Racing Control Variable Genetic Programming for Symbolic Regression

Nan Jiang, Yexiang Xue

Department of Computer Science, Purdue University, USA
{jiang631, yexiang}@purdue.edu

Abstract

Symbolic regression, as one of the most crucial tasks in AI for science, discovers governing equations from experimental data. Popular approaches based on genetic programming, Monte Carlo tree search, or deep reinforcement learning learn symbolic regression from a fixed dataset. These methods require massive datasets and long training time especially when learning complex equations involving many variables. Recently, Control Variable Genetic Programming (CVGP) has been introduced which accelerates the regression process by discovering equations from designed control variable experiments. However, the set of experiments is fixed a-priori in CVGP and we observe that sub-optimal selection of experiment schedules delay the discovery process significantly. To overcome this limitation, we propose Racing Control Variable Genetic Programming (Racing-CVGP), which carries out multiple experiment schedules simultaneously. A selection scheme similar to that used in selecting good symbolic equations in the genetic programming process is implemented to ensure that promising experiment schedules eventually win over the average ones. The unfavorable schedules are terminated early to save time for the promising ones. We evaluate Racing-CVGP on several synthetic and real-world datasets corresponding to true physics laws. We demonstrate that Racing-CVGP outperforms CVGP and a series of symbolic regressors which discover equations from fixed datasets.

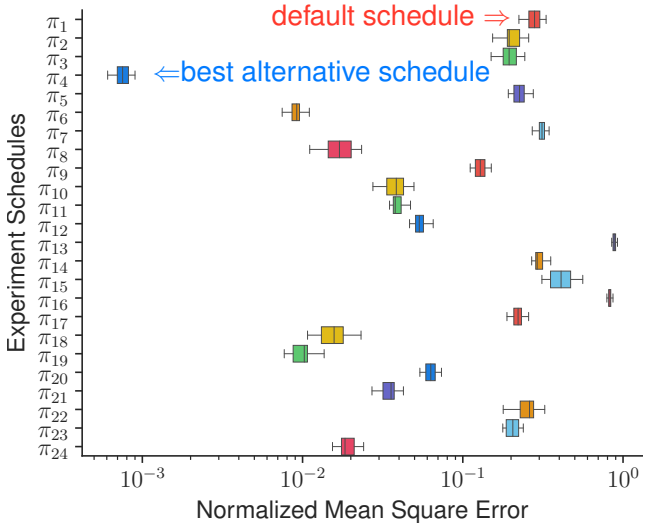


Figure 1: Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming. For the discovery of expression with 4 variables, there exists a better experiment schedule (*i.e.*, π_4) among all schedules than the default one (*i.e.*, π_1), in terms of normalized mean square error (more examples in Appendix D).

1 Introduction

Automatically discovering scientific laws from experimental data has been a long-standing aspiration of Artificial Intelligence. Its success holds the promise of significantly accelerating scientific discovery. A crucial step towards achieving this ambitious goal is symbolic regression, which involves learning explicit expressions from the experimental data. Recent advancements in this field have shown exciting progress, including works on genetic programming, Monte Carlo tree search, deep reinforcement learning and their combinations (Schmidt and Lipson 2009; Virgolin, Alderliesten, and Bosman 2019; Guimerà et al. 2020; Petersen et al. 2021; Mundhenk et al. 2021; Petersen et al. 2021; Razavi and Gamazon 2022; He et al. 2022; Sun et al. 2023; Tohme, Liu, and Youcef-Toumi 2023).

Despite remarkable achievements, the current state-of-the-art approaches are still limited to learning relatively simple expressions, typically involving only a few independent variables. The real challenge lies in symbolic regression involving multiple independent variables. The aforementioned approaches learn symbolic equations from a fixed dataset. As a result, these methods require massive datasets and extensive training time to discover complex equations.

Recently, a novel approach called Control Variable Genetic Programming (CVGP) (Jiang and Xue 2023) is introduced to accelerate symbolic regression. Instead of learning from fixed datasets collected a-priori, CVGP carries out symbolic regression using customized control variable experiments. As a motivating example, to learn the ideal gas law $pV = nRT$, one can hold n (gas amount) and T (temperature) as constants. It is relatively easy to learn p (pressure) is inversely proportional to V (volume). Indeed, CVGP

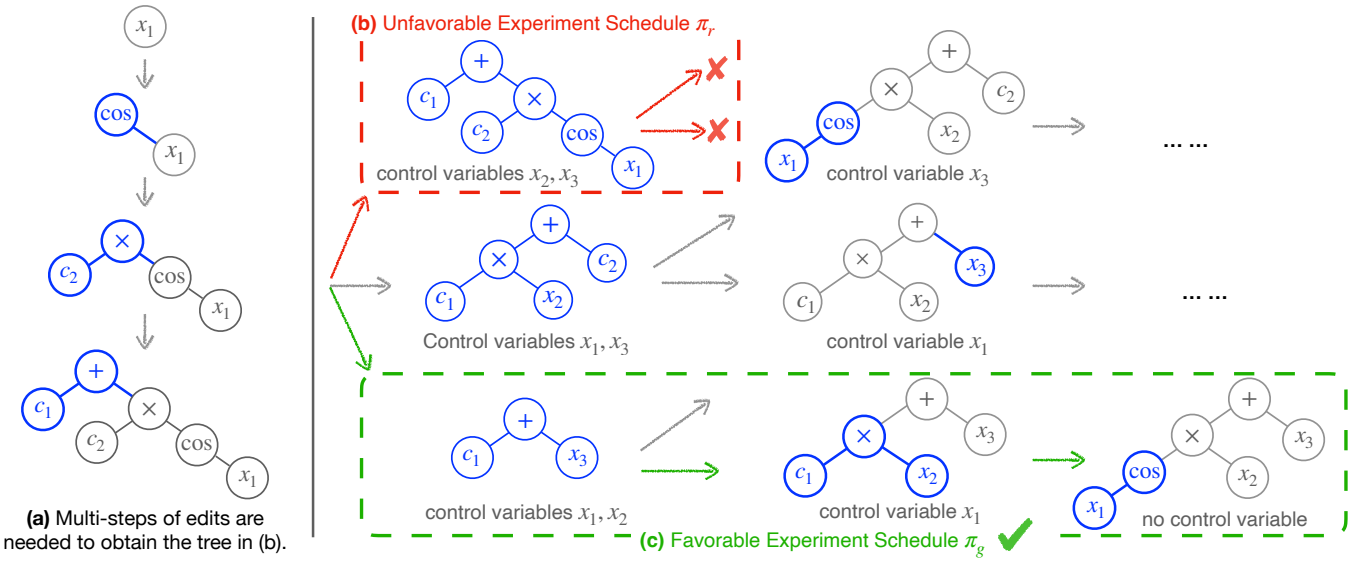


Figure 2: The favorable experiment schedule π_g is survived while the unfavorable schedule π_r is early stopped under our racing experiment schedule scheme. **(a)** Multiple steps of edits are needed to transform from a randomly initialized expression "x_1" to a complex expression "$c_1 + c_2 \cos(x_1)$". The newly inserted parts (by genetic programming algorithm) are highlighted in blue. **(b)** The red experiment schedule π_r is unfavorable because it requires many edits to reach the expression tree in the red box (shown in (a)). The red schedule is thus stopped early. **(c)** The green experiment schedule π_g is promising since it is relatively easy to discover, and every change in the expression tree is reasonable. Section 3 provides a detailed explanation.

discovers a chain of simple-to-complex symbolic expressions; e.g., first an expression involving only p and V, then involving p, V, T, etc. In each step, learning is carried out on specially collected datasets where a set of variables are held constant. The major difference between CVGP and previous approaches is that CVGP *actively explores* the space of all expressions via control variable experiments, instead of learning passively from a pre-collected dataset.

However, the set of experiments is fixed a-priori in CVGP. It first learns an equation involving only the first variable, then involving the first two variables, etc. In particular, CVGP works with a fixed *experiment schedule* (noted as π), that is the sequences of controlled variables. We observe that the sub-optimal selection of experiment schedules delays the discovery process significantly. In Fig. 1, we run CVGP with all 24 possible experiment schedules and report the quartiles of normalized mean squared errors (NMSE) of the discovered top 20 expressions. We see that certain experiment schedules (such as π_4) are significantly better than others including the default schedule π_1.

To overcome this limitation, we propose Racing-CVGP, which automatically discovers good experiment schedules that lead to accurate symbolic regression. A selection scheme over the experiment schedules is implemented, similar to that used in selecting good symbolic equations in the genetic programming process, to ensure that promising experiment schedules eventually win over the average schedules. The unfavorable schedules are terminated early to save time for promising schedules. Racing-CVGP allows flexible control variables experiments to be performed during the discovery process. If a specific set of controlled variable experiments fails to discover a good expression, it is ranked at the bottom and is eventually removed by the selection scheme. Our idea allows the algorithm to avoid spending excessive time on unfavorable experiment schedules and to focus on exploring promising controlled variable experiment schedules.

In experiments, we compare Racing-CVGP against several popular symbolic regression baselines using challenging datasets with multiple variables. On several datasets, we observe that Racing-CVGP discovers higher quality expressions in terms of the NMSE metric against several baselines. Our Racing-CVGP also takes less computational time than all the baselines. Our Racing-CVGP stops those unfavorable schedules early, which commonly leads to a longer training time. Notably, our method scales well to expressions with 8 variables while the GP, CVGP, and GPMeld methods take more than 2 days and thus are time-consuming. Our contributions can be summarized as follows:

- We identify a sub-optimal selection of experiment schedule that greatly delays the discovery process of symbolic regression. We propose Racing-CVGP to accelerate scientific discovery by maintaining good experiment schedules during learning challenging symbolic regression tasks.
- Under our racing experiment schedule, a favorable schedule is survived while unfavorable schedules are stopped early. We show that the time complexity of our Racing-CVGP is approximately close to that of the CVGP, under mild assumptions.
- In experiments, we showcase that our Racing-CVGP leads to faster discovery of symbolic expressions with smaller NMSE metrics, compared to current popular baselines over

several challenging datasets[1].

2 Preliminaries

Symbolic Regression for Scientific Discovery

A *symbolic expression* ϕ is expressed as variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and constants $\mathbf{c} = \{c_1, \ldots, c_m\}$, connected by a set of binary operators (like $\{+, -, \times, \div\}$) and/or unary operators (like $\{\sin, \cos, \log, \exp\}$). The operator set is noted as O_p. Each operand of an operator is either a variable, a constant, or a self-contained sub-expression. For example, "$x_1 + x_2$" is a expression with 2 variables (x_1 and x_2) and one binary operator ($+$). A symbolic expression can be equivalently represented as a *binary expression tree*, where the leaf nodes correspond to variables and constants and the inner nodes correspond to those operators. Fig. 3(a,b) presents two example expression trees.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and a loss function $\ell(\cdot, \cdot)$, the task of *symbolic regression* is to find the optimal symbolic expression ϕ^* with minimum loss over dataset $\mathcal{D}$, among the set of all candidate expressions (noted as Π):

$$\phi^* \leftarrow \arg\min_{\phi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i), \tag{1}$$

where the values of the open constants $\mathbf{c}$ in ϕ are determined by fitting the expression to the dataset $\mathcal{D}$. The loss function $\ell(\cdot, \cdot)$ measures the distance between the output from the candidate expression $\phi(\mathbf{x}_i, \mathbf{c}) \in \mathbb{R}$ and the ground truth $y_i \in \mathbb{R}$. A common choice of the loss function is Normalized Mean Squared Error (NMSE). Symbolic regression is shown to be NP-hard (Virgolin and Pissis 2022), due to the exponentially large size of all the candidate expressions Π.

Genetic Programming for Symbolic Regression. Genetic Programming (GP) has been a popular method for solving symbolic regression. The core idea of GP involves managing a pool of candidate expressions, noted as $\mathcal{P}$. In each generation, these candidates undergo *mutation* and *mating* steps with certain probabilities. The mutation operations randomly replace, insert a node in the expression tree, or delete a sub-tree. The mating operations pick a pair of parent expression trees and exchange their two random subtrees. In the *selection* step, expressions with the highest fitness scores, are chosen as candidates for the next generation. Here the fitness scores (noted as $\mathbf{o} \in \mathbb{R}^N$) indicate the closeness of the predicted outputs to the ground-truth outputs, like the negative NMSE. Over several generations, the expressions that fit the data well, exhibiting high fitness scores, survive in the pool of candidate solutions. The best expressions discovered throughout all generations are recorded as *hall-of-fame* solutions, noted as $\mathcal{H}$.

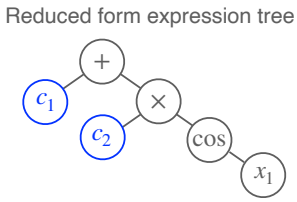
Control Variable Trials

In a regression problem, control variable trials study the relationship between a few input variables and the output with the remaining input variables fixed to be the same (Lehman, Santner, and Notz 2004; **?**). This idea was historically proposed to discover natural physical law, known as the BACON system (Langley 1977, 1979; Langley, Bradshaw, and Simon 1981). Recently, this idea has been explored for solving multivariable symbolic regression problems (Jiang and Xue 2023), *i.e.*, CVGP.

Let $\mathbf{x}_c \subseteq \mathbf{x}$ denote those control variables, and the rest are free variables. The values of controlled variables are fixed in each trial, which behaves exactly the same as constants for the learning method. In the controlled setting, the ground-truth expression behaves the same after setting those controlled variables as constants, which is noted as the *reduced form expression*. See Fig. 3(a,b) for two reduced form expressions with different control variable settings.

For a single control variable trial in symbolic regression, the corresponding dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ is first generated, where the controlled variables are fixed to one value and the remaining variables are randomly assigned. That is $\mathbf{x}_{i,k} = \mathbf{x}_{j,k}$ for the control variable x_k ($x_k \in \mathbf{x}_c$) and $1 \le i, j \le N$. See Fig. 3(a,b) for example datasets generated from the control variable trials. Given a reduced form expression and corresponding dataset, the values of open constants in the expression are determined by gradient-based optimizers, like the BFGS algorithm. In Fig. 3(a), the optimal values of open constants are $c_1 = 0.5, c_2 = 0.16$. Similarly in Fig. 3(b), we have $c_1 = 1.8$. The loss values (defined in Eq. (1)) of these two controlled variable trails over the dataset D_1 and dataset D_2 are equal to 0, indicating the optimal fitness scores.

The CVGP is built on top of the above control variable trials and GP algorithms (Jiang and Xue 2023). To fit an expression of n variables, CVGP initially only allows variable x_1 to vary and controls the values of all $n-1$ variables (*i.e.*, $\mathbf{x}_c = \mathbf{x} \backslash \{x_1\}$). Using GP as a subroutine, CVGP finds a pool of expressions $\{\phi_1, \ldots, \phi_{N_p}\}$ which best fit the data from this controlled experiment. Notice $\{\phi_1, \ldots, \phi_{N_p}\}$

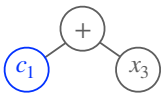


Dataset D_1

x_1	x_2	x_3	y
-1.0	0.5	0.16	0.20
0.92	0.5	0.16	0.21
0.72	0.5	0.16	0.22

Controlled to be the same

(a) controlled variable trials with $\mathbf{x}_c = \{x_2, x_3\}$.



Dataset D_2

x_1	x_2	x_3	y
0.62	1.0	0.1	0.18
0.62	1.0	0.2	0.36
0.62	1.0	0.3	0.54

Controlled to be the same

(b) controlled variable trials with $\mathbf{x}_c = \{x_1, x_2\}$.

Figure 3: **(a)** When controlling variables x_2 and x_3, the ground-truth expression $\phi = x_2 \cos(x_1) + x_3$ reduces to $c_1 \cos(x_1) + c_2$. **(b)** Controlling variables x_1 and x_2 reduces the ground-truth to $c_1 x_3$.

[1]The code is: https://bitbucket.org/xlnxyx/racing_cvgp. Please refer to https://arxiv.org/abs/2309.07934 for the Appendix.

are restricted to contain only one free variable x_1 and N_p is the pool size. This fact renders fitting them a lot easier than directly fitting the expressions involving all n variables. A small error implies that ϕ_i is close to the ground truth reduced to the one free variable. In the 2nd round, CVGP adds a second free variable x_2 and starts fitting $\{\phi'_1, \ldots, \phi'_{N_p}\}$ using the data from control variable experiments involving the two free variables x_1, x_2. After n rounds, the expressions in the CVGP pool consider all n variables. Note that CVGP assumes the existence of a `DataOracle` that allows for query a batch data with specified control variables.

3 Methodology

We first brief the issue with a fixed experiment schedule for the existing CVGP method in discovering symbolic regression. Then we present our racing experiment schedule for control variable genetic programming (Racing-CVGP).

Motivation

We define an *experiment schedule*, noted as π, as a sequence of variables controlled over all the rounds in CVGP. We use Fig. 2 to demonstrate different experiment schedules for the discovery of the ground-truth expression $\phi = \cos(x_1)x_2 + x_3$. In Fig. 2(c), CVGP runs an experiment schedule with control variables $\{x_1, x_2\}$ in the first round and runs with control variables $\{x_1\}$ in the second round and with no variable control $\emptyset$ in the last round. The corresponding experiment schedule is $\pi = (\{x_1, x_2\}, \{x_1\}, \emptyset)$. Similarly, Fig. 2(b) shows the default experiment schedule of CVGP that control variables $\{x_2, x_3\}$ initially and then control variable $\{x_3\}$, finally control no variable $\emptyset$, which is denoted as $\pi = (\{x_2, x_3\}, \{x_3\}, \emptyset)$.

Our key observations are as follows: (1) The experiment schedule plays a vital impact on the performance of CVGP than other components in the algorithm. (2) Some expressions are much easier to detect for specific experiment schedules. The existing CVGP method only considers a fixed experiment schedule $\pi = (\{x_2, \ldots, x_n\}, \{x_3, \ldots, x_n\}, \ldots, \{x_n\}, \emptyset)$ for discovering expression involving n variables. This fixed experiment schedule leads to sub-optimal performance of CVGP over some expressions, requiring more training data and computational time than other alternative schedules. See Fig. 1 for an empirical evaluation of different experiment schedules over the final identified expressions by the same CVGP method. See more examples in Appendix D.

In Fig. 2, we use the discovery of an expression $\phi = \cos(x_1)x_2 + x_3$ from the Feynman dataset as an example. The alternative (green) experiment schedule π_g in Fig. 2(c) is favorable while the default (red) schedule π_r in Fig. 2(b) is not. In Fig. 2(a), we visualize 3 necessary steps to reach from randomly initialized expression tree "x_1" to the final tree "$c_1 + c_2\cos(x_1)$" in Fig. 2(b). Every step of editing is conducted by the GP and requires drawing batches of training data to fit every intermediate expression. The edited subtrees are highlighted in blue. In comparison, it takes 1 step of edits in the tree to reach the first expression "$c_1 + x_3$" in the green experiment schedule, which leads to faster discovery using less training data. Following the green experiment schedule π_g, it takes 1 step of edits to reach the expression at the second round "$c_1x_2 + x_3$" and the last round "$\cos(x_1)x_2 + x_3$". Therefore, CVGP needs much more data and time in the 1st round following the default (red) experiment schedule π_r. The alternative (green) experiment schedule π_g is easier for the GP algorithm to discover the ground-truth expression using less data and time.

Directly evoking CVGP as a subroutine with multiple experiment schedules will not solve the problem. The expression in Fig. 1 has 24 different experiment schedules. The total running time is summarized in Fig. 6. In general, for an expression involving n variables, there are $n!$ many experiment schedules. It is time-intractable to run CVGP with all the experiment schedules for real-world scale problems.

To tackle the above issue, we propose a racing scheme over the experiment schedules. Our main principles are (1) maintaining multiple experiment schedules rather than one, and (2) allowing promising experiment schedules to survive while letting unfavorable schedules early stop. Our Racing-CVGP has a much higher chance of detecting high-quality expression using less training data and computational time than the existing CVGP.

Specifically, we implement a schedule selection procedure. Every expression in the population pool $\phi \in \mathcal{P}$ is attached with its own experiment schedule. In each round, we execute GP over all the expressions in the population pool for several generations. At the end of every round, the racing selection scheme removes (*resp.* preserves) those expressions with bad (*resp.* good) experiment schedules, based on their fitness scores. So those schedules that lead to higher fitness scores have a higher probability of survival.

We use Fig. 2 to visualize the process of our Racing-CVGP. We first initialize the population pool $\mathcal{P}$ in GP with several expressions for each control variable setting. We randomly generate simple expressions involving only x_1 with the control variables being $\{x_2, x_3\}$, where every expression is attached with a (partial) experiment schedule $\pi = (\{x_2, x_3\})$. We repeat this random expression generation for all the rest $n - 1$ control variable settings. For the 1st round, the GP algorithm is evoked over the population pool for several generations. Then we rank the expressions in the pool by the fitness score of the expression, where those expressions with higher fitness scores rank at the top of the pool. We only preserve top N_p expressions in population pool $\mathcal{P}$. Since it is much easier to detect $c_1 + x_3$ under control variable $\{x_1, x_2\}$ setting, the preserved majority expressions are attached with the experiment schedule $\pi_1 = \{x_1, x_2\}$. This ensures that we early stop the unfavorable experiment schedule $\pi = \{x_2, x_3\}$ in Fig. 2(b). Prior to the 2nd round, we randomly set free one variable from π_1. Fig. 2(c) set the free variable x_2 and only variable x_1 is controlled in the 2nd round. In the 3rd round, the majority of the expressions in the population is attached to the experiment schedule $\pi_g = (\{x_1, x_2\}, \{x_1\}, \emptyset)$, since every change over the expression tree is reasonable. The total computational resources are saved from spending time searching for the expression tree in Fig. 2(b) to explore expressions with experiment schedule $\pi = (\{x_1, x_2\}, \{x_1\})$ in Fig. 2(c).

Racing Control Variable Genetic Programming

The high-level idea of Racing-CVGP is building simple to complex symbolic expressions involving increasingly more variables following those promising experiment schedules.

Notations. Denote K multiple control variable trials as a tuple $\langle \phi, \mathbf{o}, \mathbf{c}, \mathbf{x}_c, \pi, \{\mathcal{D}_k\}_{k=1}^K \rangle$. Here ϕ stands for the symbolic expression; the fitness scores $\mathbf{o} \in \mathbb{R}^K$ for expression ϕ indicates the closeness of predicted outputs to the ground-truth outputs; $\mathbf{c} \in \mathbb{R}^{K \times L}$ are the best-fitted values (by gradient-based optimizers) to open constants. Here L is the number of open constants in the expression ϕ; $\mathbf{x}_c \subseteq \mathbf{x}$ is the set of control variables; π is the (partial) experiment schedule that leads to the current expression ϕ. $\mathcal{D}_k = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ $(1 \leq k \leq K)$ is a randomly sampled batch of data from `DataOracle` with control variables $\mathbf{x}_c$. m denotes the batch size of the data.

Initialization. For single variable $x_i \in \mathbf{x}$, we create a set of candidate expressions that only contain variable x_i and save them into the population pool $\mathcal{P}$. Then we apply a GP-based algorithm to find the best-fitted expressions, which is referred to as the `BuildGPPool` function. The initialization step corresponds to Lines 2-6 in Algorithm 1.

Execution Pipeline. Given the current control variables $\mathbf{x}_c$, we first evoke the `DataOracle` to generate data batches $\{\mathcal{D}_k\}_{k=1}^K$. This corresponds to changing experimental conditions in real science experiments. We then fit open constants in the candidate expression ϕ_{new} with the data batches by gradient-based optimizers like BFGS (Fletcher 2000). This step is noted as the `Optimize` function. Then we obtain the fitness score vector $\mathbf{o}$ and solutions to open constants $\mathbf{c}$. We save the tuple $\langle \phi, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle$ into new population pool $\mathcal{P}_{new}$. This step corresponds to Lines 8-11 in Algorithm 1.

Then GP algorithm is applied for `#Gen` generations to search for optimal structures of the expression trees in the population pool P_{new}. The function `GP` is a minimally modified genetic programming algorithm for symbolic regression, which is detailed in Appendix B. The key differences between classic GP and our Racing-CVGP are

1. During mutation, our Racing-CVGP only alters the *mutable* nodes of the candidate expression trees. In classic GP, all the tree nodes are mutable, while in Racing-CVGP, the mutable nodes of the expression trees and set of operators O_p are preset by the `FreezeEquation`.
2. Mating is only applied over a pair of expressions with the same set of controlled variables in our Racing-CVGP. Classic GP, a random pair of expressions is selected for the mating operation.
3. `Optimize` operation in Racing-CVGP dynamically samples data with oracle $\mathcal{D}_o$ under control variable setup, whereas classic GP uses data with no variable controlled.

We preserve N_p best equations in the population $\mathcal{P}$. Every expression is evaluated with the different data from *its own control variables*. An unfavorable (partial) experiment schedule will be removed at this step when the corresponding expression ϕ has a low fitness score. The schedules in the pruned population pool $\mathcal{P}$ indicate that they are favorable.

Key information is obtained by examining the outcomes of K-trials control variable experiments: (1) Consistent close-to-zero fitness value, implies that the fitted expression is close to the ground-truth equation in the reduced form. That is $\sum_{k=1}^K \mathbb{I}(o_k \leq \varepsilon)$ should equal to K, where $\mathbb{I}(\cdot)$ is an indicator function and ε is the threshold for the fitness scores. (2) Given that the equation is close to the ground truth, an open constant having similar best-fitted values across K trials suggests that the open constants are standalone. Otherwise, that open constant is a *summary* constant, that corresponds to a sub-expression involving those control variables $\mathbf{x}_c$. The j-th open constant is a standalone constant when the empirical variance of its fitted values across K trials is less than a threshold ε'. The above steps are noted as `FreezeEquation` function. This freezing operation reduces the search space and accelerates the discovery.

Finally, we randomly drop a control variable in $\mathbf{x}_c$ and update the schedule π for each equation ϕ in the population pool $\mathcal{P}$. After n rounds, we return the equations in hall-of-fame $\mathcal{H}$ with best fitness values over all the schedules. Equations in $\mathcal{H}$ are evaluated on data with no variable controlled.

Running Time Analysis. The major hyper-parameters that impact the running time of Racing-CVGP are 1) the number of genetic operations per round M; 2) total rounds n; 3) the maximum size of population pool N_p. A rough estimation of the time complexity of the proposed Racing-CVGP is $\mathcal{O}(nMN_p)$, which is the same as the CVGP algorithm. Another implicit factor of running time is the number of open constants $|\mathbf{c}|$ for every expression $\phi(\mathbf{x}, \mathbf{c})$. An expression with more open constants needs more time for optimizers (like BFGS and CG) or more advanced optimizers (like Basin Hopping (Wales and Doye 1997)) to find the solutions. We leave it to the empirical time evaluation in Figure 6.

Connection to Existing Methods. Our work is relevant to a line of work (Langley 1977, 1979; Langley, Bradshaw, and Simon 1981; King et al. 2004, 2009; Cerrato et al. 2023) that implemented human scientific discovery using AI, pioneered by the BACON systems (Langley 1977, 1979; Langley, Bradshaw, and Simon 1981). While BACON's discovery

---

Algorithm 1: Racing Control Variable Genetic Programming

Input: #input variables n; operator set O_p; `DataOracle`.
Parameters: #genetic operations per rounds `#Gen`; Size of population pool N_p; #experiment trials K.

1: $\mathcal{P} = \{\}; \mathcal{H} = \{\}$.
2: **for** $i \leftarrow 1$ to n **do** ▷ initialize
3: $\quad \mathbf{x}_c = \{x_1, \ldots, x_n\} \setminus \{x_i\}$.
4: $\quad \mathcal{P} \leftarrow \mathcal{P} \cup$ `BuildGPPool`$(\mathbf{x}_c, O_p \cup \{\text{const}, x_i\})$.
5: **for** $i \leftarrow 1$ to n **do**
6: $\quad$ **for** $\langle \phi_{new}, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ **do** ▷ control variable trials
7: $\quad\quad \{\mathcal{D}_k\}_{k=1}^K \leftarrow$ `DataOracle`$(\mathbf{x}_c, K)$.
8: $\quad\quad \mathbf{o}, \mathbf{c} \leftarrow$ `Optimize`$(\phi_{new}, \{\mathcal{D}_k\}_{k=1}^K)$.
9: $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{\langle \phi, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle\}$.
10: $\quad \mathcal{P}, \mathcal{H} \leftarrow$ `GP`$(\mathcal{P}, \mathcal{H}, \text{DataOracle}, O_p \cup \{\text{const}, x_i\})$.
11: $\quad$ **for** $\langle \phi, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ **do** ▷ racing schedule
12: $\quad\quad \phi \leftarrow$`FreezeEquation`$(\phi)$.
13: $\quad\quad$ randomly drop a variable in $\mathbf{x}_c$.
14: $\quad\quad$ save $\mathbf{x}_c$ into π.
return the set of hall-of-fame equations $\mathcal{H}$.

Table 1: On Trigonometric datasets, median (50%) and 75%-quantile NMSE values of the expressions found by all the algorithms. Our Racing-CVGP finds symbolic expressions with the smallest NMSEs. "$T.O.$" implies the algorithm is timed out for 48 hours. The 3-tuples at the top $(\cdot, \cdot, \cdot)$ indicate the number of input variables, singular terms, and cross terms in the expression.

	(3, 2, 2)		(4, 4, 6)		(5, 5, 5)		(6, 6, 10)		(8, 8, 12)	
	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
Racing-CVGP (ours)	$<$ **1E-6**	$<$ **1E-6**	**0.016**	**0.021**	**0.043**	**0.098**	**0.069**	**0.104**	**0.095**	**0.286**
CVGP	0.039	0.083	0.028	0.132	0.086	0.402	0.104	0.177	$T.O.$	$T.O.$
GP	0.043	0.551	0.044	0.106	0.063	0.232	0.159	0.230	$T.O.$	$T.O.$
Eureqa	$<$ **1E-6**	$<$ **1E-6**	0.024	0.122	0.158	0.377	0.910	1.927	0.162	2.223
DSR	0.227	7.856	2.815	9.958	2.558	3.313	6.121	16.32	0.335	0.410
PQT	0.855	2.885	2.381	13.84	2.168	2.679	5.750	16.29	0.232	0.313
VPG	0.233	0.400	2.990	11.32	1.903	2.780	3.857	19.82	0.451	0.529
GPMeld	0.944	1.263	1.670	2.697	1.501	2.295	7.393	21.71	$T.O.$	$T.O.$
SPL	0.010	0.011	0.144	0.231	0.147	0.280	0.472	0.627	0.599	0.746

was driven by rule-based engines, our Racing-CVGP uses modern learning approaches such as genetic programming.

4 Related Work

Early works in symbolic regression (Langley 1981; Lenat 1977) use heuristic search. Genetic programming is effective in searching for good candidates (Udrescu and Tegmark 2020; Virgolin, Alderliesten, and Bosman 2019; He et al. 2022). Reinforcement learning-based methods use a risk-seeking policy gradient to find the expressions (Petersen et al. 2021; Mundhenk et al. 2021). Other works use RL to adjust the probabilities of genetic operations (Chen, Wang, and Gao 2020). Some works reduce the search space by considering the composition of base functions (McConaghy 2011; Chen, Luo, and Jiang 2017).

Current research efforts are devoted to searching for polynomials with a few variables (Uy et al. 2011), time series equations (Balcan et al. 2018), and equations in physics (Udrescu and Tegmark 2020). Multivariable symbolic regression is challenging since the search space increases exponentially w.r.t. the number of input variables. Existing works for multi-variable regression are based on pre-trained encoder-decoder methods with a massive training dataset (e.g., millions of data points (Biggio et al. 2021)), and even larger generative models (e.g., million of parameters (Kamienny et al. 2022)). Our Racing-CVGP is a tailored algorithm to solve multi-variable symbolic regression.

The choice of variables is an important topic in AI, including variable ordering in decision diagrams (Cappart et al. 2022), variable selection in tree search (Song et al. 2022a), variable elimination in probabilistic inference (Dechter 2019; Derkinderen et al. 2020) and backtracking search in constraint satisfaction problems (Ortiz-Bayliss et al. 2018; Li, Feng, and Yin 2020; Song et al. 2022b). Our method is one variant of variable ordering in symbolic regression.

Our work is also relevant to experiment design, which considers drawing a minimum amount of data for determining coefficients in linear regression models (Dette and Röder 1997; Yang and Stufken 2012; Attia and Ahmed 2023). Our work considers reducing the amount of total data needed to uncover the ground truth expression.

5 Experiments

This section demonstrates that Racing-CVGP finds the symbolic expressions with the smallest Normalized Mean-Square Errors (NMSE) (in Table 1 and Table 2) and takes less computational time (in Fig. 4), among all competing approaches on several noiseless datasets. In the ablation studies, we show our Racing-CVGP is consistently better than the baselines when evaluated in different metrics (in Fig. 5). Also, our Racing-CVGP methods save a great portion of time than evoke CVGP with all the possible schedules.

Experimental Settings

Datasets. We consider several publicly available and multi-variable datasets, including 1) Trigonometric datasets (Jiang and Xue 2023), 2) Livermore2 datasets (Petersen et al. 2021), and 3) Feynamn datasets (Matsubara et al. 2022).

Evaluation Metrics. We consider two evaluation criteria for the learning algorithms: 1) The goodness-of-fit measure (NMSE), indicates how well the learning algorithms perform in discovering symbolic expressions. The medians (50%) and 75%-percentiles of the NMSE are reported. We report median values instead of means due to outliers (see Ablation Studies). This is a common practice for combinatorial optimization problems. 2) The total running time of each learning algorithm.

Baselines. We consider the following baselines based on evolutionary algorithms: 1) Genetic Programming (GP) (Fortin et al. 2012). 2) Eureqa (Dubcáková 2011). We also consider a series of baselines using reinforcement learning: 3) Priority queue training (PQT) (Abolafia, Norouzi, and Le 2018). 4) Vanilla Policy Gradient (VPG) (Williams 1992). 5) Deep Symbolic Regression (DSR) (Petersen et al. 2021). 6) Neural-Guided Genetic Programming Population Seeding (GPMeld) (Mundhenk et al. 2021). 7) Symbolic Physics Learner (SPL) (Sun et al. 2023). The remaining details are provided in Appendix C.

Experimental Result Analysis

Goodness-of-fit Benchmark. Our Racing-CVGP attains the smallest median (50%) and 75%-quantile NMSE values among all the baselines when evaluated on selected Trigonometric, Livermore2, and Feynman datasets (Table 1). This

Table 2: On Livermore2 and Feynman datasets, median (50%) and 75%-quantile NMSE values of the symbolic expressions found by all the algorithms. Our Racing-CVGP finds symbolic expressions with the smallest NMSEs. n is the number of independent variables in the expression.

	Livermore2 ($n=4$)		Livermore2 ($n=5$)		Livermore2 ($n=6$)		Feynman ($n=4$)		Feynman ($n=5$)	
	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
Racing-CVGP (ours)	**< 1E-6**	**2.03E-3**	**0.004**	**0.047**	**0.001**	**0.073**	**0.015**	**0.195**	0.577	**0.790**
CVGP	0.052	0.810	0.275	1.007	0.328	1.012	1.002	1.010	1.001	1.002
GP	0.059	0.962	0.331	1.003	1.001	1.026	1.003	1.010	1.002	1.011
Eureqa	0.508	0.980	0.083	0.249	0.026	0.302	0.026	0.397	**0.434**	0.943
DSR	0.030	0.048	0.050	0.284	0.230	0.486	0.216	0.920	0.976	1.001
PQT	0.042	0.063	0.074	0.227	0.170	0.410	0.172	0.765	1.003	1.027
VPG	0.037	0.074	0.093	0.322	0.206	0.535	0.188	0.971	1.006	1.025
GPMeld	0.029	0.061	0.049	0.259	0.144	0.504	0.177	0.708	0.940	1.002
SPL	0.035	0.463	0.181	0.201	0.229	1.005	0.143	0.542	0.632	1.002

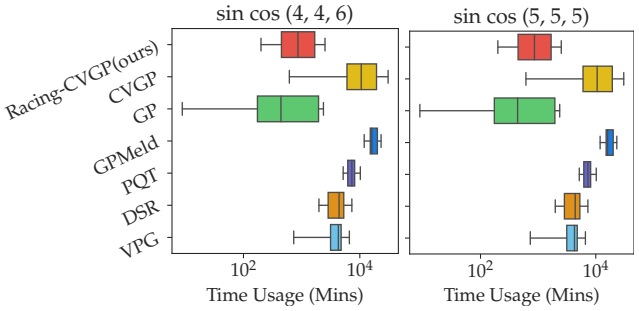


Figure 4: On selected Trigonometric datasets, quartiles of the total running time of all the methods. Our Racing-CVGP method takes less time than CVGP by early stopping those unfavorable experiment schedules.

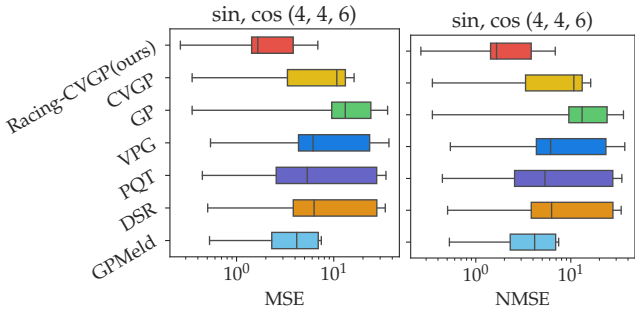


Figure 5: On selected Trigonometric datasets, MSE and NMSE evaluation metrics of the expressions found by different algorithms.

shows that our method can better handle multivariable symbolic regression problems than the current best algorithms in this area. For the Trigonometric dataset with $n = 8$ variables, both the GP and CVGP take more than 2 days to find the optimal expression. The reason is that there are too many open constants in the expressions in the population pool, making the optimization problem itself time-consuming to find the solution. This behavior is another indication that CVGP is stuck at some unfavorable experiment schedule.

Empirical Running Time Analysis. We summarize the running time analysis in Fig. 4. Our Racing-CVGP method takes less time than CVGP as well as the rest baselines. The main reason is early stop those unfavorable experiment schedules. See Appendix D for more figures.

Ablation Studies We collect the benchmark of different evaluation metrics in Fig. 5, *i.e.,* MSE and NMSE, during testing over the selected Trigonometric datasets. The RMSE and NRMSE evaluation metrics are available in Appendix D.

We further collect the time comparison between our Racing-CVGP and the CVGP with all the experiment schedules in Fig. 6. The quartiles of time distribution over 10 random expressions with 4 variables show that Our Racing-CVGP saves a great portion of the time compared with CVGP with all the schedules.

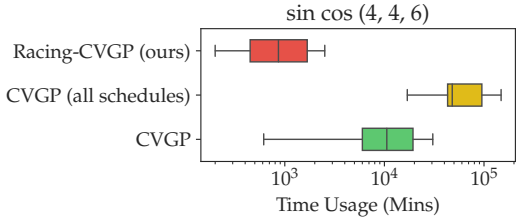


Figure 6: On a selected Trigonometric dataset, quartiles of the total running time of Racing-CVGP, CVGP, and CVGP with all the experiment schedules. Our Racing-CVGP saves a great portion of time compared with CVGP with all the schedules for expressions with $n = 4$ variables.

6 Conclusion

In this research, we propose Control Variable Genetic Programming (Racing-CVGP) for symbolic regression with many independent variables. Our Racing-CVGP can accelerate the regression process by discovering equations from promising experiment schedules and early stop those unfavorable experiment schedules. We evaluate Racing-CVGP on several synthetic and real-world datasets corresponding to true physics laws. We demonstrate that Racing-CVGP outperforms CVGP and a series of symbolic regressors which discover equations from fixed datasets.

7 Acknowledgments

We thank all the reviewers for their constructive comments. This research was supported by NSF grant CCF-1918327 and DOE – Fusion Energy Science grant: DE-SC0024583.

References

Abolafia, D. A.; Norouzi, M.; and Le, Q. V. 2018. Neural Program Synthesis with Priority Queue Training. *CoRR*, abs/1801.03526.

Attia, A.; and Ahmed, S. E. 2023. PyOED: An Extensible Suite for Data Assimilation and Model-Constrained Optimal Design of Experiments. *CoRR*, abs/2301.08336.

Balcan, M.; Dick, T.; Sandholm, T.; and Vitercik, E. 2018. Learning to Branch. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 353–362. PMLR.

Biggio, L.; Bendinelli, T.; Neitz, A.; Lucchi, A.; and Parascandolo, G. 2021. Neural Symbolic Regression that scales. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, 936–945. PMLR.

Cappart, Q.; Bergman, D.; Rousseau, L.; Prémont-Schwarz, I.; and Parjadis, A. 2022. Improving Variable Orderings of Approximate Decision Diagrams Using Reinforcement Learning. *INFORMS J. Comput.*, 34(5): 2552–2570.

Cava, W. G. L.; Orzechowski, P.; Burlacu, B.; de França, F. O.; Virgolin, M.; Jin, Y.; Kommenda, M.; and Moore, J. H. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *NeurIPS Datasets and Benchmarks*.

Cerrato, M.; Brugger, J.; Schmitt, N.; and Kramer, S. 2023. Reinforcement Learning for Automated Scientific Discovery. In *AAAI Spring Symposium on Computational Approaches to Scientific Discovery*.

Chen, C.; Luo, C.; and Jiang, Z. 2017. Elite bases regression: A real-time algorithm for symbolic regression. In *ICNC-FSKD*, 529–535. IEEE.

Chen, D.; Wang, Y.; and Gao, W. 2020. Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning. *Appl. Intell.*, 50(10): 3301–3317.

Dechter, R. 2019. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms, Second Edition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Derkinderen, V.; Heylen, E.; Martires, P. Z. D.; Kolb, S.; and Raedt, L. D. 2020. Ordering Variables for Weighted Model Integration. In *UAI*, volume 124 of *Proceedings of Machine Learning Research*, 879–888. AUAI Press.

Dette, H.; and Röder, I. 1997. Optimal discrimination designs for multifactor experiments. *The Annals of Statistics*, 25(3): 1161 – 1175.

Dubcáková, R. 2011. Eureqa: software review. *Genet. Program. Evolvable Mach.*, 12(2): 173–178.

Endres, S. C.; Sandrock, C.; and Focke, W. W. 2018. A simplicial homology algorithm for Lipschitz optimisation. *J. Glob. Optim.*, 72(2): 181–217.

Fletcher, R. 2000. *Practical methods of optimization*. John Wiley & Sons.

Fletcher, R.; and Reeves, C. M. 1964. Function minimization by conjugate gradients. *The computer journal*, 7(2): 149–154.

Fortin, F.-A.; De Rainville, F.-M.; Gardner, M.-A.; Parizeau, M.; and Gagné, C. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13: 2171–2175.

Gao, F.; and Han, L. 2012. Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Comput. Optim. Appl.*, 51(1): 259–277.

Guimerà, R.; Reichardt, I.; Aguilar-Mogas, A.; Massucci, F. A.; Miranda, M.; Pallarès, J.; and Sales-Pardo, M. 2020. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Science advances*, 6(5): eaav6971.

He, B.; Lu, Q.; Yang, Q.; Luo, J.; and Wang, Z. 2022. Taylor genetic programming for symbolic regression. In *GECCO*, 946–954. ACM.

Jiang, N.; and Xue, Y. 2023. Symbolic Regression via Control Variable Genetic Programming. In *ECML/PKDD*, Lecture Notes in Computer Science. Springer.

Kamienny, P.; d'Ascoli, S.; Lample, G.; and Charton, F. 2022. End-to-end Symbolic Regression with Transformers. In *NeurIPS*.

King, R. D.; Rowland, J.; Oliver, S. G.; Young, M.; Aubrey, W.; Byrne, E.; Liakata, M.; Markham, M.; Pir, P.; Soldatova, L. N.; Sparkes, A.; Whelan, K. E.; and Clare, A. 2009. The Automation of Science. *Science*, 324(5923): 85–89.

King, R. D.; Whelan, K. E.; Jones, F. M.; Reiser, P. G.; Bryant, C. H.; Muggleton, S. H.; Kell, D. B.; and Oliver, S. G. 2004. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971): 247–252.

Langley, P. 1977. BACON: A Production System That Discovers Empirical Laws. In *IJCAI*, 344. William Kaufmann.

Langley, P. 1979. Rediscovering Physics with BACON.3. In *IJCAI*, 505–507. William Kaufmann.

Langley, P. 1981. Data-driven discovery of physical laws. *Cognitive Science*, 5(1): 31–54.

Langley, P.; Bradshaw, G. L.; and Simon, H. A. 1981. BACON.5: The Discovery of Conservation Laws. In *IJCAI*, 121–126. William Kaufmann.

Lehman, J. S.; Santner, T. J.; and Notz, W. I. 2004. Designing computer experiments to determine robust control variables. *Statistica Sinica*, 571–590.

Lenat, D. B. 1977. The ubiquity of discovery. *Artificial Intelligence*, 9(3): 257–285.

Li, H.; Feng, G.; and Yin, M. 2020. On combining variable ordering heuristics for constraint satisfaction problems. *J. Heuristics*, 26(4): 453–474.

Matsubara, Y.; Chiba, N.; Igarashi, R.; and Ushiku, Y. 2022. SRSD: Rethinking Datasets of Symbolic Regression for Scientific Discovery. In *NeurIPS 2022 AI for Science: Progress and Promises*.

McConaghy, T. 2011. FFX: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, 235–260. Springer.

Mundhenk, T. N.; Landajuela, M.; Glatt, R.; Santiago, C. P.; Faissol, D. M.; and Petersen, B. K. 2021. Symbolic Regression via Deep Reinforcement Learning Enhanced Genetic Programming Seeding. In *NeurIPS*, 24912–24923.

Nagelkerke, N. J.; et al. 1991. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3): 691–692.

Ortiz-Bayliss, J. C.; Amaya, I.; Conant-Pablos, S. E.; and Terashima-Marín, H. 2018. Exploring the Impact of Early Decisions in Variable Ordering for Constraint Satisfaction Problems. *Comput. Intell. Neurosci.*, 2018: 6103726:1–6103726:14.

Petersen, B. K.; Landajuela, M.; Mundhenk, T. N.; Santiago, C. P.; Kim, S.; and Kim, J. T. 2021. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *ICLR*. OpenReview.net.

Razavi, S.; and Gamazon, E. R. 2022. Neural-Network-Directed Genetic Programmer for Discovery of Governing Equations. *CoRR*, abs/2203.08808.

Schmidt, M.; and Lipson, H. 2009. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923): 81–85.

Song, L.; Xue, K.; Huang, X.; and Qian, C. 2022a. Monte Carlo Tree Search based Variable Selection for High Dimensional Bayesian Optimization. In *NeurIPS*.

Song, W.; Cao, Z.; Zhang, J.; Xu, C.; and Lim, A. 2022b. Learning variable ordering heuristics for solving Constraint Satisfaction Problems. *Eng. Appl. Artif. Intell.*, 109: 104603.

Sun, F.; Liu, Y.; Wang, J.; and Sun, H. 2023. Symbolic Physics Learner: Discovering governing equations via Monte Carlo tree search. In *ICLR*. OpenReview.net.

Tohme, T.; Liu, D.; and Youcef-Toumi, K. 2023. GSR: A Generalized Symbolic Regression Approach. *Trans. Mach. Learn. Res.*, 2023.

Tsallis, C.; and Stariolo, D. A. 1996. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1-2): 395–406.

Udrescu, S.-M.; and Tegmark, M. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16).

Uy, N. Q.; Hoai, N. X.; O'Neill, M.; McKay, R. I.; and López, E. G. 2011. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program. Evolvable Mach.*, 12(2): 91–119.

Virgolin, M.; Alderliesten, T.; and Bosman, P. A. N. 2019. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *GECCO*, 1084–1092. ACM.

Virgolin, M.; and Pissis, S. P. 2022. Symbolic Regression is NP-hard. *Transactions on Machine Learning Research*.

Wales, D. J.; and Doye, J. P. 1997. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28): 5111–5116.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8: 229–256.

Yang, M.; and Stufken, J. 2012. Identifying locally optimal designs for nonlinear models: A simple extension with profound consequences. *The Annals of Statistics*, 40(3): 1665 – 1681.

A Implementation

Please find our code repository. It contains 1) the implementation of our Racing-CVGP method, 2) the list of datasets, and 3) the implementation of several baseline algorithms.

B Genetic Programming Algorithm in Racing-CVGP

For the `FreezeEquation` function used in Algorithm 1, we use Figure 7 to demonstrate the output. The `FreezeEquation` function will reduce the size of candidate nodes to be edited in the GP algorithms and increase the probability of finding expression trees with close-to-zero fitness scores.

(a) controlled variable $\mathbf{x}_c = \{x_2, x_3\}$. **(b)** controlled variable $\mathbf{x}_c = \{x_1, x_2\}$.

Figure 7: Visualization of the `FreezeEquation` function. 0 implies the node is non-editable and 1 implies the node is editable, by the GP algorithm. The `FreezeEquation` function will increase the probability of finding expression trees with close-to-zero fitness scores.

The pipeline of Genetic Programming in our Racing-CVGP framework is presented in Algorithm 2. It is a minimally modified genetic programming algorithm for symbolic regression.

For the `Mutate` step, the algorithm will apply one of the operations for the chosen expression tree:

1. Find a leaf node that is not frozen and then replace the node with a generate a full expression tree of maximum depth involving variables only in $\mathbf{x} \setminus \mathbf{x}_c$.
2. Find a node and replace it with a node of the same arity. Here arity is the number of operands taken by an operator. For example, the arity of binary operators $\{+, -, \times, \div\}$ is 2 and the arity of unary operators $\{\sin, \cos, \log, \exp\}$ is 1.
3. Inserts a node at a random position, and the original subtree at the location becomes one of its subtrees.
4. Delete a node that is not frozen, use one of its children to replace its position.

For the `Mate` step, we will pick two expressions ϕ_l, ϕ_j from the population pool $\mathcal{P}$ that has the same control variables $\mathbf{x}_{c,l} = \mathbf{x}_{c,j}$. Then we exchange two randomly chosen subtrees in the expressions. Because applying mating over two expressions with different control variables does not necessarily result in two better expressions.

Algorithm 2: GP($\mathcal{P}$, DataOracle, K, M, #Gen, #Hof, P_{mu}, P_{ma}, O_p)

Input: Initial GP Pool $\mathcal{P}$; DataOracle; # control variable trials K; GP pool size N_p; # of generations #Gen; #expressions in hall-of-fame set #Hof; mutation node library O_p.
Parameters: The GP pool and the updated hall of fame set.
Parameters: mutate probability P_{mu}; mate probability P_{ma};

for $i \leftarrow 1$ *to* #Gen **do**
  $\mathcal{P}_{new} \leftarrow \emptyset$;
  for $\langle \phi_{new}, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ **do**
    if with probability P_{mu} **then** ▷ mutation
      $\phi_{new} \leftarrow \texttt{Mutate}(\phi_{new}, O_p, \mathbf{x} \setminus \mathbf{x}_c)$;
      $\{\mathcal{D}_k\}_{k=1}^K \leftarrow \texttt{DataOracle}(\phi_{new}, \mathbf{x}_c, K)$;
      $\mathbf{o}, \mathbf{c} \leftarrow \texttt{Optimize}(\phi_{new}, \{\mathcal{D}_k\}_{k=1}^K)$;
    $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\langle \phi_{new}, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle\}$;
  $\mathcal{P} \leftarrow \mathcal{P}_{new}$; $\mathcal{P}_{new} \leftarrow \emptyset$
  for $\langle \phi_l, \pi_l, \mathbf{x}_{c,l} \rangle, \langle \phi_j, \pi_j, \mathbf{x}_{c,j} \rangle \in \mathcal{P}$ **do** ▷ mating
    if with probability P_{ma} and $\mathbf{x}_{c,l} = \mathbf{x}_{c,j}$ **then** ▷ pick two expressions with the same $\mathbf{x}_c$
      $\phi_l, \phi_j \leftarrow \texttt{Mate}(\phi_l, \phi_j)$;
      $\{\mathcal{D}_k\}_{k=1}^K \leftarrow \texttt{DataOracle}(\phi_l, \mathbf{x}_{c,l}, K)$;
      $\mathbf{o}_l, \mathbf{c}_l \leftarrow \texttt{Optimize}(\phi_l, \{\mathcal{D}_k\}_{k=1}^K)$;
      $\{\mathcal{D}_k\}_{k=1}^K \leftarrow \texttt{DataOracle}(\phi_j, \mathbf{x}_{c,j}, K)$;
      $\mathbf{o}_j, \mathbf{c}_j \leftarrow \texttt{Optimize}(\phi_j, \{\mathcal{D}_k\}_{k=1}^K)$;
    $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\langle \phi_l, \mathbf{o}_l, \mathbf{c}_l, \pi_l, \mathbf{x}_{c,l} \rangle, \langle \phi_j, \mathbf{o}, \mathbf{c}_j, \pi_j, \mathbf{x}_{c,j} \rangle\}$;
  $\mathcal{H} \leftarrow \texttt{TopK}(\mathcal{P}_{new} \cup \mathcal{H}, K = \texttt{\#Hof})$; ▷ Update the hall of fame set
return GP pool and hall-of-fame $\mathcal{P}_{new}, \mathcal{H}$.

C Experiment Settings

Dataset Configuration

Livermore2 Dataset The list of Livermore2 datasets is summarized at[2]. In Tables 3, 4, 5, we details the exact equation of Livermore2 (Petersen et al. 2021). The operator set for each expression is available in the codebase.

The list of Feynamn datasets is collected from[3]. In Tables 6, 7. We only use a subset of the expressions in the original Feynman dataset. The challenging part for this dataset is the ranges of input variables vary greatly. For example, in one equation with ID "ICh34Eq8" the ranges of all the variables are:

$$x_1 \in (10^{-11}, 10^{-9}), x_2 \in (10^5, 10^7), x_3 \in (10, 10^3), x_4 \in (10^9, 10^{11}) \tag{2}$$

In comparison, the input ranges of the Livermore2 dataset are $x_i \in (0.01, 10)$. The operator set for each expression is available in the codebase.

Evaluation Metrics

We mainly consider two evaluation criteria for the learning algorithms tested in our work: 1) the goodness-of-fit measure and 2) the total running time of the learning algorithms. The goodness-of-fit indicates how well the learning algorithms perform in discovering unknown symbolic expressions. Given a testing dataset $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ generated from the ground-truth expression ϕ, we measure the goodness-of-fit of a predicted expression $\bar{\phi}$, by evaluating the mean-squared-error (MSE) and normalized-mean-squared-error (NMSE):

$$\begin{aligned} \text{MSE} &= \frac{1}{n}\sum_{i=1}^n (y_i - \bar{\phi}(\mathbf{x}_i))^2, \qquad \text{NMSE} = \frac{\frac{1}{n}\sum_{i=1}^n (y_i - \bar{\phi}(\mathbf{x}_i))^2}{\sigma_y^2} \\ \text{RMSE} &= \sqrt{\frac{1}{n}\sum_{i=1}^n (y_i - \bar{\phi}(\mathbf{x}_i))^2}, \qquad \text{NRMSE} = \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^n (y_i - \bar{\phi}(\mathbf{x}_i))^2} \end{aligned} \tag{3}$$

where the empirical variance $\sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^n \left(y_i - \frac{1}{n}\sum_{i=1}^n y_i\right)^2}$. Note that the coefficient of determination (R^2) metric (Nagelkerke et al. 1991; Cava et al. 2021) is equal to $(1 - \text{NMSE})$ and therefore omitted in the experiments.

[2] https://github.com/brendenpetersen/deep-symbolic-optimization/blob/master/dso/dso/task/regression/benchmarks.csv

[3] https://github.com/omron-sinicx/srsd-benchmark/blob/main/datasets/feynman.py

Table 3: Detailed equation in Livermore2 datasets (part-1).

	Livermore2 ($n = 4$)	
Eq. ID	n	Expression
Vars4-1	4	$x_1 - x_2x_3 - x_2 - x_4$
Vars4-2	4	$x_1\sqrt{x_2}x_4/x_3$
Vars4-3	4	$2x_1 + x_4 - 0.01 + x_3/x_2$
Vars4-4	4	$x_1 - x_4 - (-x_1 + \sin(x_1))^4/(x_1^8x_2^2x_3^2)$
Vars4-5	4	$x_1 + \sin(x_2/(x_1x_2^2x_4^2(-3.22x_2x_4^2 + 13.91x_2x_4 + x_3)/2 + x_2))^2$
Vars4-6	4	$(-x_1 - 0.54\exp(x_1 sqrt(x_4 + \cos(x_2))\exp(-2x_1)))/x_3$
Vars4-7	4	$x_1 + \cos(x_2/\log(x_2^2x_3 + x_4))$
Vars4-8	4	$x_1(x_1 + x_4 + \sin((-x_1\exp(\exp(x_3)) + x_2)/(-4.47x_1^2x_3 + 8.31x_3^3 + 5.27x_3^2))) - x_1$
Vars4-9	4	$x_1 - x_4 + \cos(x_1(x_1 + x_2)(x_1^2x_2 + x_3) + x_3)$
Vars4-10	4	$x_1 + (x_1(x_4 + (\sqrt{x_2} - \sin(x_3))/x_3))^{1/4}$
Vars4-11	4	$2x_1 + x_2(x_1 + \sin(x_2x_3)) + \sin(2/x_4)$
Vars4-12	4	$x_1x_2 + 16.97x_3 - x_4$
Vars4-13	4	$x_4(-x_3 - \sin(x_1^2 - x_1 + x_2))$
Vars4-14	4	$x_1 + \cos(x_2^2(-x_2 + x_3 + 3.23) + x_4)$
Vars4-15	4	$x_1(x_2 + \log(x_3 + x_4 + \exp(x_2^2) - 0.28/x_1)) - x_3 - \frac{x_4}{2x_1x_3}$
Vars4-16	4	$x_3(-x_4 + 1.81/x_3) + \sqrt{x_2(-x_1^2\exp(x_2) - x_2)} - 2.34x_4/x_1$
Vars4-17	4	$x_1^2 - x_2 - x_3^2 - x_4$
Vars4-18	4	$x_1 + \sin(2x_2 + x_3 - x_4\exp(x_1) + 2.96\sqrt{-0.36x_2^3 + x_2x_3^2 + 0.94} + \log((-x_1 + x_2)\log(x_2)))$
Vars4-19	4	$(x_1^3x_2 - 2.86x_1 + x_4)/x_3$
Vars4-20	4	$x_1 + x_2 + 6.21 + 1/(x_3x_4 + x_3 + 2.08)$
Vars4-21	4	$x_1(x_2 - x_3 + x_4) + x_4$
Vars4-22	4	$x_1 - x_2x_3 + x_2\exp(x_1) - x_4$
Vars4-23	4	$-x_1/x_2 - 2.23x_2x_3 + x_2 - 2.23x_3/\sqrt{x_4} - 2.23\sqrt{x_4} + \log(x_1)$
Vars4-24	4	$-4.81\log(\sqrt{x_1\sqrt{\log(x_1(x_1x_2 + x_1 + x_4 + log(x_3)))}})$
Vars4-25	4	$0.38 + (-x_1/x_4 + cos(2x_1x_3/(x_4(x_1 + x_2x_3)))/x_4)/x_2$

Baseline Implementation

Racing-CVGP Our method is implemented on top of the **GP** following Algorithm 2. See the codebase for more details.
GP The implementation is based on the version in "baselines" of DSO package[4]. However, we re-implemented the code following the concept of their package.
CVGP The implementation is available at[5].
Eureqa This algorithm is currently maintained by the DataRobot webiste[6]. We use the python API provided [7] to send the training dataset to the DataRobot website and collect the predicted expression after 30 minutes. This website only allows us to execute their program under a limited budget. Due to budgetary constraints, we were only able to test the datasets for the noiseless settings. For the Eureqa method, the fitness measure function is negative RMSE. We generated large datasets of size 10^5 in training each benchmark.
DSR, PQT, GPMeld These algorithms are evaluated based on an implementation in[8]. For every ground-truth expression, we generate a dataset of sizes 10^5 training samples. Then we execute all these baselines on the dataset with the configurations listed in Table 8.

The official implementation of Symbolic Physics Learner (SPL) (Sun et al. 2023)[9] does not support open constants. Thus SPL is not considered in this research.

For the four baselines (*i.e.*, PQT, VPG, DSR, GPMeld), the reward function is INV-NRMSE, which is defined as $\frac{1}{1+\text{NRMSE}}$.

Optimizers

We consider several optimizers CG (Fletcher and Reeves 1964) Nelder-Mead (Gao and Han 2012), BFGS (Fletcher 2000), Basin Hopping (Wales and Doye 1997), SHGO (Endres, Sandrock, and Focke 2018), Dual Annealing (Tsallis and Stariolo

[4]https://github.com/DEAP/deap
[5]https://github.com/jiangnanhugo/cvgp
[6]https://docs.datarobot.com/en/docs/modeling/analyze-models/describe/eureqa.html
[7]https://pypi.org/project/datarobot/
[8]https://github.com/brendenpetersen/deep-symbolic-optimization
[9]https://github.com/isds-neu/SymbolicPhysicsLearner

Table 4: Detailed equation in Livermore2 datasets (part-2).

	Livermore2 ($n = 5$)	
Eq. ID	n	Expression
Vars5-1	5	$-x_1 + x_2 - x_3 + x_4 - x_5 - 4.75$
Vars5-2	5	$x_3 \left(x_1 + x_5 + \frac{0.27}{(x_3^2 + \frac{(x_2+x_4)}{(x_1 x_2 + x_2)})} \right)$
Vars5-3	5	$2x_1x_2x_3 + x_5 - \sin(x_1 \log(x_2) - x_1 + x_4)$
Vars5-4	5	$x_2 + x_3x_4 + x_5^2 + \sin(x_1)$
Vars5-5	5	$x_5 + 0.36\sqrt{(\log(x_1x_2 + x_3 + \log(x_2 + x_4)))}$
Vars5-6	5	$x_1x_4 + x_1 + x_2 + x_5 + \sqrt{(0.08x_1/(x_3x_5) + x_3)}$
Vars5-7	5	$x_1x_5 + \sqrt{(x_1x_2\cos(x_1) - x_1/(x_2 + x_3 + x_4 + 8.05))}$
Vars5-8	5	$\sqrt{(x_2)}x_3 - x_4 - 0.07(x_1 + (x_1 - x_2)\sqrt{(x_2 - 0.99)})\cos(x_5)$
Vars5-9	5	$x_1(x_3 + (x_1 + x_2)/(x_2x_4 + x_5))$
Vars5-10	5	$x_1/(x_4(-0.25x_1x_3x_4 + x_2 - 8.43x_4x_5)\sin(x_3 + \log(x_2))) + x_4x_5$
Vars5-11	5	$-x_4^2 + \sqrt{\frac{x_1(x_3+x_5)-x_2+x_5}{x_3}} + 0.47\sqrt{x_3 \frac{x_1 - \sqrt{x_2} + x_2}{x_2}}$
Vars5-12	5	$x_1 \left(x_2 - \frac{1}{x_3(x_4+x_5)} \right)$
Vars5-13	5	$\sqrt{(x_1(x_5(x_2 - 1.52) - \cos(4.03x_3 + x_4)))}$
Vars5-14	5	$-x_1/(x_2x_5) + \cos(x_1x_3x_4\exp(-x_2))$
Vars5-15	5	$-x_4 + \log(x_1/\log(11.06x_2x_5^2) + x_3) - \cos(x_2 + x_5 + \sqrt{(x_2x_5)})$
Vars5-16	5	$x_2 + 0.33x_5(x_1/(x_1^2 + x_2) + x_3x_4^(3/2))$
Vars5-17	5	$x_1 - \sin(x_2) + \sin(x_3) - \cos(-x_2 + \sqrt{(x_4)} + x_5) + 0.78$
Vars5-18	5	$x_1x_2 - x_4 - (\sqrt{(x_3^2/(x_1(x_3 + x_4)))} - 1.13/x_3)/x_5$
Vars5-19	5	$4.53x_1x_2 + x_1 - x_1\cos(\sqrt{(x_2)})/x_2 - x_3 - x_4 - x_5$
Vars5-20	5	$-\exp(x_1 + x_5) + \sin(x_1 - 4.81)/(0.21(x_5 - \log(x_3 + x_4) - \exp(x_5))/x_2)$
Vars5-21	5	$\sqrt{(x_4)}(2x_1 + \cos(x_1(x_3x_4\exp(x_1x_2) + x_3 - \log(x_3) - 3.49))/x_5)$
Vars5-22	5	$-x_1 - x_2 + x_3 + \sqrt{(x_1 - x_2(\sin(x_3) - \log(x_1x_5/(x_2^2 + x_4))/x_4))} - 0.73$
Vars5-23	5	$x_1(x_2/(x_3 + \sqrt{(x_2(x_4 + x_5))}(-x_3 + x_4)) - x_5)$
Vars5-24	5	$-x_2x_5 + \sqrt{(x_1 + x_2(-x_1 + x_4\cos(\sqrt{x_3} + x_3) - \frac{x_2 + 7.84x_3^2x_5}{x_5}) + \frac{x_2}{x_3}}$
Vars5-25	5	$x_1 + \log(x_1(-3.57x_1^2x_2 + x_1 + x_2 + x_3\log(-x_1x_4\sin(x_3)/x_5 + x_3)))$

1996). The list of local and global optimizers shown in Figure 12 are from Scipy library[10].

Hyper-parameter Configuration

We list the major hyper-parameter setting for all the algorithms in Table 8. Note that if we use the default parameter settings, the GPMeld algorithm takes more than 1 day to train on one dataset. Because of such slow performance, we cut the number of genetic programming generations in GPMeld by half to ensure fair comparisons with other approaches.

[10]https://docs.scipy.org/doc/scipy/reference/optimize.html#global-optimization

Table 5: Detailed equation in other small-scale datasets (part-3). n stands for the number of maximum variables.

		Livermore2 ($n = 6$)
Eq. ID	n	Expression
Vars6-1	6	$x_1 - x_6 + (x_1 + x_4 + x_5)\sqrt{(x_1^2 + x_2 - x_3)}$
Vars6-2	6	$x_1(2x_2 + x_2/x_3 + x_4 + \log(x_1 x_5 x_6))$
Vars6-3	6	$\sqrt{(x_2 + x_5 - x_6 + x_3^4 x_4^4/(x_1 x_2^4))}$
Vars6-4	6	$x_1(x_2(x_1^2 + x_1) - x_2 + x_3^2 - x_3 - x_5 - x_6 - \sin(x_4) - \cos(x_4))^2$
Vars6-5	6	$x_2\sqrt{(x_1 x_2)(x_1 x_3 - x_3 - x_4)} + x_5 + x_6$
Vars6-6	6	$(x_1/(x_2 x_3 + \log(\cos(x_1))^2) - x_2 x_4 + \sin((x_2 x_4 + x_5)/x_6) + \cos(x_3))\log(x_1)$
Vars6-7	6	$x_1\sqrt{(x_1 - x_6^2 + \sin((x_1\exp(-x_2) - x_4(x_2 + x_3^2))/(x_2 + x_5)))}$
Vars6-8	6	$x_1 + x_2^2 + 0.34 x_3 x_5 - x_4 + x_6$
Vars6-9	6	$x_4(x_1 + \exp(13.28 x_3^2 x_6 - x_5^2\log(x_2^4))/(x_1 x_3 - x_2^2 + x_2 - x_6 - \log(x_3)))$
Vars6-10	6	$x_1 + 61.36 x_2^6 + x_2/(x_1 x_3(x_4 - \cos(x_4(2x_1 x_2 x_6/x_5 + x_5))))$
Vars6-11	6	$(x_1 + x_1/(x_2 + x_4(8.13 x_1^2 x_6 + x_1 x_2 x_3 + 2x_2 + x_5 + x_6)))^2$
Vars6-12	6	$(\sqrt{2}\sqrt{(x_1)} - x_2 - x_3/\sqrt{(x_4(8.29 x_1 x_3^2 + x_1 x_5) + x_4 + x_6)})/x_6$
Vars6-13	6	$x_1 + x_5 + 0.21\sqrt{(x_1/(x_2^2 x_3^2\sqrt{(x_6)}(\sqrt{(x_3)} + x_3 + 2x_6 + (x_2 + x_4 + x_5)/x_5)))}$
Vars6-14	6	$-2.07 x_6 + \log(x_2 - x_6 - \sqrt{(x_3(x_5 + \log(-x_1 + x_5 + 1))/x_4)})$
Vars6-15	6	$x_1(x_1 + \cos(x_2^2 x_3 x_4(x_5 - 0.43 x_6^2)))/x_4$
Vars6-16	6	$-\sqrt{(x_1)} - x_1 + x_2 - x_4 - x_5 - \sqrt{(x_6/x_3)} - 3.26$
Vars6-17	6	$x_1/(x_2 x_4(-x_5 + \log(x_6^2\cos(2x_2 + x_3^2 - x_4)^2))(129.28 x_1^2 x_2^2 + x_3))$
Vars6-18	6	$\sqrt{x_5}(2x_1 + \cos(x_1(x_3 x_4\exp(x_1 x_2) + x_3 - \log(x_3) - 3.49))/x_6)$
Vars6-19	6	$x_1 + x_2 + x_3 + 0.84\sqrt{(-x_3 x_6 + x_4 - x_5 + \sqrt{((x_2 + \log(x_3 + \exp(x_2)))/(x_2 - x_4))})}$
Vars6-20	6	$(x_1 - 0.97 x_1/(x_5 - x_6(x_1^(3/2)x_4 + x_6)) - x_2 + x_3 + \sin(x_1^2)/x_1)^2$
Vars6-21	6	$x_1 + x_3 + (x_1 + \sin(-3.47 x_2\log(x_6)/x_5 + x_4 + 25.56\exp(x_5^2)/x_2)^2)\sin(x_2)$
Vars6-22	6	$x_1 + (x_4 + \sin(-0.22(x_3 - x_4 + 1.0))\cos(x_6))\cos(x_2 + 2.27 x_5)$
Vars6-23	6	$x_1 + x_4 + \log(x_1^2 + x_1(-x_6 + 1.88\sqrt{(0.71 x_1 + x_2 + 0.28(x_3 - x_4/x_5))}))$
Vars6-24	6	$-0.59(1.42(0.24 x_2 + \sqrt{x_3}/(x_6\sqrt{(-x_4 + x_5)}))^(1/4) + \sin(x_1))/x_6$
Vars6-25	6	$x_1 - x_2^2 - x_3 + x_5\cos(x_3) + x_5 + x_6 - 2.19\sqrt{(-x_3 - 0.44/x_4)}$

D Extra Experimental Analysis

Impact of Experiment Schedulese: See Figure 8,9

Empirical Running Time: See Figure 10

Impact of Evaluation Metrics: See Figure 11

Table 6: Detailed equations in Feynman datasets ($n = 4$). n stands for the number of maximum variables.

	Feynman ($n = 4$)	
Eq. ID	n	Expression
I.8.14	4	$\sqrt{(x_0 - x_1)^2 + (x_2 - x_3)^2}$
I.13.4	4	$0.5x_0(x_1^2 + x_2^2 + x_3^2)$
I.13.12	4	$6.6743e - 11x_0x_1(-1/x_3 + 1/x_2)$
I.18.4	4	$(x_0x_1 + x_2x_3)/(x_0 + x_2)$
I.18.16	4	$x_0x_1x_2\sin(x_3)$
I.24.6	4	$0.25x_0x_3^2(x_1^2 + x_2^2)$
I.29.16	4	$\sqrt{x_0^2 + 2x_0x_1\cos(x_2 - x_3) + x_1^2}$
I.32.17	4	$0.0035\pi x_0^2x_1^2x_2^4/(x_2^2 - x_3^2)^2$
I.34.8	4	$x_0x_1x_2/x_3$
I.40.1	4	$x_0\exp(-7.10292768111229e + 23x_1x_2/x_3)$
I.43.16	4	$x_0x_1x_2/x_3$
I.44.4	4	$1.38e - 23x_0x_1\log(x_2/x_3)$
I.50.26	4	$x_0(x_3\cos(x_1x_2)^2 + \cos(x_1x_2))$
II.11.20	4	$2.41e + 22x_0x_1^2x_2/x_3$
II.34.11	4	$x_0x_1x_2/(2x_3)$
II.35.18	4	$x_0/(\exp(7.24e + 22x_1x_2/x_3) + \exp(-7.24e + 22x_1x_2/x_3))$
II.35.21	4	$x_0x_1\tanh(7.24e + 22x_1x_2/x_3)$
II.38.3	4	$x_0x_1x_2/x_3$
III.10.19	4	$x_0\sqrt{x_1^2 + x_2^2 + x_3^2}$
III.14.14	4	$x_0(\exp(7.24e + 22x_1x_2/x_3) - 1)$
III.21.20	4	$-x_0x_1x_2/x_3$
BONUS.1	4	$3.32e - 57x_0^2x_1^2/(x_2^2\sin(x_3/2)^4)$
BONUS.3	4	$x_0(1 - x_1^2)/(x_1\cos(x_2 - x_3) + 1)$
BONUS.11	4	$4x_0\sin(x_1/2)^2\sin(x_2x_3/2)^2/(x_1^2\sin(x_3/2)^2)$
BONUS.19	4	$-1872855580.36049(8.07e + 33x_0/x_1^2 + 8.98e + 16x_2^2(1 - 2x_3))/\pi$

Table 7: Detailed equations in Feynman datasets ($n = 5$). n stands for the number of maximum variables.

	Feynman ($n = 5$)	
Eq. ID	n	Expression
I.12.11	5	$x_0(x_1 + x_2x_3\sin(x_4))$
II.2.42	5	$x_0x_3(x_1 - x_2)/x_4$
II.6.15a	5	$84707476846.623x_0x_1\sqrt{(x_3^2 + x_4^2)}/(\pi x_2^5)$
II.11.3	5	$x_0x_1/(x_2(x_3^2 - x_4^2))$
II.11.17	5	$x_0(7.24e + 22x_1x_2\cos(x_3)/x_4 + 1)$
II.36.38	5	$7.24e + 22x_0x_1/x_2 + 9.10e + 16x_0x_3x_4/x_2$
III.9.52	5	$1.21e + 34\pi x_0x_1\sin(x_2(x_3 - x_4)/2)^2/(x_2(x_3 - x_4)^2)$
bonus.4	5	$\sqrt{2}\sqrt{(x_1 - x_2 - x_3^2/(2x_0x_4^2))}/x_0$
bonus.12	5	$x_0(-x_0x_2^3x_4/(x_2^2 - x_4^2)^2 + 4pix_1x_3x_4)/(4\pi x_1x_2^2)$
bonus.13	5	$x_1/(4\pi x_0\sqrt{x_2^2 - 2x_2x_3\cos(x_4) + x_3^2})$
bonus.14	5	$x_0(-x_2 + x_3^3(x_4 - 1)/(x_2^2(x_4 + 2)))\cos(x_1)$
bonus.16	5	$x_1x_4 + 8.98e + 16\sqrt{x_3^2 + 1.11e - 17(x_0 - x_1x_2)^2}$

Impact of Optimizers

Here we study the impact of using global and local optimizers over those non-convex expressions. With the introduction of control variable experiments, fitting the open constants in the expressions is solving more and more non-convex optimization problems.

For those expressions in the populations, an optimizer might find a set of open constants for a structurally correct expression with large NMSE errors, resulting in a low ranking in the whole population. Such structurally correct expressions will not be included after several rounds of genetic operations.

We summarize the experimental result in Figure 12. In general, the list of global optimizers (SHGO, Direct, Basin-Hopping, and Dual-Annealing) fits better for the open constants than the list of local optimizers but they take significantly more CPU

Table 8: Major hyper-parameters settings for all the algorithms considered in the experiment.

	Racing-CVGP	GP	DSR	PQT	GPMeld	Eureqa
Reward function	NegMSE	NegMSE	InvNRMSE	InvNRMSE	InvNRMSE	NegRMSE
Training set size	$25,600$	$25,600$	$50,000$	$50,000$	$50,000$	$50,000$
Testing set size	256	$25,600$	256	256	256	256
Batch size	256	256	1024	1024	1024	N/A
#CPUs for training	1	1	4	4	4	1
ϵ-risk-seeking policy	N/A	0.02	0.02	0.02	N/A	N/A
#genetic generations	100	100	N/A	N/A	60	10,000
#Hall of fame	10	10	25	25	25	N/A
Mutation Probability	0.5	0.5	0.5	N/A	N/A	N/A
Mating Probability	0.5	0.5	0.5	N/A	N/A	N/A
training time (hours)	$\sim$0.5	$\sim$0.5	$\sim$0.5	$\sim$0.5	$\sim$6	$\sim$0.5

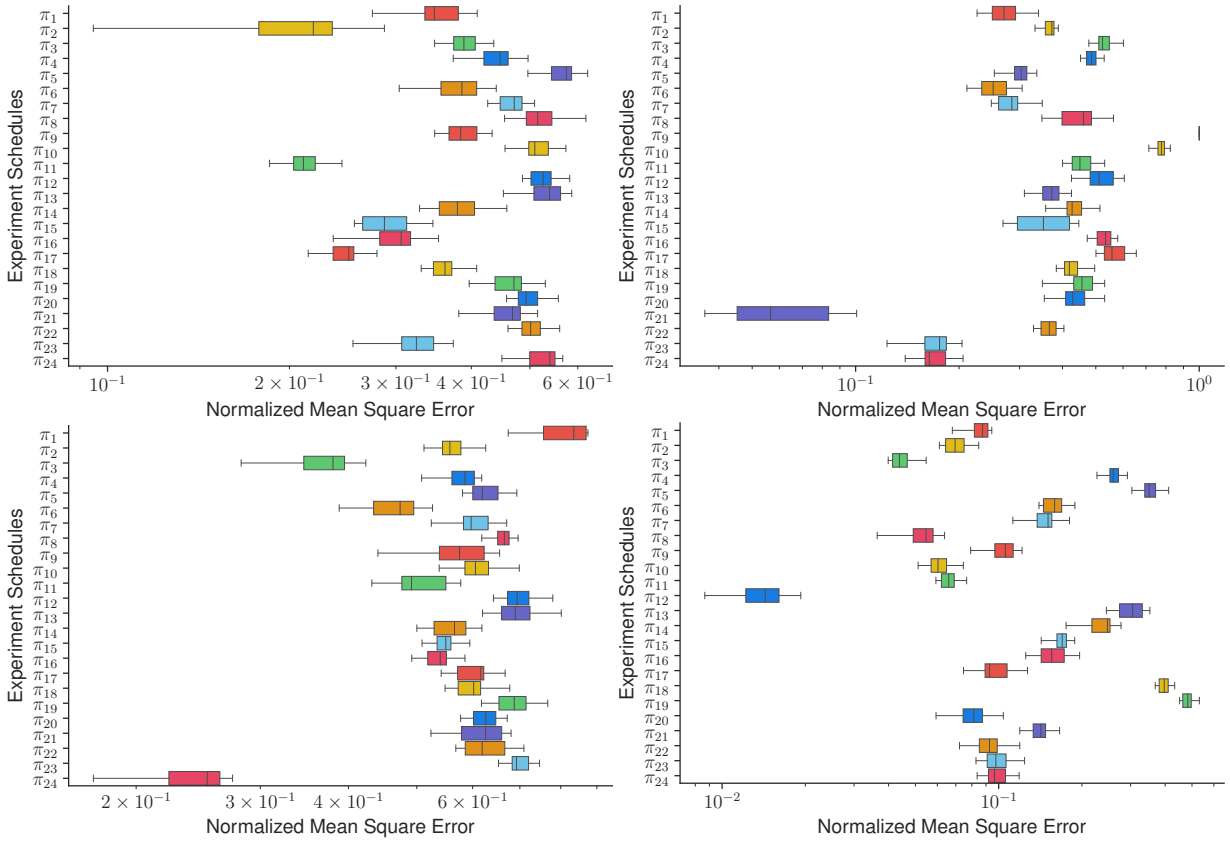


Figure 8: Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming, on the Trigonometric $(4, 4, 6)$ with operator set $\{+, -, \times, \div, \sin, \cos\}$ dataset. For the discovery of 10 different expressions with 4 variables, there always exists a better experiment schedule than the default one (*i.e.*, π_1), in terms of normalized mean square error.

resources and time for computations.

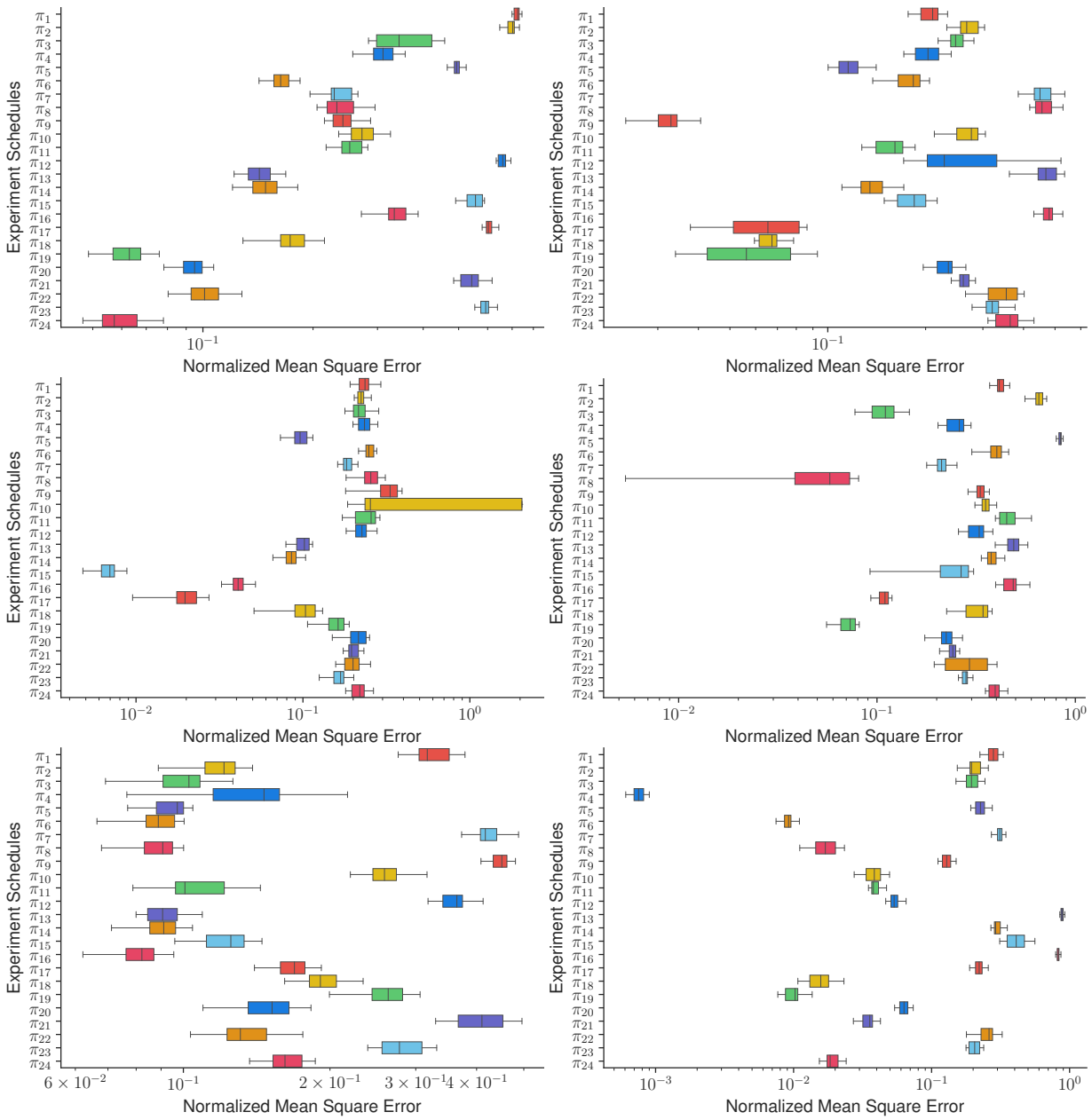


Figure 9: (Continued) Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming. For the discovery of expression with 4 variables, there always exists a better experiment schedule than the default one (*i.e.*, π_1), in terms of normalized mean square error.

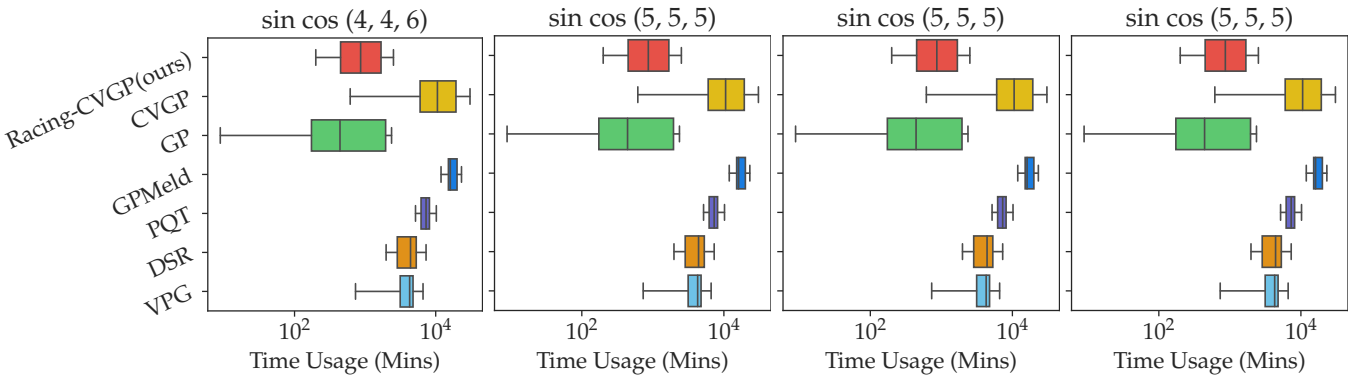


Figure 10: On Trigonometric datasets, quartiles of the total running time of all the methods. Our Racing-CVGP method takes less time than CVGP by early stopping those unfavorable experiment schedules.

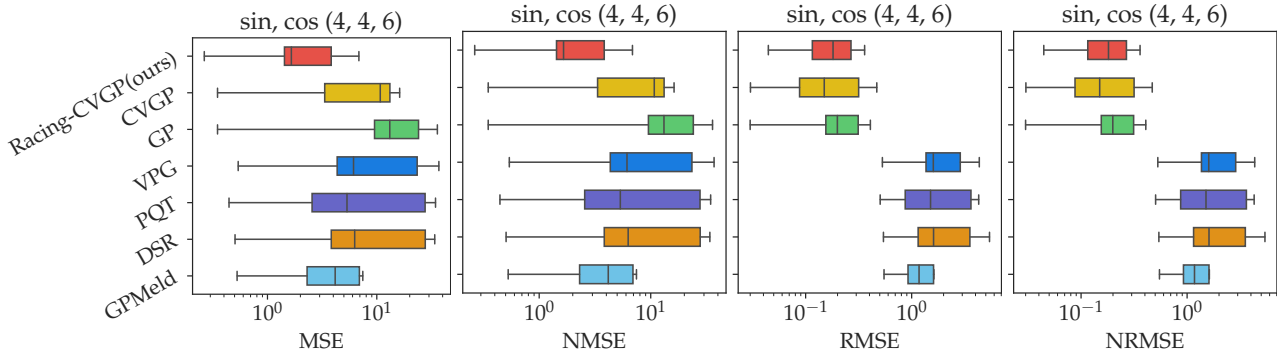


Figure 11: On selected Trigonometric datasets, MSE, NMSE, RMSE, and NRMSE evaluation metrics of the expressions found by different algorithms.

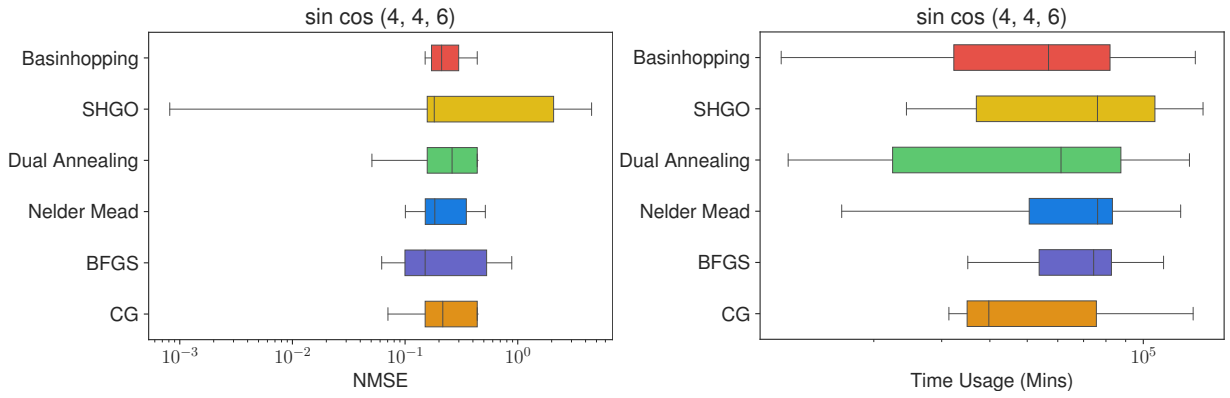


Figure 12: Impact of optimizers on finding the values of open constants for non-convex expressions. Over 10 randomly generated expressions involving 4 variables, SHGO can find better solutions (in terms of NMSE metric) than local optimizers (including Nelder-Mead, BFGS, CG), while the time taken by SHGO is higher than local optimizers.