LEARNING SYMBOLIC EXPRESSIONS: MIXED-INTEGER FORMULATIONS, CUTS, AND HEURISTICS*

JONGEUN KIM†, SVEN LEYFFER‡, AND PRASANNA BALAPRAKASH†

Abstract. In this paper we consider the problem of learning a regression function without assuming its functional form. This problem is referred to as symbolic regression. An expression tree is typically used to represent a solution function, which is determined by assigning operators and operands to the nodes. The symbolic regression problem can be formulated as a nonconvex mixed-integer nonlinear program (MINLP), where binary variables are used to assign operators and nonlinear expressions are used to propagate data values through nonlinear operators such as square, square root, and exponential. We extend this formulation by adding new cuts that improve the solution of this challenging MINLP. We also propose a heuristic that iteratively builds an expression tree by solving a restricted MINLP. We perform computational experiments and compare our approach with a mixed-integer program-based method and a neural-network-based method from the literature.

Key words. Symbolic regression, mixed-integer nonlinear programming, local branching heuristic, expression tree.

AMS subject classifications. 90C11, 90C26, 90C27, 90C30.

1. Introduction.

We consider the problem of learning symbolic expressions, which is referred to as *symbolic regression*. Symbolic regression is a form of regression that learns functional expressions from observational data. Unlike traditional regression, symbolic regression does not assume a fixed functional form but instead learns the functional relationship and its constants. Given observational data in terms of independent variables, $x_i \in \mathbb{R}^d$, and dependent variables (function values), $z_i \in \mathbb{R}$, for $i = 1, \ldots, n_{\text{data}}$, symbolic regression aims to find the best functional form that maps the x-values to the z-values by solving the following optimization problem:

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n_{\text{data}}} \left(f(x_i) - z_i\right)^2, \tag{1.1}$$

where $\mathcal{F}$ is the space of functions from which f is chosen. We note that other loss functions involving general norms are also possible and that, in general, problem (1.1) is an infinite-dimensional optimization problems. Various applications of symbolic regression have been presented in different fields including materials science [21], fluid systems [7], physics [16, 19, 20], and civil engineering [17].

Symbolic regression is especially useful when we do not know the precise functional form that relates the independent variables x to the dependent variables z or when we wish to exploit the freedom of optimally choosing the functional form. Given data $(x_i, z_i) \in \mathbb{R}^{d+1}$, $i = 1, \ldots, n_{\text{data}}$ and a set of mathematical operators, symbolic regression searches for a best-fit mathematical expression as a combination of these operators, independent variables, and constant. Given suitable restrictions on the function space $\mathcal{F}$ (e.g., a finite set of mathematical operators), we can formulate (1.1) as a nonconvex mixed-integer nonlinear program (MINLP). In this paper, we describe new cutting planes to enhance the MINLP formulation, develop new heuristics to solve

*ARGONNE NATIONAL LABORATORY, PREPRINT ANL/MCS-P9445-0212.

†Industrial and Systems Engineering, University of Minnesota Twin Cities, Minneapolis, MN 55455, USA (kim00623@umn.edu).

‡Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA (leyffer@mcs.anl.gov, pbalapra@anl.gov).

the resulting MINLP, and demonstrate the effectiveness of our approach on a broad set of test problems.

The remainder of this paper is organized as follows. In the rest of this section, we review some background material on expression trees and the literature on symbolic regression. In section 2, we introduce a MINLP formulation that improves the existing formulations by adding new sets of cutting planes. In section 3, we introduce the sequential tree construction heuristic for solving the MINLPs arising in symbolic regression. We demonstrate the effectiveness of our ideas in a detailed numerical comparison in section 4, before concluding with some final remarks in section 5.

1.1. Review of Expression Trees. A mathematical expression can be represented by an expression tree. Figure 1 shows an expression tree of the pendulum formula (in general, an expression tree is not unique). An expression tree can be constructed by assigning an operand (an independent variable (x_j) or constant (cst)) or an operator ($+$, $-$, $*$, $/$, exp, log, $(\cdot)^2$, $(\cdot)^3$, $\sqrt{}$, etc.) to the nodes on a tree.

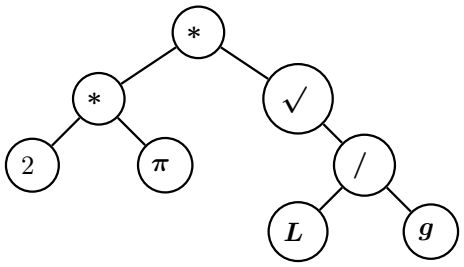


Fig. 1: An expression tree of the pendulum formula, $T = 2\pi\sqrt{\frac{L}{g}}$.

The objective in (1.1) is to minimize the empirical loss, in other words, to maximize the accuracy of the expression. Additionally, the objective function may include a regularization term modeling the complexity of the expression to obtain a simple expression. A common way to measure the complexity of symbolic regression is to calculate the number of nodes in an expression tree. We review in section 2 how we can formulate (1.1) by modeling expression trees using binary variables.

1.2. Methods, Test Problems, and Challenges for Symbolic Regression. Over the past few years, researchers have expressed renewed interest in learning symbolic expressions. Both exact formulations and solution techniques based on mixed-integer nonlinear programming and heuristic search techniques have been proposed. Recently, machine learning methodologies and hybrid techniques have also been proposed. Below, we briefly review each class of methods as well as test problem collections.

Heuristic Techniques for Symbolic Regression. Genetic programming is the most common approach for solving symbolic regression. It begins with an initial population of individuals or randomly selected expression trees. These trees are compared by a fitness measure and error metrics. Individuals with high scores have a higher probability of being selected for the next iteration of crossover, mutation, and reproduction. Ideas to enhance genetic algorithms have been proposed in [11, 12] to reduce the search space. Nicolau and McDermott [14] use prior information of the values of the dependent variable. The quality of solutions is not stable, however, because genetic algorithms are a stochastic process, which means that it can generate different

solutions for the same input data and the same settings. Kammerer et al. [11] remark that "it might produce highly dissimilar solutions even for the same input data."

Exact Mixed-Integer Approaches to Symbolic Regression. Exact approaches based on the MINLP formulation are deterministic in the sense that they return the same solution if the input data and parameter settings are the same. In principle MINLP approaches are exact in the sense that they will recover the global solution of (1.1), although their runtime may be prohibitively long in practice. MINLP formulations were first proposed in [3], extended in [4, 13], and independently studied in [2, 10]. MINLP formulations use binary variables to define the expression tree and continuous variables to represent intermediate values for each node and each data point; see section 2. The resulting optimization problem is a nonlinear, nonconvex MINLP, because it involves nonlinear operators such as $*, /, \exp$, and $\log$. The solution time typically increases exponentially in the maximum depth of the tree. To limit the runtime of the MINLP solvers, several researchers [2, 3, 4, 13] limit the structure of the expression tree (usually by limiting its depth) and solve a smaller MINLP. The approaches in [2, 3, 4] are tested only on noiseless data. Neumann et al. [13] propose an interesting methodology to avoid overfitting: (1) they add a constraint to limit the complexity of the expression tree (number of nodes); (2) they then generate a portfolio of solutions by varying the complexity limitation on the training set; and (3) they choose the solution based on the validation error.

Approaches Based on Machine Learning Methodologies. An approach based on training a neural network has been proposed in [19, 20] and is referred to as AI Feynman. A key of AI Feynman is to discover functional properties of the overall function using a neural network in order to reduce the search space and decompose the overall symbolic regression problem into smaller subproblems. The method recursively applies dimension reduction techniques (dimensional analysis, symmetry, and separability detection) until the remaining components are simple enough to be detected by polynomial fits or complete enumeration. These techniques require knowledge of the units of the independent and the dependent variables in order to perform the dimensional analysis, as well as smoothness of the underlying expression, because the method trains a neural network to evaluate values on missing points. A related method is considered in [5], which involves using a graph neural network.

Hybrid Techniques for Symbolic Regression. Austel et al. [1] propose an interesting idea that considers a generalized expression tree instead of an expression tree. A generalized tree assigns a monomial $(hx_1^{a_1}x_2^{a_2}\cdots x_d^{a_d})$ to each leaf node, instead of a single variable or constant. Consequently, generalized expression trees can represent a larger class of functions for the same depth.

The algorithm has two steps. First, it lists all generalized trees up to depth D. The number of generalized trees is reduced by removing redundant expressions. For example, both the generalized tree corresponding to the summation of two monomials $(hx_1^{a_1}x_2^{a_2}\cdots x_d^{a_d} + gx_1^{b_1}x_2^{b_2}\cdots x_d^{b_d})$ and the generalized tree corresponding to the subtraction of two monomials $(h'x_1^{a_1'}x_2^{a_2'}\cdots x_d^{a_d'} - g'x_1^{b_1'}x_2^{b_2'}\cdots x_d^{b_d'})$ are equivalent because both can represent the same set of functions; therefore, one of the expressions can be removed from the list. The list of all generalized trees up to depth one with operators $\{+, -, *, /, \sqrt{}\}$ is

$$L_1,\ \sqrt{L_1},\ (L_1 + L_2 \equiv L_1 - L_2),\ (L_1 * L_2 \equiv L_1/L_2),$$

where L_1 and L_2 are monomials and $\equiv$ represents that two expressions are equivalent. Second, the algorithm solves optimization problems to find a global solutions for each

generalized tree. For example, the optimization problem of $L_1 + L_2$ is

$$\min_{h,a_1,\dots,a_d,g,b_1,\dots,b_d} \sum_{i=1}^{n_{\text{data}}} \left(z_i - \left(h x_{i,1}^{a_1} x_{i,2}^{a_2} \cdots x_{i,d}^{a_d} + g x_{i,1}^{b_1} x_{i,2}^{b_2} \cdots x_{i,d}^{b_d} \right) \right)^2, \tag{1.2}$$

where h, g are bounded continuous variables and a, b are bounded integer variables. Even though the problem assumes that the tree structure of the generalized expression tree is given, it is a mixed-integer nonlinear (nonconvex) programming problem that is in NP-hard. Constraints are added based on the knowledge of the units of the independent and the dependent variables to reduce the search space.

Benchmark Problems for Symbolic Regression. Several test problem collections have been produced to test symbolic regression ideas. For example, ALAMO, the Automatic Learning of Algebraic Models, is a software package for symbolic regression; see http://minlp.com/alamo. The test set of [4] (see http://minlp.com/nlp-and-minlp-test-problems) has 24 instances. In [2], the authors consider Kepler's law $d = \sqrt[3]{c\tau^2(M+m)}$ and the period of the pendulum $\tau = 2\pi\sqrt{\ell/g}$. A set of examples from [22] is available at http://gpbenchmarks.org. A number of test problems are also in [19, 20] and are available at https://space.mit.edu/home/tegmark/aifeynman.html.

The Challenges of Symbolic Regression. Solving symbolic regression problems such as (1.1) has been shown to be challenging; see, for example,[1, 2, 4, 13]. In particular, the problem complexity increases with the number of operators and the number of operator types. For example, the number of expression trees represented by d independent variables and B binary operators with a maximum depth D is more than $(B \cdot d)^{2^D}/B$.[1] Another challenge is the non-convexity of the problem, which remains even if the expression tree is fixed because the problem with a fixed expression tree is equivalent to optimize parameters of an arbitrary functional form.

2. An Improved MINLP Formulation of Symbolic Regression.

We review and improve a MINLP formulation that searches an expression tree with the minimum training error given data points $(x_{i,1}, \dots, x_{i,d}, z_i) \in \mathbb{R}^{d+1}$ for $i = 1, \dots, n_{\text{data}}$. Our formulation improves the one proposed by [4]. The new constraints remove equivalent expression trees and tighten the feasible set of the relaxation.

2.1. Notation.

A *relaxation* refers to the relaxation obtained by relaxing binary variables. We let $[a] := \{1, 2, \dots, a\}$ for $a \in \mathbb{Z}_{++}$, the set of positive integers.

2.2. Inputs.

We are given a set of operators and a set of nodes that define the superset of all feasible expression trees. By limiting the number of nodes and operands used to construct the expression tree, we transform the infinite-dimensional problem (1.1) into a finite-dimensional problem. We denote by $\mathcal{P} \subseteq \{+, -, *, /, \sqrt{}, \exp, \log, \dots\}$ a set of operators. To streamline our presentation, we define the set of binary operators $\mathcal{B} := \mathcal{P} \cap \{+, -, *, /\}$, the set of unary operators $\mathcal{U} := \mathcal{P} \cap \{\sqrt{}, \exp, \log\}$, and the set of operands $\mathcal{L} = \{x_1, \dots, x_d, \text{cst}\}$, where cst is a constant. We denote the set of all operators and operands by $\mathcal{O} := \mathcal{B} \cup \mathcal{U} \cup \mathcal{L}$.

We identify each node of the tree by an integer, and we let $\mathcal{N}$ denote the set of all nodes in the tree. We denote the children of node $n \in \mathcal{N}$ by $2n$ and $2n+1$, respectively. We assume $1 \in \mathcal{N}$, which is the root of the tree. We denote by $\mathcal{T}$ the set

[1] Let T_δ denote the number of expression trees up to depth δ. $T_D \geq (B \cdot d)^{2^D}/B$ is derived from the system of $T_0 \geq n$ and $T_\delta \geq B \cdot T_{\delta-1}^2$. The equality holds if we allow an expression tree to use only binary operators and the independent variables.

of terminal nodes (that have no child). We assume that $\mathcal{N}$ corresponds to a full binary tree.[2] For example, $\mathcal{N} = \{1, 2, 3, 6, 7\}$ is a full binary tree, while $\mathcal{N} = \{1, 2, 3, 4\}$ is not because node 2 has only one child, namely, node 4. Provided that the set of operators $\mathcal{O}$ is finite and we limit the number of nodes, this MINLP formulation transforms the infinite-dimensional functional approximation (1.1) into a finite-dimensional problem.

2.3. Decision Variables and Individual Variable Restrictions. There are three types of decision variables. The binary variable y_n^o is one if operator o is assigned to node n, and zero otherwise. Variable c_n is the constant value at node n if node n exists, and zero otherwise. Therefore, y_n^o and c_n determine the expression tree. Variable $v_{i,n}$ represents the intermediate computation value at node n for data point i. In other words, $v_{i,n}$ is the value of the symbolic expression represented by the subtree rooted by node n at data point i. Therefore, $v_{i,1}$ is the value predicted by the expression tree of data point i. All continuous variables are bounded. To streamline our presentation, we use $n \notin \mathcal{T}$ instead of $n \in \mathcal{N} \setminus \mathcal{T}$ in the following discussions. We define $\mathcal{Y} := \{(n, o),\ \forall o \in \mathcal{O},\ \forall n \notin \mathcal{T}\} \cup \{(n, o),\ \forall o \in \mathcal{L},\ \forall n \in \mathcal{T}\}$, the set of all pairs of node n and operator o such that o can be assigned to n. To summarize, our model has the following set of variables and ranges:

$$y_n^o \in \{0, 1\}, \qquad \forall (n, o) \in \mathcal{Y}, \tag{2.1a}$$

$$c_{\text{lo}} \le c_n \le c_{\text{up}}, \qquad \forall n \in \mathcal{N}, \tag{2.1b}$$

$$v_{\text{lo}} \le v_{i,n} \le v_{\text{up}}, \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}. \tag{2.1c}$$

We assume without loss of generality that $v_{\text{lo}} \le c_{\text{lo}} \le 0 \le c_{\text{up}} \le v_{\text{up}}$.

2.4. Objective Function. We minimize the mean of the squared errors

$$\min \quad \frac{1}{n_{\text{data}}} \sum_{i=1}^{n_{\text{data}}} (z_i - v_{i,1})^2. \tag{2.2}$$

Additionally, we might add a regularization term such as $\lambda \sum_{(n,o)\in\mathcal{Y}} y_n^o$, where $\lambda \in \mathbb{R}_+$ is a regularization parameter to promote a sparser expression tree.

2.5. Constraints. In subsections 2.5.1 and 2.5.2, we introduce constraints that are necessary to solve this problem. In subsections 2.5.3 to 2.5.5, we introduce constraints that remove equivalent expression trees and/or reduce the space of the relaxation, an acton that potentially leads to an improvement in computation. We compare the constraints with the similar constraints in [4] in each section. For the sake of completeness, the formulation proposed in [4] is summarized in Appendix A in terms of our notations.

2.5.1. Tree-Defining Constraints. Tree-defining constraints enforce that the assignment of operators and operands results in a valid expression tree. The constraints consist of (2.3a), (2.3b), (A.1a), and (A.1b). In addition to (A.1a) and (A.1b) from [4], we use the following tree-defining constraints:

$$\sum_{o\in\mathcal{B}\cup\mathcal{U}} y_n^o = \sum_{o\in\mathcal{O}} y_{2n+1}^o, \qquad n \notin \mathcal{T}, \tag{2.3a}$$

$$\sum_{o\in\mathcal{B}} y_n^o = \sum_{o\in\mathcal{O}} y_{2n}^o, \qquad n \notin \mathcal{T}. \tag{2.3b}$$

[2] A full (proper) binary tree is a tree in which every node has zero or two children.

Constraint (2.3a) enforces that a binary/unary operator is assigned to node n if and only if its right child (node $2n+1$) exists. Constraint (2.3b) enforces that a binary operator is assigned to node n if and only if its left child (node $2n$) exists. Constraint (A.1a) forces the assignment of at most one operator to a node. Constraint (A.1b) forces the expression tree to include at least one independent variable. All four constraints are necessary to obtain an expression tree with valid operator/operand assignments.

In contrast, the tree-defining constraints from [4] are (A.1a)-(A.1f). Figure 2 shows two assignments of operators and operands to an expression tree. Both represent the symbolic expression x_1, and both are feasible in (A.1c)-(A.1f), but only Figure 2a is feasible in (2.3a)-(2.3b). We formalize this observation by showing in

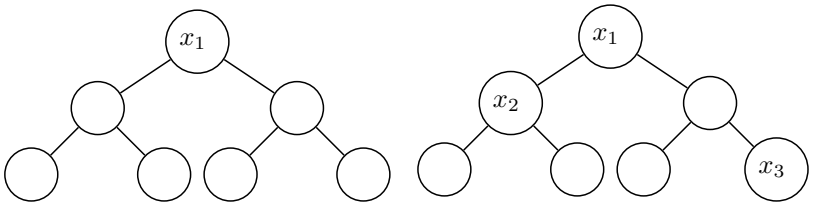


Fig. 2: Two equivalent expression trees corresponding to x_1. Both are feasible in the [4]'s formulation, while only (a) is feasible in our improved formulation.

Lemma 2.1 that our tree-defining constraints system is tighter.

LEMMA 2.1. *It holds that* $\{y \in \{0,1\}^{|\mathcal{Y}|} \mid (2.3a), (2.3b), (A.1a), (A.1b)\} \subsetneq \{y \in \{0,1\}^{|\mathcal{Y}|} \mid (A.1a)\text{-}(A.1f)\}$.

Proof. Let $S := \{y \mid (2.1a), (2.3a), (2.3b), (A.1a), (A.1b)\}$ and $T := \{y \mid (2.1a), (A.1a)\text{-}(A.1f)\}$. We first show that $S \subseteq T$. Pick any point $y \in S$. We need to show that y satisfies (A.1c)-(A.1f). Point y satisfies (A.1c) and (A.1d) because those are relaxations of (2.3a) and (2.3b), respectively, by the fact that $\mathcal{B}$ (set of binary operators), $\mathcal{U}$ (set of unary operators), and $\mathcal{L}$ (set of operands) are a partition of $\mathcal{O}$ (set of operators and operands). We can also show that y satisfies (A.1e) by

$$\sum_{o \in \mathcal{U} \cup \mathcal{L}} y_n^o \leq 1 - \sum_{o \in \mathcal{B}} y_n^o = 1 - \sum_{o \in \mathcal{O}} y_{2n}^o,$$

where the inequality and the equality hold by (A.1a) and (2.3b), respectively. Similarly, we can show that y satisfies (A.1f) by

$$\sum_{o \in \mathcal{L}} y_n^o \leq 1 - \sum_{o \in \mathcal{B} \cup \mathcal{U}} y_n^o = 1 - \sum_{o \in \mathcal{O}} y_{2n+1}^o,$$

where the inequality and the equality hold by (A.1a) and (2.3a), respectively. Therefore, y satisfies all the constraints in T, and consequently $S \subseteq T$ holds.

We next show that there exists $y \in T \setminus S$. Let $\mathcal{N} = [7]$, $y_1^{x_1} = y_2^{x_2} = y_7^{x_3} = 1$, and otherwise $y_n^o = 0$. Figure 2b shows the expression tree corresponding to y. The y satisfies (A.1a) and (A.1b). Also, y satisfies (A.1a)–(A.1b) because every left-hand-side value is zero. However, y does not satisfy (2.3a) and (2.3b) because x_2 and x_3 cannot be assigned unless an operator is assigned to their parents. Therefore, the proof is complete. ∎

2.5.2. Value-Defining Constraints. Value-defining constraints enforce that the value of $v_{i,n}$ is computed based on the solution expression tree and data points. Specifically, if an operand is assigned to node n, then $v_{i,n}$ is equal to the value of the operand. If a unary operator $\otimes(x)$ is assigned to node n, then $v_{i,n}$ is equal to $\otimes(v_{i,2n+1})$. If a binary operator $\otimes$ is assigned to node n, then $v_{i,n}$ is equal to $v_{i,2n} \otimes v_{i,2n+1}$. Cozaad and Sahinidis [4] introduce a value-defining constraint for each data point, each node, and each operator or operand, given in (A.2b)–(A.11c) in Appendix A.2. All these constraints are necessary to ensure the correct prediction values for each data point given an expression tree.

We propose a set of improved value-defining constraints, (2.4a)–(2.4b) together with (A.4a)–(A.11c) from [4]:

$$v_{i,n} \leq \sum_{j=1}^{d} x_{i,j} y_n^{x_j} + v_{\text{up}} \sum_{o \in \mathcal{B} \cup \mathcal{U} \cup \{\text{cst}\}} y_n^o, \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}, \tag{2.4a}$$

$$v_{i,n} \geq \sum_{j=1}^{d} x_{i,j} y_n^{x_j} + v_{\text{lo}} \sum_{o \in \mathcal{B} \cup \mathcal{U} \cup \{\text{cst}\}} y_n^o, \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}. \tag{2.4b}$$

Our value-defining constraints replace (A.2a)-(A.3b) with (2.4a)-(2.4b). Both (2.4a)-(2.4b) and (A.2a)-(A.3b) represent the following disjunction:

$$\bigvee_{o \in \{x_1, \ldots, x_d\}} \begin{bmatrix} y_n^o = 1 \\ v_{i,n} = x_{i,j}, \\ \forall i \in [n_{\text{data}}] \end{bmatrix} \bigvee \begin{bmatrix} \sum_{o \in \mathcal{O}} y_n^o = 0 \\ v_{i,n} = 0, \\ \forall i \in [n_{\text{data}}] \end{bmatrix} \bigvee \begin{bmatrix} \sum_{o \in \mathcal{B} \cup \mathcal{U} \cup \{\text{cst}\}} y_n^o = 1 \\ v_{\text{lo}} \leq v_{i,n} \leq v_{\text{up}}, \\ \forall i \in [n_{\text{data}}] \end{bmatrix}, \quad \forall n \in \mathcal{N}. \tag{2.5}$$

Note that if $\sum_{o \in \mathcal{B} \cup \mathcal{U} \cup \{\text{cst}\}} y_n^o = 1$ (i.e., node n exists), then the value of $v_{i,n}$ is determined by (A.4a)–(A.11c). Our formulation reduces the number of constraints. The number of constraints (2.4a)–(2.4b) is $n_{\text{data}}|\mathcal{N}|$, while the number of constraints (A.2a)–(A.3b) is $n_{\text{data}}|\mathcal{N}|(d+1)$. We show in Lemma 2.3 that our formulation does not change the feasible set and, in fact, reduces the space of the relaxation.

We first show in Lemma 2.2 that we can merge k big-M constraints associated with different constant bounds on an identical function. This merge reduces the space of the relaxation while it does not change the feasible space.

LEMMA 2.2. *Let m be a positive integer, $k \in [m]$, $w \in \mathbb{R}^k$, $M > \max_{i \in [k]} w_i$. Let $\mathcal{F}_B = \{y \in \{0,1\}^m \mid \sum_{i=1}^m y_i = 1\}$ and $\mathcal{F}_C = \{y \in \mathbb{R}_+^m \mid \sum_{i=1}^m y_i = 1\}$. Consider sets S and T, where*

$$S := \{(x,y) \in \mathbb{R} \times \mathbb{R}^n \mid f(x) \leq \sum_{i \in [k]} w_i y_i + M(1 - \sum_{i \in [k]} y_i)\},$$

$$T := \{(x,y) \in \mathbb{R} \times \mathbb{R}^n \mid f(x) \leq w_i y_i + M(1 - y_i),\ \forall i \in [k]\}.$$

Then, the following relations hold:

$$\{(x,y) \in S \mid y \in \mathcal{F}_B\} = \{(x,y) \in T \mid y \in \mathcal{F}_B\}, \tag{2.6a}$$

$$\{(x,y) \in S \mid y \in \mathcal{F}_C\} \subsetneq \{(x,y) \in T \mid y \in \mathcal{F}_C\}. \tag{2.6b}$$

Proof. The proof of this result is given in Appendix B. ∎

Next, we show that the new constraints do not change the feasible set of the MINLP but improve its continuous relaxation.

LEMMA 2.3. *Let* $\mathcal{F}_B = \{(y,v) \in \{0,1\}^{|\mathcal{Y}|} \times [v_{lo}, v_{up}]^{n_{data}|\mathcal{N}|} \mid \text{(A.1a)}\}$ *and* $\mathcal{F}_C = \{(y,v) \in [0,1]^{|\mathcal{Y}|} \times [v_{lo}, v_{up}]^{n_{data}|\mathcal{N}|} \mid \text{(A.1a)}\}$. *It holds that*

$$\{(y,v) \in \mathcal{F}_B \mid \text{(2.4a)-(2.4b)}\} = \{(y,v) \in \mathcal{F}_B \mid \text{(A.2a)-(A.3b)}\}, \tag{2.7a}$$

$$\{(y,v) \in \mathcal{F}_C \mid \text{(2.4a)-(2.4b)}\} \subsetneq \{(y,v) \in \mathcal{F}_C \mid \text{(A.2a)-(A.3b)}\}. \tag{2.7b}$$

Proof. Constraints (A.1a), (2.4a)–(2.4b), and (A.2a)–(A.3b) are all separable in $n \in \mathcal{N}$. Thus, it is sufficient to show that (2.7a) and (2.7b) hold for a specific n. We introduce $y_n^{\text{none}} := 1 - \sum_{(n',o)\in\mathcal{Y}:n'=n} y_n^o$. By definition, $y_n^{\text{none}} + \sum_{(n',o)\in\mathcal{Y}:n'=n} y_n^o = 1$. Then, we can merge all "$v_{i,n} \le \cdots$" constraints in (A.2a)–(A.3b) into (2.4a) for all $n \in \mathcal{N}$, an action that corresponds to merging constraints in T to the constraint in S in Lemma 2.2. Similarly, we merge all "$v_{i,n} \ge \cdots$" constraints in (A.2a)–(A.3b) into (2.4b) for all $n \in \mathcal{N}$. By Lemma 2.2, this merge strictly reduces the space of the relaxation while it does not change the feasible set. Therefore, the proof is complete. ∎

2.5.3. Redundancy-Eliminating Constraints. Redundancy-eliminating constraints exclude redundant operations and remove three kinds of redundancy described in Table 1.

Redundancy type	Example	Constraints
Association property	$x_1 - (x_2 - 3) = x_1 + (3 - x_2)$	(2.8a), (2.8b)
Operations on constants	$2 = \sqrt{4} = 1.5 + 0.5$	(2.8c), (A.12d)
Nested operations	$x = e^{\log(x)} = \log(e^x)$	(A.12e), (A.12f)

Table 1: Redundancy removing constraints.

In addition to the redundancy-eliminating constraints (A.12d)-(A.12f) from [4], we introduce the following constraints:

$$y_n^+ + y_{2n+1}^- \le 1, \qquad n \notin \mathcal{N}_{\text{perfect}}, \tag{2.8a}$$

$$y_n^* + y_{2n+1}^/ \le 1, \qquad n \notin \mathcal{N}_{\text{perfect}}, \tag{2.8b}$$

$$y_{2n+1}^{\text{cst}} \le y_n^+ + y_n^*, \qquad n \notin \mathcal{T}, \tag{2.8c}$$

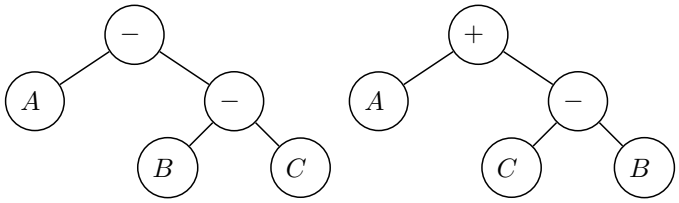
where $\mathcal{N}_{\text{perfect}}$ is a set of nodes whose rooted subtree of $\mathcal{N}$ is a perfect binary tree.[3] Specifically, the constraints exclude all equivalent expressions except the first expression of the examples in Table 1.

Redundancy induced by the association property is not considered in [4], while our redundancy-removing constraints (2.8a) and (2.8b) exclude such cases. For example, all the expression trees in Figures 3 and 4 are feasible in the formulation in [4]; however, only (a) is feasible in our formulation.

Constraint (2.8c) allows the right child to be a constant only if $+$ or $*$ is assigned to node n. The constraints exclude $-C$ and $/C$ because equivalent expressions $+(-C)$ and $*(1/C)$ are feasible. In addition, they exclude $\sqrt{C}$, $\exp C$, and $\log C$ because we

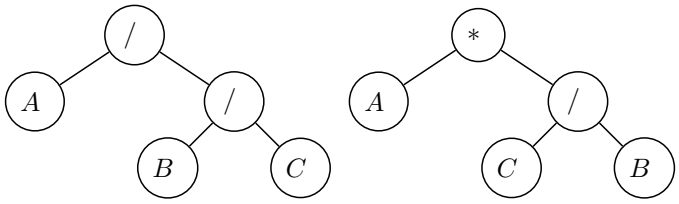
[3] A perfect binary tree is a binary tree in which all nonterminal nodes have two children and all terminal nodes have the same depth.

can represent them as a single constant node. This type of redundancy is considered in [4], and the corresponding constraints are (A.12a)–(A.12c). We show in Lemma 2.4 that the substitution (2.8c) for (A.12a)–(A.12c) results in a tighter relaxation.



(a) $A-(B-C)$ (b) $A+(C-B)$

Fig. 3: Two equivalent expression trees with addition and subtraction. Both are feasible in [4], but only (a) is feasible in (2.8a).



(a) $A/(B/C)$ (b) $A*(C/B)$

Fig. 4: Two equivalent expression trees with multiplication and division. Both are feasible in [4], but only (a) is feasible in (2.8b).

LEMMA 2.4. *Let $\mathcal{F}_B$ and $\mathcal{F}_C$ denote the set of binary and continuous y, respectively, that satisfy the tree-defining constraints, that is,*

$$\mathcal{F}_B = \{y \in \{0,1\}^{|\mathcal{Y}|} \mid (2.3a), (2.3b), (A.1a), (A.1b)\},$$
$$\mathcal{F}_C = \{y \in [0,1]^{|\mathcal{Y}|} \mid (2.3a), (2.3b), (A.1a), (A.1b)\}.$$

It follows that

$$\{y \in \mathcal{F}_B \mid (2.8c)\} = \{y \in \mathcal{F}_B \mid (A.12b)\text{-}(A.12d)\}, \tag{2.10a}$$
$$\{y \in \mathcal{F}_C \mid (2.8c)\} \subsetneq \{y \in \mathcal{F}_C \mid (A.12b)\text{-}(A.12d)\}. \tag{2.10b}$$

Proof. Recall (A.12b)-(A.12d):

$$y^{\text{cst}}_{2n+1} \le 1 - \sum_{o \in \mathcal{U}} y^o_n, \qquad n \notin \mathcal{T},$$
$$y^{\text{cst}}_{2n+1} \le 1 - y^-_n, \qquad n \notin \mathcal{T},$$
$$y^{\text{cst}}_{2n+1} \le 1 - y^/_n, \qquad n \notin \mathcal{T}.$$

By relaxing (2.3a), we get the following inequality:

$$y^{\text{cst}}_{2n+1} \le 1 - y^{\text{none}}_n, \qquad n \notin \mathcal{T},$$

where $y_n^{\text{none}} := 1 - \sum_{o \in \mathcal{B} \cup \mathcal{U}} y_n^o$. Let us consider that those constraints are defining the upper bound of y_{2n+1}^{cst} depending on the choice of y_n. For example, $y_{2n+1}^{\text{cst}} \leq 1 - y_n^-$ enforces the upper bound by 0 if $y_n^- = 1$; otherwise it relaxes this constraint. By Lemma 2.2, we can merge the constraints for each n. The merged constraints are

$$y_{2n+1}^{\text{cst}} \leq 1 - \sum_{o \in \mathcal{U}} y_n^o - y_n^- - y_n^{/} - y_n^{\text{none}} = y_n^+ + y_n^*, \qquad n \notin \mathcal{T}.$$

By Lemma 2.2, this merge does not change the feasible set; it reduces the feasible set of the relaxation. ∎

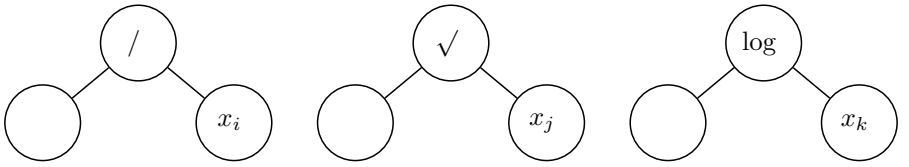
2.5.4. Implication Cuts. Implication cuts are motivated by the fact that some operators are domain-restricted, for example, $/$, $\sqrt{}$, and log. From a set of given data points, we can identify the characteristics of the independent variables:

$$\mathcal{X}_{\text{posi}} = \{i \in [d] \mid \exists j \in [n_{\text{data}}] : x_{i,j} > 0\}, \tag{2.11a}$$

$$\mathcal{X}_{\text{nega}} = \{i \in [d] \mid \exists j \in [n_{\text{data}}] : x_{i,j} < 0\}, \tag{2.11b}$$

$$\mathcal{X}_{\text{zero}} = \{i \in [d] \mid \exists j \in [n_{\text{data}}] : x_{i,j} = 0\}. \tag{2.11c}$$

All invalid expression trees with up to depth one are described in Figure 5. We can



(a) $(\cdot)/x_i$, $i \in \mathcal{X}_{\text{zero}}$ (b) $\sqrt{x_j}$, $j \in \mathcal{X}_{\text{nega}}$ (c) $\log(x_k)$, $k \in \mathcal{X}_{\text{nega}} \cup \mathcal{X}_{\text{zero}}$

Fig. 5: Invalid expression trees because of domain restriction.

write the constraints that restrict the expressions in Figure 5 as follows:

$$y_n^{/} + y_{2n+1}^{x_j} \leq 1, \qquad \forall j \in \mathcal{X}_{\text{zero}},\ \forall n \notin \mathcal{T}, \tag{2.12a}$$

$$y_n^{\surd} + y_{2n+1}^{x_j} \leq 1, \qquad \forall j \in \mathcal{X}_{\text{nega}},\ \forall n \notin \mathcal{T}, \tag{2.12b}$$

$$y_n^{\log} + y_{2n+1}^{x_j} \leq 1, \qquad \forall j \in \mathcal{X}_{\text{nega}} \cup \mathcal{X}_{\text{zero}},\ \forall n \notin \mathcal{T}. \tag{2.12c}$$

Constraints (2.12a)–(2.12c) exclude an expression tree that includes the expressions in Figure 5 as a subtree. We can generate more invalid trees from depth-two or higher-depth trees. One example is $\sqrt{x_j x_k}$ for $j \in \mathcal{X}_{\text{posi}}$ and for $k \in \mathcal{X}_{\text{nega}}$. The corresponding constraint is

$$y_n^{\surd} + y_{2n+1}^* + y_{4n+2}^{x_j} + y_{4n+3}^{x_k} \leq 3, \tag{2.13}$$
$$\forall j \in \mathcal{X}_{\text{posi}},\ \forall k \in \mathcal{X}_{\text{nega}},\ \forall n \in \mathcal{N} : \{4n+2, 4n+3\} \subseteq \mathcal{N}.$$

Note that the expressions in Figure 5 are already infeasible in both our formulation and the formulation in [4] because of (A.8d), (A.9c), and (A.11c). However, adding implication cuts reduces the feasible space of the relaxation. Example 2.5 shows that the solution $y_n^{\surd} = y_{2n+1}^{x_j} = 0.9$ for $j \in \mathcal{X}_{\text{nega}}$ is feasible in the space of the relaxation of the formulation without implication cuts, whereas it violates (2.12b).

Example 2.5. We consider a symbolic regression problem with $\mathcal{N} = \{1, 3\}$, $\mathcal{P} = \{\sqrt{}\}$ and a single data point $(x_{1,1}, z_1) = (-1, 5)$. Suppose that $v_{\text{lo}} = -10$, $v_{\text{up}} = 10$, and $\epsilon = 0.01$. To streamline the presentation, we assume that $y_1^{\text{cst}} = y_3^{\text{cst}} = 0$. The formulation without implication cuts is follows:

$$
\begin{aligned}
\min \quad & (5 - v_{1,1})^2, \\
\text{s.t.} \quad & \text{Tree-Defining Constraints:} \\
& y_1^{\surd} + y_1^{x_1} \le 1, \quad y_3^{x_1} \le 1, \quad y_1^{x_1} + y_3^{x_1} \ge 1, \quad y_1^{\surd} = y_3^{x_1}, \\
& \text{Value-Defining Constraints:} \\
& v_{1,1} \le (-1) y_1^{x_1} + 10 y_1^{\surd}, \quad v_{1,1} \ge (-1) y_1^{x_1} - 10 y_1^{\surd}, \\
& v_{1,3} \le (-1) y_3^{x_1}, \quad v_{1,3} \ge (-1) y_3^{x_1}, \\
& v_{1,1}^2 - v_{1,3} \le 90(1 - y_1^{\surd}), \quad v_{1,1}^2 - v_{1,3} \ge -10(1 - y_1^{\surd}), \\
& 0.01 - v_{1,3} \le 10.01(1 - y_1^{\surd}), \\
& \text{Variable Restrictions:} \\
& -10 \le v_{1,1}, v_{1,3} \le 10, \quad y_1^{\surd}, y_1^{x_1}, y_3^{x_1} \in \{0, 1\}.
\end{aligned}
$$

Note that there is no redundancy-removing constraint because we consider only a limited set of operators and a limited set of nodes. We consider the relaxation that replaces $y_1^{\surd}, y_1^{x_1}, y_3^{x_1} \in \{0, 1\}$ with $y_1^{\surd}, y_1^{x_1}, y_3^{x_1} \in [0, 1]$. Consider the solution $(y_1^{\surd}, y_1^{x_1}, y_3^{x_1}, v_{1,1}, v_{1,3}) = (0.9, 0.1, 0.9, 0, -0.9)$. The solution is feasible in the relaxation, whereas it violates $y_1^{\surd} + y_3^{x_1} \le 1$, which is (2.12b).

2.5.5. Symmetry-Breaking Constraints. Symmetry-breaking constraints remove equivalent expression trees because of a symmetric operator including addition $(+)$ and multiplication $(*)$, for example, $x_1 + x_2 = x_2 + x_1$. We retain only those expression trees in which the left argument value is greater than or equal to the right argument value for the first data point:

$$
v_{1,2n} - v_{1,2n+1} \ge (v_{\text{lo}} - v_{\text{up}})(1 - y_n^+ - y_n^*), \qquad n \in \mathcal{N}_{\text{perfect}}. \tag{2.14}
$$

Symmetry-breaking constraints are discussed in [4, (5)]. We rewrite the constraints in terms of our notations because we assume that $\mathcal{N}$ corresponds to a full binary tree whereas [4] assumes that $\mathcal{N}$ corresponds to a perfect binary tree.

2.6. Summary. We summarize the formulation and our contributions in Table 2.

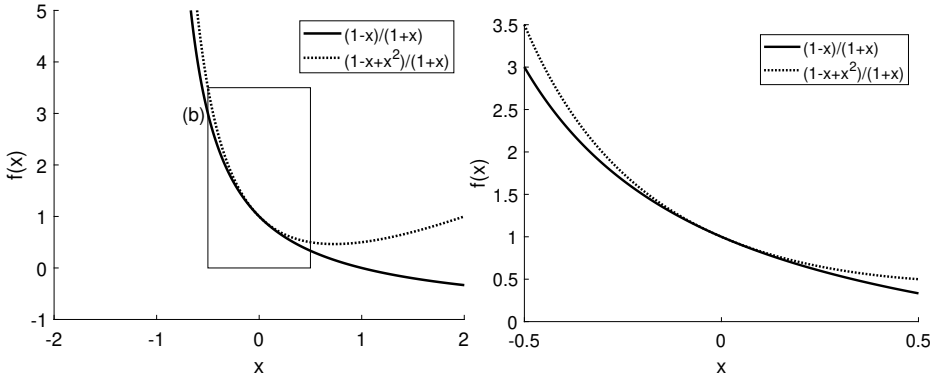
Table 2: Summary of the formulation and contributions compared with the benchmark formulation [4].

Categories	Variables	Convexity	Remove additional equiv. expressions	Reduce relaxation space
Objective	v	convex		
Tree-defining constraints	y, c	linear	✓	✓
Value-defining constraints	y, c, v	nonconvex		✓
Redundancy-eliminating constraints	y	linear	✓	✓
Implication cuts	y	linear		✓
Symmetry-breaking constraints	y, v	linear		

The formulation we propose for symbolic regression is

$$
\begin{aligned}
\min_{y,c,v} \quad & \frac{1}{n_{\text{data}}} \sum_{i=1}^{n_{\text{data}}} (z_i - v_{i,1})^2 \\
\text{s.t.} \quad & \text{(2.3a)-(2.3b)}, \text{(A.1a)-(A.1b)}, && \text{(Tree-defining constraints)} \\
& \text{(2.4a)-(2.4b)}, \text{(A.4a)-(A.11c)}, && \text{(Value-defining constraints)} \\
& \text{(2.8a)-(2.8c)}, \text{(A.12d)-(A.12f)}, && \text{(Redundancy-eliminating constraints)} \\
& \text{(2.12a)-(2.12c)}, && \text{(Implication cuts)} \\
& \text{(2.14)}, && \text{(Symmetry-breaking constraints)} \\
& y_n^o \in \{0,1\}, && \forall (n,o) \in \mathcal{Y}, \\
& c_{\text{lo}} \le c_n \le c_{\text{up}}, && \forall n \in \mathcal{N}, \\
& v_{\text{lo}} \le v_{i,n} \le v_{\text{up}}, && \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}.
\end{aligned}
$$

3. Sequential Tree Construction Heuristic. In this section, we propose a heuristic that searches a solution by building up an expression tree starting from a simple approximation to a comprehensive formula. It repeatedly modifies a part of the current expression tree in order to lower the training error. It is motivated by the fact that a comprehensive formula can be approximated by a simple formula, and we observe that those two formulas have similar structures. For example, $\frac{1-x+x^2}{1+x}$ can be approximated by $\frac{1-x}{1+x}$ when $|x|$ is small, see Figure 6. We can achieve the comprehensive formula by adding x^2 to the simple formula. Another example can be found in Kepler's third law, where the comprehensive formula $d = \sqrt[3]{c\tau^2(M+m)}$ is approximated by a simple formula $d = \sqrt[3]{c\tau^2 M}$ for $M \gg m$.



(a) $x \in (-1, 2)$ (b) $x \in (-0.5, 0.5)$

Fig. 6: Illustration of $\frac{1-x+x^2}{1+x}$ and $\frac{1-x}{1+x}$.

Our approach is motivated by a heuristic framework for general MINLPs that searches an improved solution from the neighbors of the current solution, namely, local branching proposed by [8]. A local branching heuristic iteratively explores the neighbors by solving a restricted MINLP with a branch-and-bound solver. We develop a *sequential tree construction heuristic (STreCH)* based on the formulation in section 2. In subsection 3.1 we define the neighbors of an expression tree that is the core of our heuristic, and in subsection 3.2 we discuss how to speed up the heuristic.

3.1. Definition of Neighbors and a Basic Heuristic. We define the distance between two expression trees as the number of nodes assigned different operators/operands. There are three cases of a node with different assignment:

- a change of a node assignment from an operator/operand to another operator/operand,
- an addition of a new node, and
- a deletion of a node.

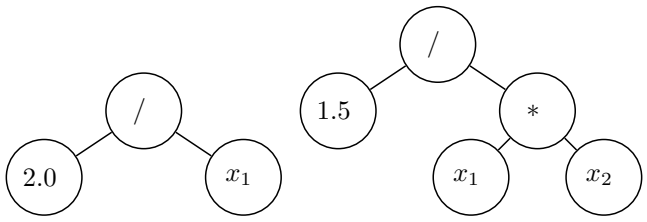
We consider constant as an operand. This distance does not count a change from a constant value to another constant value. For example, the distance between two trees in Figure 7 is three because of one change (from x_1 to $*$) and two additions of a node. The change in the constant value (from 2.0 to 1.5) is not counted.

We define a k-neighbor of a given solution $\bar{y}$ as an expression tree with distance at most k. Let $\mathcal{N}_{\text{active}}(\bar{y})$ be the set of nodes in which an operator/operand is assigned. Let $o(n, \bar{y})$ for $n \in \mathcal{N}_{\text{active}}(\bar{y})$ be the operator/operand that is assigned to node n. The set of k-neighbors of $\bar{y}$, $\mathcal{NB}_k(\bar{y})$, is represented as follows:

$$\delta(\bar{y}, y) = \sum_{n \in \mathcal{N}_{\text{active}}(\bar{y})} (1 - y_n^{o(n,\bar{y})}) + \sum_{n \notin \mathcal{N}_{\text{active}}(\bar{y})} \sum_{o \in \mathcal{O}} y_n^o, \tag{3.1}$$

$$\mathcal{NB}_k(\bar{y}) = \{y \mid \delta(\bar{y}, y) \leq k\}. \tag{3.2}$$

The restricted MINLP that searches k-neighbors needs a single additional linear constraint described in (3.2).



(a) $2.0/x_1$ (b) $1.5/(x_1x_2)$

Fig. 7: Two expression trees with distance three.

Our basic heuristic works as follows. It first obtains an initial solution by solving a small-sized problem. At each iteration, it searches for an improvement of the current solution by exploring its neighbors by solving a restricted MINLP with (3.2). It terminates by optimality (the objective value is less than a given tolerance ϵ) or by time limit. In our implementation we encounter the following situations:

- When the tree size (the number of nodes) of the incumbent solution is large, it takes a long time to search its neighbors.
- There is no better solution in its neighbor at some iteration.

We next propose some ideas to mitigate these situations.

3.2. Heuristics to Speed Up Ideas and STreCH. In this section we propose STreCH, whose pseudocode is described in Algorithm 3.3. Algorithm 3.1 describes a restricted MINLP solve, and Algorithm 3.2 describes the solution improvement procedure. In addition, we propose a number of heuristics to speed up the solution process:

ALGORITHM 3.1 `ResMINLP` (approximately solving a restricted MINLP).

Data: (x, z), $\mathcal{P}$, $\mathcal{N}$; (optional) $\bar{y}$, k_1, k_2, β, γ
Result: An expression tree (y, c) found by MINLP and its training error ω
Formulate a MINLP with with data (x, z), operators $\mathcal{P}$, nodes $\mathcal{N}$;
if *$\bar{y}$, k_1, and k_2 are given* **then**
  Add constraint $k_1 \leq \delta(\bar{y}, y) \leq k_2$ to search within $\mathcal{NB}_{k_2}(\bar{y}) \setminus \mathcal{NB}_{k_1}(\bar{y})$;
if *$\bar{y}$ and β are given* **then**
  Fix some variables in y by given solution $\bar{y}$ and fix level β;
if *γ is given* **then**
  Set the node limit of a branch-and-bound solver to γ;
Solve the problem with a branch-and-bound solver;
return (y, c, ω);

Early Termination. A branch-and-bound solver returns an optimal solution with the proof of optimality. However, proving optimality for a restricted problem does not guarantee optimality of the whole problem. Therefore, we terminate the solver under one of the following conditions: (1) when it finds an improved solution, or (2) it reaches a time-limit to prove optimality. First, we stop an iteration when the solver finds a solution whose training error is smaller than $(100 * \delta)\%$ of the training error of the incumbent. Second, we stop an iteration when it reaches at a predetermined time limit or node limit.

Start Value. We provide the incumbent as a starting point. In general, it is not always efficient especially when we are looking for an optimal solution with the proof of optimality. However, it helps in combination with early termination.

Fix a Part of an Expression Tree. As the size of an expression tree grows, the size of k-neighbors increases. We fix the top part of an expression tree to reduce the search space. Specifically, given an incumbent and $\beta \in \mathbb{Z}_{++}$, we fix a node that has a descendant at the β-th lower generation where the first lower generation is the children and the ith generation is the children of $(i-1)$th generation. For example, when $\beta = 1$, we fix all nonleaf nodes. When $\beta = 2$, we fix all grandparents of some node.

With the implementation of early termination, there are three situations when a MINLP solver terminates at an iteration. Let $\bar{y}$ denote the current incumbent and k denote the current distance. We define the next iteration for each situation as follows:

1. The solver returns a better solution. Then, we solve a MINLP restricted by the neighbors of the returned solution.
2. The solver proves that there is no better solution within the neighbors. Then, we search in a larger neighborhood, $\mathcal{NB}_{k'}(\bar{y}) \setminus \mathcal{NB}_k(\bar{y})$ where $k' > k$.
3. The solver terminates by time limit or node limit and no better solution has been found. Then, we increase the time limit or the node limit in the next iteration.

The pseudocode of STreCH is described in Algorithms 3.1 to 3.3. Algorithm 3.1 describes a restricted MINLP. Algorithm 3.2 describes the solution-improving procedure. The inputs of Algorithm 3.2 are the solution $\bar{y}$ and the parameters that are given in the beginning and not changed during the procedure: k_{init} is the initial distance to define neighbors, $k_{\max}$ is the maximum distance of candidate neighbors, β_{init} is the initial node fix level, $\beta_{\max}$ is the maximum node fix level, γ_{init} is the initial

ALGORITHM 3.2 `Improve` (procedure to find an improved solution from a solution).

Data: (x, z), $\mathcal{P}$, $\mathcal{N}$, solution $(\bar{y}, \bar{c}, \bar{\omega})$
Result: An improved solution if found, otherwise the given solution
$(k_1, k_2, \beta, \gamma) \leftarrow (0, k_{\text{init}}, \beta_{\text{init}}, \gamma_{\text{init}})$; `// Initialize the parameters.`
repeat `// Repeatedly search neighbors.`
  $(y, c, \omega) \leftarrow$ `ResMINLP`$(x, z, \mathcal{P}, \mathcal{N}, k_1, k_2, \beta, \gamma)$;
  if $\omega < \bar{\omega}$ **then**
    return (y, c, ω) ; `// Return an improved solution.`
  else if *Terminated by node limit and* $10\gamma \leq \gamma_{\max}$ **then**
    $\gamma \leftarrow 10\gamma$; `// Spend more time on this problem.`
  else `// Spent enough time.`
    $(k_1, k_2) \leftarrow (k_2, k_2 + 2)$; `// Change the neighbor set.`
    if $k_2 > k_{\max}$ **then**
      $(k_1, k_2) \leftarrow (0, k_{\text{init}})$;
      $\beta \leftarrow \beta + 1$;
      if $\beta > \beta_{\max}$ **then**
        return $(\bar{y}, \bar{c}, \bar{\omega})$; `// Return the given solution.`
until *Time limit reached*;

ALGORITHM 3.3 STreCH.

Data: (x, z), $\mathcal{P}$, $\mathcal{N}$, $\mathcal{N}_{\text{init}}$ (the node set for the initial problem)
Result: An expression tree (y, v) and its training error ω
$(y, c, \omega) \leftarrow$ `ResMINLP`$(x, z, \mathcal{P}, \mathcal{N}_{\text{init}})$; `// Solve an initial problem.`
repeat `// Repeatedly improve a solution.`
  $(y', c', \omega') \leftarrow$ `Improve`$(x, z, \mathcal{P}, \mathcal{N}, y, c, \omega)$;
  if $\omega' < \omega$ **then**
    $(y, c, \omega) \leftarrow (y', c', \omega')$; `// Update the incumbent.`
  else
    break;
until *Time limit reached*;
return (y, c, ω);

node limit for the branch-and-bound tree, and $\gamma_{\max}$ is the maximum node limit for the branch-and-bound tree. Algorithm 3.3 describes the overall loop.

We recommend solving a single MINLP by limiting the number of nodes in the branch-and-bound tree instead of limiting time in the heuristic. The reason is that limiting the number of node guarantees that the same solution will be reproduced at each iteration whereas limiting time may return a different solution depending on how much resource is available on the computing machine.

4. Computational Experiments. We perform computational experiments on the improved formulation and the sequential tree construction heuristic. The first experiment tests our ability to find a global solution. We investigate whether the new constraints deliver an improvement in computation. The second experiment tests the ability to find a good approximated symbolic function with limited information. We assume that a limited number of observations is given and no additional information

such as the unit of variables is available. We compare our methods with AI Feynman [19, 20], which is a state-of-the-art symbolic regression solver specialized for physics formulas.

Test Problems. We test 71 formulas from the Feynman database for symbolic regression [20] whose operator set is a subset of $\{+, -, *, /, \sqrt{}\}$. Table 3 shows that all formulas can be represented by an expression tree of depth five. We assume that

Depth	1	2	3	4	5	Total
# of formulas	4	25	22	7	13	71

Table 3: Distribution of the required depth to represent a formula in the test problems.

(i) no unit information is available, (ii) we have a small number of observations (10 data points), and (iii) the observations are noisy. Although (i)–(iii) were discussed in [20] independently, the combination of all three was not discussed.

Hardware and Software. Our computational experiments are performed on a computer with Intel Xeon Gold 6130 CPU cores running at 2.10 GHz and 192 GB of memory. The operating system is Linux Ubuntu 18.04. The code is written in Julia 1.5.3 with SCIP 7.0.0 [9] as a MINLP solver that showed the best performance among open-source global MINLP solvers for this problem [4]. The code is available at https://github.com/jongeunkim/STreCH.

4.1. Comparison of MINLP Formulations. We compare the formulations described in section 2 with the formulation from [4].

4.1.1. Experimental Setup for Comparing MINLP Formulations. We start by investigating the effect of the new optional constraints that do not need to be included in the formulation but can reduce the search space. In the experiments in [4], the variance of the computational performance is large with regard to the inclusion/exclusion of optional constraints, and the authors suggest running all possible formulations in parallel. In this experiment, we consider four formulations for each method (ours and [4]) by adding or not adding redundancy-eliminating constraints and symmetry-breaking constraints. *Imp* and *Coz* stand for the improved formulation and the formulation from [4], respectively. *-F* refers to the full formulation (adding all the constraints). *-N* refers to the formulation with only the necessary constraints (tree and value defining constraints). *-R* refers to the formulation with the necessary constraints and the redundancy-eliminating constraints. *-S* refers to the formulation with the necessary constraints and the symmetry-breaking constraints. The configuration of the formulations are described in Table 4. We do not consider implication

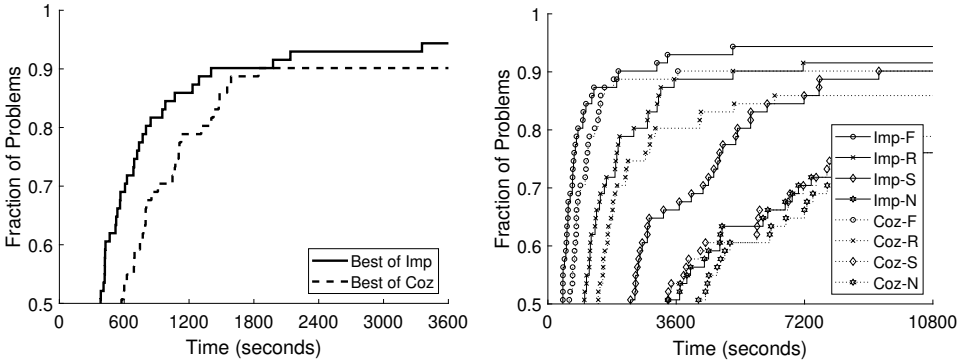
Formulations	Imp-F	Imp-R	Imp-S	Imp-N	Coz-F	Coz-R	Coz-S	Coz-N
Objective	(2.2)							
Tree	(2.3a)-(2.3b), (A.1a)-(A.1b)				(A.1a)-(A.1f)			
Value	(2.4a)-(2.4b), (A.4a)-(A.11c)				(A.2a)-(A.11c)			
Redundancy	(2.8a)-(2.8c), (A.12d)-(A.12f)		-		(A.12a)-(A.12f)		-	
Symmetry	(2.14)	-	(2.14)	-	(2.14)	-	(2.14)	-

Table 4: Formulations used in the experiments.

cuts because all the independent variables used in the test functions are positive, which means that there are no implication cuts. We limit the depth of the expression

tree to two (seven nodes) in order to find an optimal solution for all instances within the prespecified time limit (three hours).

4.1.2. Results Comparing MINLP Formulations. First, we compare our best results with the best results of [4] in Figure 8a. We collect the smallest solution times among four formulations for each method. We visualize our results using performance profiles [6] in Figure 8. Figure 8a shows that our formulations can solve 70% of instances within ten minutes while [4]'s can solve 50% of instances. Our formulations failed to solve 2.8% of instances (2 of 71) within three hours while [4]'s failed to solve 5.6% of instances (4 of 71) within the time limit.



(a) Best of four. (b) All.

Fig. 8: Comparison of solution times of MINLP solves with the improved formulations and those of [4].

Next, we compare all eight formulations in Figure 8b. Now we draw eight lines for each formulation. Figure 8b shows that our formulation with all optional constraints performs best. The formulation terminates first in more than half of the instances (36 of 71) and is in the top three in 81.7% of the instances (58 of 71). Figure 9 shows that the newly proposed optional constraints in the improved formulation also reduce the number of nodes in the branch-and-bound tree (BnBnodes) compared with the existing ones.

Our experiments show that it clearly is better to add all optional constraints to reduce the search space. This conclusion differs from the result in [4]. We believe that this difference may be due to a difference in the branch-and-bound solvers: we use SCIP while [4] uses BARON [15, 18].

4.2. STreCH and AI Feynman. The goal of this experiment is to compare the performance of our methods with the state-of-the-art symbolic regression method AI Feynman.

4.2.1. Experimental Setup for Heuristic Comparison. We test both methods with noisy data and perform cross-validation to select the final symbolic expression. We first generate a training set with noise and a validation and testing set without noise. For each method, we find a set of candidate formulas using the training set. Next, we select the formula from the candidates that has the lowest validation error. We then compute the testing error of the selected formula.

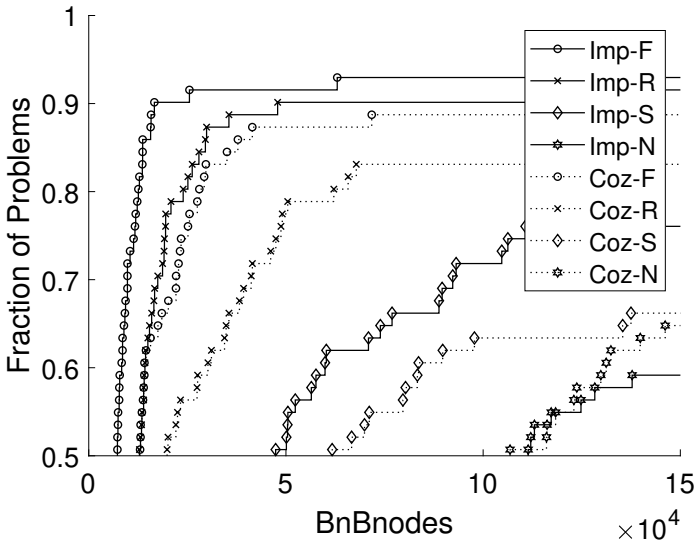


Fig. 9: Comparison of the number of nodes in the branch-and-bound tree of MINLP solves with the improved formulations and the the formulations of [4].

We consider three approaches: a STreCH-based approach, a MINLP-based approach, and AI Feynman. The first two approaches solve multiple instances with different parameter setups in parallel to collect candidate formulas. The set of parameters includes the type of formulation (adding or not adding optional constraints), the maximum depth of the expression tree, the type of constant (integer or fractional), and the bounds on the constants. When a single instance is solved, the STreCH-based approach uses the heuristic described in section 3, and the MINLP-based approach solves the MINLP problem in section 2.

We run AI Feynman ourselves because there are no reported computational experiments in our setting (running without no unit information, for a small number of observations, and noisy data). We download the AI Feynman code from https://github.com/SJ001/AI-Feynman. We use the default parameters except the set of operators used in brute-force because the default set does not include all operators used in tested functions. Instead, we use the largest operator set that includes all used operators. Note that the AI Feynman code itself manages computing resources in parallel.

4.2.2. Results for Comparison of Heuristics. First, we investigate how many formulas can be rediscovered by each method. We run all methods on the dataset with ten training data points, a noise level of 10^{-4}, and no unit information. In Table 5, we observe that every method can discover the correct formula when it can be represented by an expression tree of depth one or two. When the depth is three, the STreCH and the MINLP approaches discover twice as many formulas as AI Feynman. When the depth is four or five, the STreCH and the MINLP approaches cannot discover the original formulas, while AI Feynman discovers two formulas.

Next, we investigate formulas for which the methods return different solutions. We first consider formulas that were discovered by the STreCH and the MINLP approaches but not by AI Feynman. These include $\frac{q_1 q_2 r}{4\pi\epsilon r^3}$ (Feynman Eq. I.12.2) and

Table 5: Required depth to represent a formula.

Depth	# Formulas	Discovery rate (%)		
		STreCH	MINLP	AI Feynman
≤ 2	29	100.0	100.0	100.0
3	22	59.1	54.5	27.2
≥ 4	20	0.0	0.0	10.0

$\frac{2I}{4\pi\epsilon c^2 r}$ (Feynman Eq. II.13.17). We suspect that this difference happens because the STreCH and the MINLP approaches can assign any constant value at a node in the expression tree whereas AI Feynman relies on the user-specified particular constants. Specifically, when solving the problem of $\frac{q_1 q_2 r}{4\pi\epsilon r^3}$, the STreCH and the MINLP approaches can assign $0.159(=\frac{1}{2\pi})$ at a node while AI Feynman needs a few steps in combination with a prespecified constant (π) and operators ($x \to 2x$ and $x \to \frac{1}{x}$). These few steps might hinder the ability of the path to rediscover the original formula. This weakness also has been pointed out in [1].

Second, we investigate formulas that were discovered by AI Feynman but not by the STreCH and the MINLP approaches. These include $\frac{m_0}{\sqrt{1-v^2/c^2}}$ and $\frac{\rho_{c_0}}{\sqrt{1-v^2/c^2}}$. AI Feynman benefits from the use of trigonometrical functions, $\arcsin(\cos(x))$, which are equivalent to $\sqrt{1-x^2}$. Third, we consider formulas where the STreCH approach could find the original formula but the MINLP approach failed. These include $\frac{(h/(2\pi))^2}{2E_n d^2}$ (Feynman Eq. III.15.14). Table 6 shows how the STreCH develops a solution at each iteration. Because the formula is a monomial, there are multiple sequences to reach the correct formula. For example, the correct formula can be obtained from $\frac{h^2}{E_n d^2}$, $\frac{Ch^2}{d^2}$, and $\frac{Ch}{E_n d^2}$, where $C = (8\pi^2)^{-1}$. Therefore, there are formula structures such as a monomial that the STreCH performs well.

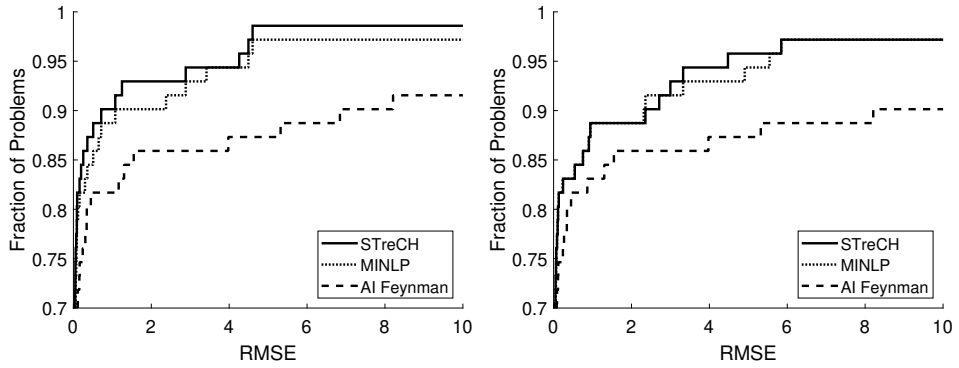
Table 6: Progress of STreCH to discover $\frac{(h/(2\pi))^2}{2E_n d^2}$.

Iteration	Incumbent*	Update	Time Spent (s)
1	$\frac{c_1 h}{d}$	initial solution**	75.33
2	$\frac{c_2 h}{E_n d}$	$h \to h/E_n$	2.82
3	$\frac{c_3 h}{E_n d^2}$	$d \to d^2$	68.39
4	$\frac{c_4 h}{E_n d^2}$	change the constant value	1.73
5	$\frac{c_5 h^2}{E_n d^2}$	$h \to h^2$	26.24
6	$\frac{c_6 h^2}{E_n d^2}$	change the constant value	161.77

* c_1-c_6 are constant values

** achieved by solving a depth-two problem

We next compare the testing errors of the three methods. We perform the computational experiments on both noiseless and noisy data. Figure 10 shows the distributions of the root mean square testing errors of the results. We see that the solutions generated by the STreCH and the MINLP approaches have a lower testing error compared with AI Feynman's solutions.



(a) Noiseless. (b) Noise level of 0.01%.

Fig. 10: Comparison of the testing errors achieved by the STreCH approach, the MINLP approach, and AI Feynman.

5. Conclusion. In this paper we present MINLP-based methods for symbolic regression. We propose an improved MINLP formulation. We also propose a new heuristic, STreCH, which is based on the tighter formulation and builds an expression tree by repeatedly modifying a solution expression tree. Compared with state-of-the-art methods, our methods are able to discover more correct formulas when there is a lack of data. When an original formula is difficult to rediscover, our methods return a formula that has a lower testing error.

For future work, our method can be integrated with AI Feynman. AI Feynman decomposes to small problems and solves those using polynomial fit and brute-force methods. Since our method is good at finding a relatively simple symbolic expression, it would be a good option for solving small subproblems that arise within the AI Feynman decomposition within a tight time limit, of, say, less that a minute.

Acknowledgements. This work was supported by the Applied Mathematics activity within the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

Appendix A. The MINLP Formulation Proposed in [4].

The formulation proposed by [4] is written in terms of our notations.

A.1. Tree-Defining Constraints. The constraints are (1d)–(1i) in [4].

$$\sum_{o\in\mathcal{O}} y_n^o \le 1, \qquad n \in \mathcal{N}, \tag{A.1a}$$

$$\sum_{n\in\mathcal{N}}\sum_{j=1}^{d} y_n^{x_j} \ge 1, \tag{A.1b}$$

$$\sum_{o\in\mathcal{B}\cup\mathcal{U}} y_n^o \le \sum_{o\in\mathcal{O}} y_{2n+1}^o, \qquad n \notin \mathcal{T}, \tag{A.1c}$$

$$\sum_{o\in\mathcal{B}} y_n^o \le \sum_{o\in\mathcal{O}} y_{2n}^o, \qquad n \notin \mathcal{T}, \tag{A.1d}$$

$$\sum_{o\in\mathcal{U}\cup\mathcal{L}} y_n^o \le 1 - \sum_{o\in\mathcal{O}} y_{2n}^o, \qquad n \notin \mathcal{T}, \tag{A.1e}$$

$$\sum_{o\in\mathcal{L}} y_n^o \le 1 - \sum_{o\in\mathcal{O}} y_{2n+1}^o, \qquad n \notin \mathcal{T}. \tag{A.1f}$$

A.2. Value-Defining Constraints. The constraints are (1b) and (1c) in [4].
No Assignment.

$$v_{i,n} \le v_{\text{up}}(1 - \sum_{o\in\mathcal{O}} y_n^o), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}, \tag{A.2a}$$

$$v_{i,n} \ge v_{\text{lo}}(1 - \sum_{o\in\mathcal{O}} y_n^o), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}. \tag{A.2b}$$

Independent Variables.

$$v_{i,n} \le x_{i,j} y_n^{x_j} + v_{\text{up}}(1 - y_n^{x_j}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N},\ \forall j \in [d], \tag{A.3a}$$

$$v_{i,n} \ge x_{i,j} y_n^{x_j} + v_{\text{lo}}(1 - y_n^{x_j}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N},\ \forall j \in [d]. \tag{A.3b}$$

Constant.

$$v_{i,n} - c_n \le (v_{\text{up}} - c_{\text{lo}})(1 - y_n^{\text{cst}}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}, \tag{A.4a}$$

$$v_{i,n} - c_n \ge (v_{\text{lo}} - c_{\text{up}})(1 - y_n^{\text{cst}}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \in \mathcal{N}. \tag{A.4b}$$

Addition.

$$v_{i,n} - (v_{i,2n} + v_{i,2n+1}) \le (v_{\text{up}} - 2v_{\text{lo}})(1 - y_n^{+}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.5a}$$

$$v_{i,n} - (v_{i,2n} + v_{i,2n+1}) \ge (v_{\text{lo}} - 2v_{\text{up}})(1 - y_n^{+}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.5b}$$

Subtraction.

$$v_{i,n} - (v_{i,2n} - v_{i,2n+1}) \le (2v_{\text{up}} - v_{\text{lo}})(1 - y_n^{-}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.6a}$$

$$v_{i,n} - (v_{i,2n} - v_{i,2n+1}) \ge (2v_{\text{lo}} - v_{\text{up}})(1 - y_n^{-}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.6b}$$

Multiplication.

$$v_{i,n} - v_{i,2n}v_{i,2n+1} \le (v_{\text{up}} - \min\{v_{\text{lo}}^2, v_{\text{lo}}v_{\text{up}}, v_{\text{up}}^2\})(1 - y_n^{*}), \quad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.7a}$$

$$v_{i,n} - v_{i,2n}v_{i,2n+1} \ge (v_{\text{lo}} - \max\{v_{\text{lo}}^2, v_{\text{up}}^2\})(1 - y_n^{*}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.7b}$$

Division.

$$v_{i,n}v_{i,2n+1} - v_{i,2n} \le (\max\{v_{\text{lo}}^2, v_{\text{up}}^2\} - v_{\text{lo}})(1 - y_n^{/}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.8a}$$

$$v_{i,n}v_{i,2n+1} - v_{i,2n} \ge (\min\{v_{\text{lo}}^2, v_{\text{lo}}v_{\text{up}}, v_{\text{up}}^2\} - v_{\text{up}})(1 - y_n^{/}), \quad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.8b}$$

$$\epsilon y_n^{/} \le v_{i,2n}^2, \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.8c}$$

$$\epsilon y_n^{/} \le v_{i,2n+1}^2, \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.8d}$$

Square Root.

$$v_{i,n}^2 - v_{i,2n+1} \le (\max\{v_{\text{lo}}^2, v_{\text{up}}^2\} - v_{\text{lo}})(1 - y_n^{\surd}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.9a}$$

$$v_{i,n}^2 - v_{i,2n+1} \ge (-v_{\text{up}})(1 - y_n^{\surd}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.9b}$$

$$\epsilon - v_{i,2n+1} \le (\epsilon - v_{\text{lo}})(1 - y_n^{\surd}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.9c}$$

Exponential.

$$v_{i,n} - \exp(v_{i,2n+1}) \le v_{\text{up}}(1 - y_n^{\exp}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.10a}$$

$$v_{i,n} - \exp(v_{i,2n+1}) \ge (v_{\text{lo}} - \exp(v_{\text{up}}))(1 - y_n^{\exp}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.10b}$$

Logarithm.

$$\exp(v_{i,n}) - v_{i,2n+1} \le (\exp(v_{\text{up}} - v_{\text{lo}})(1 - y_n^{\log}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.11a}$$

$$\exp(v_{i,n}) - v_{i,2n+1} \ge (-v_{\text{up}})(1 - y_n^{\log}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}, \tag{A.11b}$$

$$\epsilon - v_{i,2n+1} \le (\epsilon - v_{\text{lo}})(1 - y_n^{\log}), \qquad \forall i \in [n_{\text{data}}],\ \forall n \notin \mathcal{T}. \tag{A.11c}$$

A.3. Redundancy-Eliminating Constraints. These constraints are (2a), (2b), (3a), (3b), (4a), and (4b) in [4]. Define $\mathcal{O}_{\text{pair}}$ as the set of all inverse unary operation pairs o and o' such as $(\exp, \log)$, and $((\cdot)^2, \surd)$.

$$y_{2n+1}^{\text{cst}} + \sum_{o \in \mathcal{U}} y_n^o \le 1, \qquad n \notin \mathcal{T}, \tag{A.12a}$$

$$y_{2n+1}^{\text{cst}} + y_n^{-} \le 1, \qquad n \notin \mathcal{T}, \tag{A.12b}$$

$$y_{2n+1}^{\text{cst}} + y_n^{/} \le 1, \qquad n \notin \mathcal{T}, \tag{A.12c}$$

$$y_{2n}^{\text{cst}} + y_{2n+1}^{\text{cst}} \le 1, \qquad n \notin \mathcal{T}, \tag{A.12d}$$

$$y_n^o + y_{2n+1}^{o'} \le 1, \qquad n \notin \mathcal{T},\ (o, o') \in \mathcal{O}_{\text{pair}}, \tag{A.12e}$$

$$y_n^{o'} + y_{2n+1}^o \le 1, \qquad n \notin \mathcal{T},\ (o, o') \in \mathcal{O}_{\text{pair}}. \tag{A.12f}$$

A.4. Symmetry-Breaking Constraints. This constraint is (5) in [4].

$$v_{1,2n} - v_{1,2n+1} \ge (v_{\text{lo}} - v_{\text{up}})(1 - y_n^{+} - y_n^{*}), \qquad n \in \mathcal{N}_{\text{perfect}} \tag{A.13}$$

Appendix B. Proof of Lemma 2.2.

Proof. We first show (2.6a). Note that $\mathcal{F}_B = \{e_i,\ \forall i \in [m]\}$, where e_i is the ith standard unit vector where the ith element is one, and otherwise zero. Equality (2.6a) can be shown by

$$\begin{aligned}\{(x, y) \in S \mid y \in \mathcal{F}_B\} &= \bigcup_{i=1}^{m} (S \cap \{(x, y) \mid y = e_i\}) \\ &= \bigcup_{i=1}^{m} (T \cap \{(x, y) \mid y = e_i\}) = \{(x, y) \in T \mid y \in \mathcal{F}_B\}.\end{aligned}$$

The equality in the middle holds because of the following observations:

$$S \cap \{(x, y) \mid y = e_i\} = \begin{cases} \{(x, e_i) \mid f(x) \le w_i\} & \text{if } i \in [k], \\ \{(x, e_i) \mid f(x) \le M\} & \text{otherwise,} \end{cases}$$

$$T \cap \{(x,y) \mid y = e_i\} = \begin{cases} \{(x,e_i) \mid f(x) \le \min\{w_i, M\} = w_i\} & \text{if } i \in [k], \\ \{(x,e_i) \mid f(x) \le M\} & \text{otherwise.} \end{cases}$$

We next show (2.6b). Let $S_C := \{(x,y) \in S \mid y \in \mathcal{F}_C\}$ and $T_C := \{(x,y) \in T \mid y \in \mathcal{F}_C\}$. First, it holds that $S_C \subseteq T_C$ because the constraint in S dominates every constraint in T. It is sufficient to show that

$$\begin{aligned} \sum_{i\in[k]} w_i y_i + M(1 - \sum_{i\in[k]} y_i) &= w_j y_j + \sum_{i\in[k]:i\neq j} w_i y_i + M(1 - \sum_{i\in[k]} y_i) \\ &\le w_j y_j + \sum_{i\in[k]:i\neq j} M y_i + M(1 - \sum_{i\in[k]} y_i) = w_j y_j + M(1 - y_j), \end{aligned}$$

for all $j \in [k]$. Second, we show that there exists $(x,y) \in T_C \setminus S_C$. Let $j = \arg\min_{i\in[k]} w_i$. Consider $(\bar{x}, \bar{y})$ with $\bar{x} = \frac{1}{m} w_j + \frac{m-1}{m} M$ and $\bar{y}_i = \frac{1}{m}$ for all i. Point $(\bar{x}, \bar{y})$ is in $\mathcal{F}_C$. Point $(\bar{x}, \bar{y})$ is not in S because

$$\begin{aligned} &\bar{x} - \left(\sum_{i\in[k]} w_i \bar{y}_i + M(1 - \sum_{i\in[k]} \bar{y}_i) \right) \\ &= \left(\frac{1}{m} w_j + \frac{m-1}{m} M \right) - \left(\frac{\sum_{i\in[k]} w_i}{m} + \frac{m-k}{m} M \right) = \frac{\sum_{i\in[k]:i\neq j} (M - w_i)}{m} > 0, \end{aligned}$$

while the point is in T because

$$\begin{aligned} \bar{x} - (w_i \bar{y}_i + M(1 - \bar{y}_i)) &= \left(\frac{1}{m} w_j + \frac{m-1}{m} M \right) - \left(\frac{1}{m} w_i + \frac{m-1}{m} M \right) \\ &= \frac{1}{m}(w_j - w_i) \le 0 \end{aligned}$$

for all $i \in [k]$. Therefore, $(\bar{x}, \bar{y}) \in T_C \setminus S_C$, which completes the proof. ∎

REFERENCES

[1] V. Austel, C. Cornelio, S. Dash, J. Goncalves, L. Horesh, T. Josephson, and N. Megiddo, *Symbolic regression using mixed-integer nonlinear optimization*, 2020, https://arxiv.org/abs/2006.06813.

[2] V. Austel, S. Dash, O. Gunluk, L. Horesh, L. Liberti, G. Nannicini, and B. Schieber, *Globally optimal symbolic regression*, in NIPS 2017 Symposium on Interpretable Machine Learning, 2017, https://arxiv.org/abs/1710.10720.

[3] A. Cozad, *Data- and theory-driven techniques for surrogate-based optimization*, PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, 2014.

[4] A. Cozad and N. V. Sahinidis, *A global MINLP approach to symbolic regression*, Mathematical Programming, 170 (2018), pp. 97–119, https://doi.org/10.1007/s10107-018-1289-x.

[5] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, *Discovering symbolic models from deep learning with inductive biases*, 2020, https://arxiv.org/abs/2006.11287.

[6] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical programming, 91 (2002), pp. 201–213, https://doi.org/10.1007/s101070100263.

[7] T. Duriez, S. L. Brunton, and B. R. Noack, *Machine learning control-taming nonlinear dynamics and turbulence*, Springer, 2017, https://doi.org/10.1007/978-3-319-40624-4.

[8] M. Fischetti and A. Lodi, *Local branching*, Mathematical programming, 98 (2003), pp. 23–47, https://doi.org/10.1007/s10107-003-0395-5.

[9] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig, *The SCIP Optimization Suite 6.0*, Tech. Report 18-26, Zuse Institute Berlin, 2018, https://nbn-resolving.de/urn:nbn:de:0297-zib-69361.

[10] L. Horesh, L. Liberti, and H. Avron, *Globally optimal minlp formulation for symbolic regression*, tech. report, IBM Research, 2016, https://dominoweb.draco.res.ibm.com/reports/rc25620.pdf.

[11] L. Kammerer, G. Kronberger, B. Burlacu, S. M. Winkler, M. Kommenda, and M. Affenzeller, *Symbolic regression by exhaustive search: Reducing the search space using syntactical constraints and efficient semantic structure deduplication*, in Genetic Programming Theory and Practice XVII, Springer, 2020, pp. 79–99, https://doi.org/10.1007/978-3-030-39958-0_5.

[12] G. Kronberger, L. Kammerer, B. Burlacu, S. M. Winkler, M. Kommenda, and M. Affenzeller, *Cluster analysis of a symbolic regression search space*, in Genetic Programming Theory and Practice XVI, Springer, 2019, pp. 85–102, https://doi.org/10.1007/978-3-030-04735-1_5.

[13] P. Neumann, L. Cao, D. Russo, V. S. Vassiliadis, and A. A. Lapkin, *A new formulation for symbolic regression to identify physico-chemical laws from experimental data*, Chemical Engineering Journal, (2019), p. 123412, https://doi.org/10.1016/j.cej.2019.123412.

[14] M. Nicolau and J. McDermott, *Genetic programming symbolic regression: What is the prior on the prediction?*, in Genetic Programming Theory and Practice XVII, Springer, 2020, pp. 201–225, https://doi.org/10.1007/978-3-030-39958-0_11.

[15] N. V. Sahinidis, *BARON 17.8.9: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2017, https://sahinidis.coe.gatech.edu/baron.

[16] M. Schmidt and H. Lipson, *Distilling free-form natural laws from experimental data*, Science, 324 (2009), pp. 81–85, https://doi.org/10.1126/science.1165893.

[17] B. Tarawneh, W. A. Bodour, and K. Al Ajmi, *Intelligent computing based formulas to predict the settlement of shallow foundations on cohesionless soils*, The Open Civil Engineering Journal, 13 (2019), https://doi.org/10.2174/1874149501913010001.

[18] M. Tawarmalani and N. V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Mathematical Programming, 103 (2005), pp. 225–249, https://doi.org/10.1007/s10107-005-0581-8.

[19] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, *AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity*, 2020, https://arxiv.org/abs/2006.10782.

[20] S.-M. Udrescu and M. Tegmark, *AI Feynman: A physics-inspired method for symbolic regression*, Science Advances, 6 (2020), p. eaay2631, https://doi.org/10.1126/sciadv.aay2631.

[21] Y. Wang, N. Wagner, and J. M. Rondinelli, *Symbolic regression in materials science*, MRS Communications, 9 (2019), pp. 793–805, https://doi.org/10.1557/mrc.2019.85.

[22] D. R. White, J. Mcdermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaundefinedkowski, U.-M. O'Reilly, and S. Luke, *Better GP benchmarks: Community survey results and proposals*, Genetic Programming and Evolvable Machines, 14 (2013), pp. 3—-29, https://doi.org/10.1007/s10710-012-9177-2.