distribution and found their SAM approach produces smaller trees than the standard GP. Following their formulation, SAM can be seen as a regularization technique in GP; however, unlike many methods like Parsimony Pressure [23], Grand Complexity [25], or Dynamic Limits [30], SAM does not consider model size as an optimization target.

Subsampling led to faster evaluation of sharpness, but it may have caused the SAM metric to be unstable compared to evaluating sharpness using the whole dataset. The subsampled metric may overlook localized sharpness in models if the subsample happens to select points only in smoother regions. Future work could explore the trade-off between subsample size and stability in the sharpness metric.

Incorporating model sharpness as a metric to detect model overfitting and instability could be essential for improving the stability, generalization ability, and simplicity of models developed by GP, leading to more trustworthy models. If a model becomes

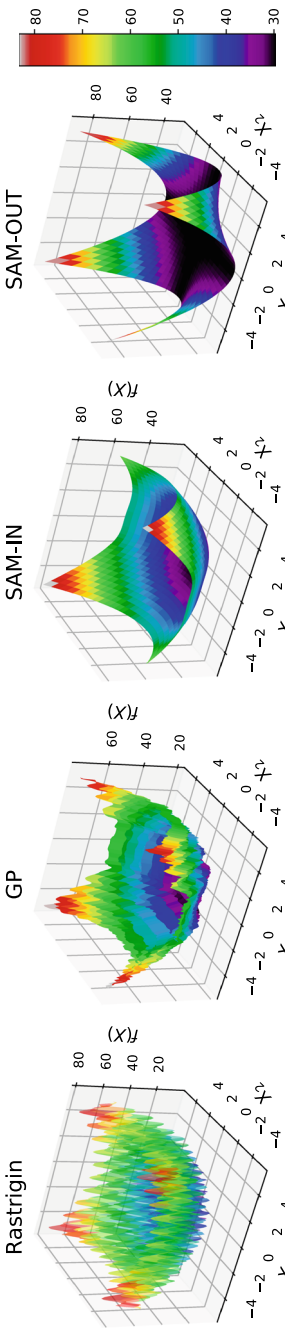


**Fig. 8.10** An example of using TGP and sharpness to generate approximations of the original data generator Rastrigin function (left subplot). Penalizing for sharpness allows smoother approximations, which can make finding the global optima from those approximations trivial

unstable when being applied to make predictions and returns values that are wildly infeasible, a user would likely lose trust in the approach and seek out another method for developing models.

Our method for detecting sharpness is agnostic to the specific machine learning method being used since the perturbations only occur at the inputs and outputs of models. This means that our approach is general and can easily be adopted for other machine learning approaches as long as assumptions about training data distribution are not violated. We believe that the proposed sharpness metric should become part of the standard machine learning pipeline either during model training as a fitness metric or, minimally, as a post-processing selection criterion to ensure the models being deployed are stable.

**Acknowledgements** WB acknowledges support from the Koza Endowment fund administered by Michigan State University. Computer support by MSU's iCER high-performance computing center is gratefully acknowledged.

## References

1. Bakurov, I., Buzzelli, M., Castelli, M., Vanneschi, L., Schettini, R.: General purpose optimization library (GPOL): a flexible and efficient multi-purpose optimization library in python. Appl. Sci. **11**(11), 4774 (2021). https://www.mdpi.com/2076-3417/11/11/4774
2. Bakurov, I., Castelli, M., Gau, O., Fontanella, F., Vanneschi, L.: Genetic programming for stacked generalization. Swarm Evol. Comput. **65**, 100913 (2021). https://doi.org/10.1016/j.swevo.2021.100913
3. Bakurov, I., Muñoz Contreras, J.M., Castelli, M., Rodrigues, N., Silva, S., Trujillo, L., Vanneschi, L.: Geometric semantic genetic programming with normalized and standardized random programs. Genet. Program. Evol. Mach. **25**(1) (2024). https://doi.org/10.1007/s10710-024-09479-1
4. Banzhaf, W., Bakurov, I.: On the nature of the phenotype in tree genetic programming. In: Li, X., Handle, J., et al. (eds.) Proceedings of the Genetic and Eovlutionary Computation Conference (GECCO-2024), pp. 868–877. ACM Press, New York (2024). https://dl.acm.org/doi/10.1145/3638529.3654129
5. Banzhaf, W., Hu, T., Ochoa, G.: How the combinatorics of neutral spaces leads genetic programming to discover simple solutions. In: Winkler, S., Trujillo, L., Ofria, C., Hu, T. (eds.) Genetic Programming Theory and Practice XX, pp. 65–86. Springer Nature, Singapore (2024). https://doi.org/10.1007/978-981-99-8413-8_4
6. Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., Parascandolo, G.: Neural symbolic regression that scales. In: International Conference on Machine Learning, pp. 936–945. PMLR (2021)
7. Brunton, S.L., Proctor, J.L., Kutz, J.N.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc. Natl. Acad. Sci. **113**(15), 3932–3937 (2016)
8. Da Costa, L., Landry, J.A., Levasseur, Y.: Treating noisy data sets with relaxed genetic programming. In: Artificial Evolution: 8th International Conference, Evolution Artificielle, EA 2007, Tours, France, October 29–31, 2007, Revised Selected Papers, pp. 1–12. Springer, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-540-79305-2_1
9. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Stat. **32**(2), 407–499 (2004). https://doi.org/10.1214/009053604000000067
10. Foret, P., Kleiner, A., Mobahi, H., Neyshabur, B.: Sharpness-aware minimization for efficiently improving generalization. In: ICLR Spotlight (2021). https://arxiv.org/abs/2010.01412

11. Gonçalves, I., Silva, S., Fonseca, C.M.: On the generalization ability of geometric semantic genetic programming. In: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (eds.) Genetic Programming, pp. 41–52. Springer International Publishing, Cham (2015)
12. Gonçalves, I., Silva, S., Fonseca, C.M., Castelli, M.: Unsure when to stop? In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM (2017). https://doi.org/10.1145/3071178.3071328
13. Gonçalves, I., Silva, S., Melo, J.B., Carreiras, J.M.B.: Random sampling technique for overfitting control in genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) Genetic Programming, pp. 218–229. Springer, Berlin, Heidelberg (2012)
14. Harrison, D., Rubinfeld, D.L.: Hedonic housing prices and the demand for clean air. J. Environ. Econ. Manag. **5**(1), 81–102 (1978). https://www.sciencedirect.com/science/article/pii/0095069678900062
15. Haut, N., Banzhaf, W., Punch, B.: Correlation versus RMSE loss functions in symbolic regression tasks. In: Trujillo, L., Winkler, S.M., Silva, S., Banzhaf, W. (eds.) Genetic Programming Theory and Practice XIX, pp. 31–55. Springer Nature, Singapore (2023). https://doi.org/10.1007/978-981-19-8460-0_2
16. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Trans. Evol. Comput. **9**(3), 303–317 (2005)
17. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (eds.) Genetic Programming, pp. 70–82. Springer, Berlin, Heidelberg (2003)
18. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
19. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**, 87–112 (1994)
20. La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.: Contemporary symbolic regression methods and their relative performance. In: Vanschoren, J., Yeung, S. (eds.) Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, vol. 1 (2021). https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c0c7c76d30bd3dcaefc96f40275bdc0a-Paper-round1.pdf
21. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, pp. 6391–6401. Curran Associates Inc., Red Hook, NY, USA (2018)
22. Livadiotis, G., McComas, D.J.: Fitting method based on correlation maximization: applications in space physics. J. Geophys. Res.: Space Phys. **118**(6), 2863–2875 (2013). https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/jgra.50304
23. Luke, S., Panait, L.: Fighting bloat with nonparametric parsimony pressure. In: Guervós, J.J.M., Adamidis, P., Beyer, H.G., Schwefel, H.P., Fernández-Villacañas, J.L. (eds.) Parallel Problem Solving from Nature - PPSN VII, pp. 411–421. Springer, Berlin, Heidelberg (2002)
24. Moraglio, A., Krawiec, K., Johnson, C.: Geometric semantic genetic programming. In: Coello, C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) Parallel Problem Solving from Nature - PPSN XII. Lecture Notes in Computer Science, vol. 7491, pp. 21–31. Springer, Berlin Heidelberg (2012)
25. Ni, J., Rockett, P.: Tikhonov regularization as a complexity measure in multiobjective genetic programming. IEEE Trans. Evol. Comput. **19**(2), 157–166 (2015). https://doi.org/10.1109/TEVC.2014.2306994
26. Nordin, P., Banzhaf, W.: An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. Adapt. Behav. **5**(2), 107–140 (1997)
27. Pietropolli, G., Manzoni, L., Paoletti, A., Castelli, M.: Combining geometric semantic GP with gradient-descent optimization. In: Genetic Programming: 25th European Conference. EuroGP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings, pp. 19–33. Springer, Berlin, Heidelberg (2022)

28. Pietropolli, G., Manzoni, L., Paoletti, A., Castelli, M.: On the hybridization of geometric semantic GP with gradient-based optimizers. Genet. Program. Evol. Mach. **24**(2), Article number: 16 (2023). https://rdcu.be/dpWsR. Online first
29. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms-a comprehensive survey. Swarm Evol. Comput. **33**, 18–45 (2017)
30. Silva, S., Costa, E.: Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genet. Program. Evol. Mach. **10**, 141–179 (2009). https://api.semanticscholar.org/CorpusID:10925054
31. Sudholt, D.: Analysing the robustness of evolutionary algorithms to noise: refined runtime bounds and an example where noise is beneficial. Algorithmica **83**(4), 976–1011 (2021). https://doi.org/10.1007/s00453-020-00671-0
32. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, GECCO'01, pp. 155–162. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
33. Tsanas, A., Xifara, A.: Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy Build. **49**, 560–567 (2012). https://www.sciencedirect.com/science/article/pii/S037877881200151X
34. Vanneschi, L.: An introduction to geometric semantic genetic programming. In: Schütze, O., Trujillo, L., Legrand, P., Maldonado, Y. (eds.) NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop NEO 2015 held at September 23–25 2015 in Tijuana, Mexico, pp. 3–42. Springer International Publishing, Cham (2017)
35. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th annual Conference on Genetic and Evolutionary Computation, pp. 877–884 (2010)
36. Vanneschi, L., Gustafson, S.: Using crossover based similarity measure to improve genetic programming generalization ability. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1139–1146 (2009)
37. Vanneschi, L., Silva, S., Castelli, M., Manzoni, L.: Geometric semantic genetic programming for real life applications. In: Riolo, R., Moore, J.H., Kotanchek, M. (eds.) Genetic Programming Theory and Practice XI, pp. 191–209. Springer, New York, NY (2014)
38. Vladislavleva, E.J., Smits, G.F., den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009)
39. Yeh, I.C.: Concrete compressive strength. UCI machine learning repository (2007). https://doi.org/10.24432/C5PK67
40. Zhang, H., Chen, Q., Xue, B., Banzhaf, W., Zhang, M.: Sharpness-aware minimization for evolutionary feature construction in regression (2024). arXiv: https://arxiv.org/pdf/2405.06869