Symbolic Regression is NP-hard

Anonymous authors
Paper under double-blind review

Abstract

Symbolic regression (SR) is the task of learning a model of data in the form of a mathematical expression. By their nature, SR models have the potential to be accurate and human-interpretable at the same time. Unfortunately, finding such models, i.e., performing SR, appears to be a computationally intensive task. Historically, SR has been tackled with heuristics such as greedy or genetic algorithms and, while some works have hinted at the possible hardness of SR, no proof has yet been given that SR is, in fact, NP-hard. This begs the question: Is there an exact polynomial-time algorithm to compute SR models? We provide evidence suggesting that the answer is probably negative by showing that SR is NP-hard.

1 Introduction

Symbolic regression (SR) is a sub-field of machine learning concerned with discovering a model of the given data in the form of a mathematical expression (or equation) (Koza, 1994; Schmidt & Lipson, 2009). For example, consider having measurements of planet masses m_1 and m_2, the distance r between them, and the respective gravitational force F. Then, an SR algorithm would ideally re-discover the well-known expression (or an equivalent formulation thereof) $F = G \times \frac{m_1 m_2}{r^2}$, with $G = 6.6743 \times 10^{-11}$, by opportunely combining the mathematical operations (here, of multiplication and division) with the variables and constant at play.

The appeal of learning models as mathematical expressions goes beyond obtaining predictive power alone, as is commonplace in machine learning. In fact, SR models are particularly well suited for human interpretability and in-depth analysis (Otte, 2013; Virgolin et al., 2021b; La Cava et al., 2021). This aspect enables a safe and responsible use of machine learning models for high-stakes societal applications, as requested in the AI acts by the European Union and the United States (European Commission, 2021; 117th US Congress, 2022; Jobin et al., 2019). Moreover, it enables scientists to gain deeper knowledge about the phenomena that underlie the data. Consequently, SR enjoys wide applicability: SR has successfully been applied to astrophysics (Lemos et al., 2022), chemistry (Hernandez et al., 2019), control (Derner et al., 2020), economics (Verstyuk & Douglas, 2022), mechanical engineering (Kronberger et al., 2018), medicine (Virgolin et al., 2020b), space exploration (Märtens & Izzo, 2022), and more (Matsubara et al., 2022).

As we will describe in Sec. 2, many different algorithms have been proposed to address SR, ranging from genetic algorithms to deep learning ones. Existing algorithms either lack optimality guarantees or heavily restrict the space of SR models to consider. In fact, there is a wide belief in the community that SR is an NP-hard problem[1] (Lu et al., 2016; Petersen et al., 2019; Udrescu & Tegmark, 2020; Li et al., 2022). However, to the best of our knowledge, this belief had yet to be solidified in the form of a proof prior to the advent of this paper. Indeed, we prove that there exist instances of the SR problem for which one cannot discover the best-possible mathematical expression in polynomial time. Id est, SR is an NP-hard problem.

2 Background: Existing SR algorithms

The introduction of SR is generally attributed to John R. Koza (e.g., Zelinka et al. (2005) make this claim); however, the problem of finding a mathematical expression or equation that explains empirical measurements

[1] Lu et al. (2016) state that SR is NP-hard but provide no reference nor proof.

was already considered in earlier works (Gerwin, 1974; Langley, 1981; Falkenhainer & Michalski, 1986). Such works build mathematical expressions by iterative application of multiple heuristic tests on the data.

Koza is best known for his pioneering work on genetic programming (GP), i.e., the form of evolutionary computation where candidate solutions are variable-sized and represent programs (Koza et al., 1989; Koza, 1990; 1994). Early forms of GP where proposed by Cramer (1985); Hicklin (1986). Koza showed that GP can be used to discover SR models by encoding mathematical expressions as computational trees (see Fig. 1). In such trees, internal nodes represent functions (e.g., $+$, $-$, $\times$, etc.) that are drawn from a pre-decided set of possibilities, and leaf nodes represent variables or constants (e.g., x_1, x_2, ..., -1, π, etc.). GP evolves a population of trees by initially sampling random trees, and then conducts the following steps: (1) stochastic replacement and recombination of their sub-trees; (2) evaluation of the fitness by executing the trees and assessing their output; and (3) stochastic survival of the fittest.

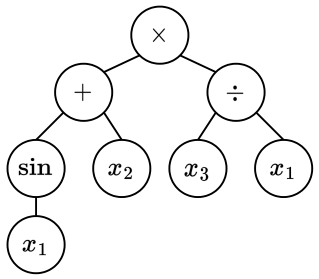

Figure 1: Example of a tree that encodes $f(\mathbf{x}) = (\sin(x_1) + x_2) \times x_3/x_1$.

Recently, La Cava et al. (2021) proposed *SRBench*, a benchmarking platform for SR that includes more than 20 algorithms have been applied on more than 250 data sets. SRBench shows that several state-of-the-art algorithms for SR are GP-based. Among these, at the time of writing, *Operon* by Burlacu et al. (2020) was found to perform best in terms of discovering accurate SR models; and *GP-GOMEA* by Virgolin et al. (2021a) was found to perform best in terms of discovering decently-accurate and relatively-simple SR models (i.e., shorter mathematical expressions). Other forms of GP, such as *strongly-typed* GP (Montana, 1995), *grammar-guided* GP (McKay et al., 2010), and *grammatical evolution* (O'Neill & Ryan, 2001), are often used to tackle *dimensionally-aware* SR, i.e., the search of mathematical expressions with constraints to achieve meaningful combinations of units of measurement.

SR has been addressed with many other types of algorithms than genetic ones, oftentimes in order to obtain a deterministic behavior. Worm & Chiu (2013) and Kammerer et al. (2020) proposed enumeration algorithms which make SR tractable by restricting the space of possible models to consider and including dynamic programming and pruning strategies. Cozad (2014); Cozad & Sahinidis (2018) showed how SR can be addressed with mixed integer nonlinear programming. McConaghy (2011) proposed *FFX*, which generates a linear combination of many functions that are linearly-independent from each other, and then fits its coefficients with the *elastic net* (Zou & Hastie, 2005) to promote sparsity. Olivetti de França (2018) and Rivero et al. (2022) propose greedy algorithms that start from small mathematical expressions and iteratively expand them, by replacing existing components with larger ones from a set of possibilities.

Lastly, recent years have seen the proposal of deep learning-based algorithms for SR. Petersen et al. (2020) cast the SR problem as a reinforcement learning one and train a recurrent neural network to generate accurate SR models. Udrescu & Tegmark (2020) leverage neural networks in order to test for symmetries and invariances in the data that are then used to prune the space of possible SR models. An end-to-end approach is taken by Kamienny et al. (2022) and Vastl et al. (2022), who train deep neural transformers to produce SR models directly from the data. Li et al. (2022) seek SR models by proposing a convexified formulation of deep reinforcement learning.

In summary, existing SR algorithms are either heuristics, which do not guarantee optimality (e.g., genetic, greedy, or deep learning-based algorithms), or they are exact algorithms that achieve optimality but only over a small subset of all possible SR models, to limit the runtime (e.g., dynamic programming and mixed-integer nonlinear programming algorithms). This strongly hints to the fact that SR is NP-hard. As mentioned earlier, no proof has yet been given.

3 Preliminaries

We will hereon refer to SR models as *functions* when appropriate, as this is their fundamental nature. To begin, let us recall the concept of function composition, which is central to SR.

Definition 1. Function composition

Given two functions $f : \mathbb{A} \to \mathbb{B}$ and $g : \mathbb{B} \to \mathbb{C}$, function composition, which we denote by $g \circ f$, is the operation that produces a third function $h : \mathbb{A} \to \mathbb{C}$, such that $h(x) = g(f(x))$.

Thanks to function composition, we can now define the concept of *search space* of an SR problem.

Definition 2. Search space of SR

Let $\mathcal{P}$ be a set of functions and variables. The search space of SR is the function space $\mathcal{F}$ that contains all functions that can be formed by composition of the elements of $\mathcal{P}$ and their compositions.

To better understand what Def. 2 states, consider that $\mathcal{P}$ can be set to contain a mix of functions that perform basic algebraic operations such as addition, subtraction, multiplication, and division; transcendental functions such as sin, cos, log, exp; constant functions (or simply *constants*), such as $c_{42}(x) = 42$ and $c_\pi(x) = \pi$ for any x; and variables of interest for the problem at hand, such as x_1, x_2, x_3. $\mathcal{P}$ is typically referred to as the *primitive set*, and its elements as *primitives* (Poli et al., 2008). Once $\mathcal{P}$ has been decided, $\mathcal{F}$ is determined. For example, choosing $\mathcal{P} = \{+(\cdot,\cdot), -(\cdot,\cdot), \times(\cdot,\cdot), x_1, x_2, -1, +1\}$ means that $\mathcal{F}$ will contain a subset of all possible polynomials of arbitrary degree in x_1 and x_2. In particular, $\mathcal{F}$ is a subset because only some coefficients can be expressed, by composing constants with addition, subtraction, and multiplication.

Let us clarify a point regarding constants in particular. Normally, one would include constants considered to be relevant to the instance of SR at hand. For example, if the unknown phenomenon for which an SR model is sought is suspected to have sinusoidal components, it may be advisable to include multiples of π in $\mathcal{P}$. Moreover, $\mathcal{P}$ can be set to contain special elements that represent probability distributions from which constants can be sampled (see the concept of *ephemeral random constant* described by Koza (1994); Poli et al. (2008)). We denote one such element by $\mathfrak{R}$ and, e.g., $\mathfrak{R}$ can be chosen to represent the uniform distribution between two numbers, or the normal distribution with a certain mean and variance. When an SR algorithm picks $\mathfrak{R}$ from $\mathcal{P}$ to compose an SR model, a constant is sampled from the distribution identified by $\mathfrak{R}$. Here (more specifically, in Corollary 1) we will generously assume that any constant can be sampled directly from $\mathfrak{R}$, and therefore that there is no need for a real-valued optimizer to be part of the SR algorithm. For example, having $\mathcal{P} = \{+(\cdot,\cdot), -(\cdot,\cdot), \times(\cdot,\cdot), x_1, x_2, \mathfrak{R}\}$ will mean that $\mathcal{F}$ contains *all* polynomials of arbitrary degree in x_1 and x_2.

We can now proceed by providing a definition of the SR problem. While this definition can be extended to other domains, we focus on handling real-valued numbers as the majority of the works takes place in this domain, and subsets thereof.

Definition 3. Symbolic Regression (SR) problem

Given a set $\mathcal{P}$ of functions and variables, a metric $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, vectors $\mathbf{x}_i = (x_{1,i}, \ldots, x_{d,i}) \in \mathbb{R}^d$ and scalars $y_i \in \mathbb{R}$, for $i = 1, \ldots, n$, the SR problem asks for finding a function $f^\star$ such that:

$$f^\star = \arg\min_{f \in \mathcal{F}} \mathcal{L}\left(\mathbf{y}, f(\mathbf{x})\right), \tag{1}$$

where $\mathcal{F}$ is the search space that is defined by $\mathcal{P}$.

We provide some remarks concerning the proposed definition of the SR problem. Firstly, let us map the objects provided in the definition to terms familiar to a machine learning audience. The pair $(\mathbf{x}_i, y_i)$ is normally what is referred to as *observation*, *data point*, *example*, or *sample*, where $x_{j,i}$ is the value of the jth *feature* or *variable* for the ith observation, and y_i is the value of the *label* or *target variable* for the same observation. The set that contains the observations upon which $\mathcal{L}$ is computed, i.e., $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, is called *training set*. Moreover, the metric $\mathcal{L}$ is called *loss function*.

Normally, we actually desire f to *generalize* to *new* (or also called *unseen*) observations, i.e., observations which are similar to those in $\mathcal{D}$ but not exactly the same (they come from the same underlying and unknown distribution). In other words, it is not sufficient that $f^\star$ is a best-possible function with respect to the training set, as the loss should remain minimal also for new observations that are not available to us. Still, considering a "pure optimization" formulation, as given in Eq. (1), can be considered to be a pre-requisite for being able to machine-learn accurate models from the data; in fact, much literature that concerns the generation of

provably-optimal models provides proofs with respect to the training set alone (see, e.g., results for decision trees (Hu et al., 2019)). In a similar fashion, here we will consider the case of minimizing the loss with respect to the training set $\mathcal{D}$ and show that this is already problematic for any SR algorithm.

As loss function, we consider $\mathcal{L}$ to be a metric (i.e., distance) which operates between the output $f(\mathbf{x}_i)$ and the label y_i across $i = 1, \ldots, n$. Commonly-used loss functions such as the *mean absolute error*, *mean squared error*, and *root mean squared error* fit this definition. However, certain works include regularization terms in the loss function, such as $\lambda \times C(f)$, where $\lambda \in \mathbb{R}$ controls the regularization strength and $C : \mathcal{F} \to \mathbb{R}$ is a function of the complexity of f. Typical goals of such regularization terms are improving generalization (by limiting effects akin to Runge's phenomenon (Fornberg & Zuev, 2007)) and improving the interpretability of f. For the latter, implementations of C range from weighed counting of the number of primitives that constitute f (Ekart & Nemeth, 2001; Hein et al., 2018), to machine learning models trained from human feedback to predict f's interpretability (Virgolin et al., 2020a; 2021b). Here, for simplicity, we focus on $\mathcal{L}$ being a plain metric as stated in Def. 3 or, equivalently put, we consider $\lambda = 0$.

Lastly on Def. 3, we consider only cases in which f is *not* a recursive function, which to the best of our knowledge is the case for the majority of the literature on SR. Recursive function discovery is an interesting topic in general (see, e.g., d'Ascoli et al. (2022)), but it is not interesting here because recursive functions can take exponential time to compute (consider, e.g., Fibonacci's sequence). Therefore, it is obvious that the SR problem cannot be solved in polynomial time if certain recursive functions can be considered. Here, we will assume that computing $f(\mathbf{x})$ and $\mathcal{L}(\mathbf{y}, f(\mathbf{x}))$ can be done in polynomial time. Regarding $\mathcal{L}$, our assumption is met for all commonly-used metrics by which $\mathcal{L}$ is implemented (mean absolute error, mean squared error, variants thereof with margins, etc.). In fact, computing losses of such form takes $O(n)$ time, i.e., the runtime is linear in the number of observations. Regarding the computation of $f(\mathbf{x})$, f itself can be implemented as a directed acyclic graph, where nodes represent the functions and variables from $\mathcal{P}$, and edges represent compositions. To compute $f(\mathbf{x})$, it suffices to visit each node of the graph for each observation, thus requiring $O(\ell \times n)$, where ℓ is the number of primitives in f. Fig. 1 shows an example of such a graph, especially in the form of a *tree*, which is perhaps the most common way of encoding mathematical expressions in SR (see, e.g., the SR algorithms benchmarked by La Cava et al. (2021)).

We conclude this section with the following important definition.

Definition 4. Decision version of the SR problem (SR-Dec)

Given an SR instance and an $\epsilon \in \mathbb{R}_0^+$, SR-Dec outputs YES *if and only if:*

$$\exists f \in \mathcal{F} : \mathcal{L}\left(\mathbf{y}, f(\mathbf{x})\right) \leq \epsilon. \tag{2}$$

Essentially, Def. 4 is the problem of deciding whether there exists a function f in the search space such that its loss is smaller than a chosen threshold ϵ.

4 The result

We proceed directly by providing the main result of this paper.

Theorem 1. *The SR problem is NP-hard.*

Proof. Let us begin by stating that SR-Dec is in NP. Recall that the computations of $f(\mathbf{x})$ and $\mathcal{L}(\mathbf{y}, f(\mathbf{x}))$ take polynomial time (see Sec. 3). Of course, the check $\leq \epsilon$ takes $O(1)$ time. Thus, if f is guessed by an oracle, then we can provide an answer to SR-Dec in polynomial time.

We proceed by considering the unbounded subset sum problem (USSP). USSP is a similar problem to the unbounded knapsack problem, where a same item can be put in the knapsack an arbitrary number of times, and the weight of an item corresponds exactly to the profit gained by including that item in the knapsack. The decision version of USSP, USSP-Dec, is defined as follows. Given $j = 1, \ldots, k$ (k items), $w_j \in \mathbb{N}$ (weight

of that item), and $t \in \mathbb{N}$ (the target), USSP-Dec asks:

$$\exists \mathbf{m} : \sum_{j=1}^{k} w_j m_j = t? \tag{3}$$

where $m_j \in \mathbb{N}_0$ (multiplicity with which an item is picked). USSP-Dec is known to be NP-complete (Kellerer et al., 2004).

To prove that SR-Dec is NP-complete, we show that any instance of USSP-Dec can be reduced to some instance of SR-Dec in polynomial time. To this end, we will restrict SR-Dec as follows: (1) We pick the set of primitives $\mathcal{P}$ to be $\mathcal{P} = \{+, x_1, \ldots, x_d\}$; (2) We set $\epsilon = 0$. In other words, we set the search space $\mathcal{F}$ to contain only linear sums of the features in the data set $\mathcal{D}$, i.e., functions of the form $f(\mathbf{x}) = \sum_{j=1}^{d} x_j m_j$ with $m_j \in \mathbb{N}_0$. SR-Dec will output YES if and only if there exists such a function in $\mathcal{F}$ that achieves zero loss, i.e., it perfectly interpolates all observations in $\mathcal{D}$.

Next, we craft $\mathcal{D}$ to have a single observation ($n = 1$) and k features ($d = k$). For the only observation in $\mathcal{D}$ (dropping the index for the observation number, since there is only one), we set $x_1 = w_1, x_2 = w_2, \ldots, x_k = w_k$, and $y = t$.

Then, the following holds:

$$\begin{aligned}
&\textit{(Eq. (2))} && \exists f \in \mathcal{F} : \mathcal{L}\left(y, f(\mathbf{x})\right) \leq \epsilon? && (4)\\
&\textit{(Choosing } \epsilon = 0\textit{)} && \exists f \in \mathcal{F} : \mathcal{L}\left(y, f(\mathbf{x})\right) \leq 0? && (5)\\
&\textit{(}\mathcal{L}\left(y, f(\mathbf{x})\right) = 0 \iff f(\mathbf{x}) = y\textit{)} && \exists f \in \mathcal{F} : f(\mathbf{x}) = y? && (6)\\
&\textit{(Equivalence } y = t \textit{ due to } \mathcal{D}\textit{)} && \exists f \in \mathcal{F} : f(\mathbf{x}) = t? && (7)\\
&\textit{(Expanding } \mathcal{F} \textit{ based on choice of } \mathcal{P}\textit{)} && \exists f \in \left\{ \sum_{j=1}^{d} x_j m_j : m_j \in \mathbb{N}_0 \right\} : f(\mathbf{x}) = t? && (8)\\
&\textit{(Equivalence } x_j = w_j, d = k \textit{ due to } \mathcal{D}\textit{)} && \exists f \in \left\{ \sum_{j=1}^{k} w_j m_j : m_j \in \mathbb{N}_0 \right\} : f(\mathbf{x}) = t? && (9)\\
&\textit{(Re-formulating in terms of } \mathbf{m}\textit{)} && \exists \mathbf{m} : \sum_{j=1}^{k} w_j m_j = t? && (10)
\end{aligned}$$

In other words, there exist some instances of SR-Dec that can be re-formulated as USSP-Dec (cfr. Eqs. (3) and (10)). Now, since assembling $\mathcal{P}$ as stated above takes linear time in k, picking $\epsilon = 0$ takes $\mathcal{O}(1)$ time, and constructing $\mathcal{D}$ as stated above takes linear time in k, then any instance of USSB-Dec can be reduced to some instance of SR-Dec in polynomial time: SR-Dec is NP-complete.

We conclude the proof with a *reductio ab absurdum*. Let us assume that there exists an algorithm to compute an optimal $f^\star$ for the SR problem (Def. 3) in polynomial time. An optimal $f^\star$ is the one for which the loss is minimal, which means that using $f^\star$ in Eq. (2) allows us to immediately answer SR-Dec. Since verifying that $\mathcal{L}(\mathbf{y}, f^\star(\mathbf{x})) \leq \epsilon$ takes polynomial time, we conclude that if the SR problem can be solved in polynomial time, then we can also solve SR-Dec in polynomial time. Therefore, the SR problem is NP-hard.

□

We remark that, in the proof of Theorem 1, we construct $\mathcal{P}$ so as not to contain $\mathfrak{R}$ (nor any constant). Some readers might disagree with this quite broad definition of SR. In fact, some SR algorithms heavily rely on the presence of constants as well as on their optimization (e.g., *FFX* by McConaghy (2011) and *FEAT* by La Cava et al. (2018)). Not allowing for arbitrary constants to be present in the functions of the search space might be seen as a violation of the very definition of SR. In other words, some might think that $\mathcal{P}$ *must* contain $\mathfrak{R}$. We next show that SR remains NP-hard in this special case.

Corollary 1. *The SR problem is NP-hard even when $\mathcal{P}$ must include $\mathfrak{R}$.*

Proof. We follow a similar construction of the proof of Theorem 1. Namely, the only difference from before is in the way we pick $\mathcal{P}$ and construct $\mathcal{D}$. This time, we set $\mathcal{P}$ to additionally contain $\mathfrak{R}$, i.e., $\mathcal{P} = \{+, x_1, x_2, \ldots, x_d, \mathfrak{R}\}$. This means that the function space $\mathcal{F}$ now contains functions of the form $f(\mathbf{x}) = c + \sum_{j=1}^{d} x_j m_j$ with $m_j \in \mathbb{N}_0$ and $c \in \mathbb{R}$ (sampled from $\mathfrak{R}$). As to $\mathcal{D}$, we will now include two observations instead of a single one. The first observation is set as before, i.e., $x_{1,1} = w_1, x_{2,1} = w_2, \ldots, x_{k,1} = w_k$ ($d = k$) and $y_1 = t$. As to the second observation, we set $x_{1,2} = 0, x_{2,2} = 0, \ldots, x_{k,2} = 0$ and $y_2 = 0$, i.e., the value of all features and of the label are set to zero. Now, $\mathcal{L}(\mathbf{y}, f(\mathbf{x})) = 0 \iff f(\mathbf{x}_i) = y_i$ for *both* $i = 1, 2$. For $f(\mathbf{x}_2) = y_2 = 0$, since any f has the form $f(\mathbf{x}) = c + \sum_{j=1}^{d} x_j m_j$ and $x_{j,2} = 0$, for all j, then $f(\mathbf{x}_2) = c + \sum_{j=1}^{d} 0 \times m_j = c$. But $f(\mathbf{x}_2) = y_2 = 0 \iff c = 0$. In other words, we know that every f for which $c \neq 0$ is one for which SR-Dec outputs NO. Therefore, by construction, we can immediately ignore all of those functions, and consider only the subset of $\mathcal{F}$ that contains functions of the form $f(\mathbf{x}) = 0 + \sum_{j=1}^{d} x_j m_j = \sum_{j=1}^{d} x_j m_j$. For every one of such functions, the loss for the second observation is by construction zero and we can therefore ignore it. Consequently, we are now back to the same setting considered in Theorem 1, which concludes the proof.

□

5 Conclusion

Our main contribution here was to prove that symbolic regression (SR), i.e., the problem of discovering an accurate model of data in the form of a mathematical expression, is in fact NP-hard. In particular, we have provided formal definitions of what SR should entail, and showed how the decision version of the unbounded subset sum problem can be reduced to a decision version of the SR problem. Except for the general definition of SR we considered, we have additionally shown that SR remains NP-hard even when the set of primitives must contain distributions from which constants can be sampled.

We hope that this note inspires more works on lower and upper bounds of different SR variants.

References

117th US Congress. Algorithmic accountability act, 2022. URL `https://www.congress.gov/bill/117th-congress/house-bill/6580/`.

Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon C++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pp. 1562–1570, 2020.

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

Alison Cozad and Nikolaos V Sahinidis. A global MINLP approach to symbolic regression. *Mathematical Programming*, 170(1):97–119, 2018.

Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, pp. 183–187, 1985.

Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

Erik Derner, Jiří Kubalík, Nicola Ancona, and Robert Babuška. Constructing parsimonious analytic models for dynamic systems via symbolic regression. *Applied Soft Computing*, 94:106432, 2020.

Aniko Ekart and Sandor Z. Nemeth. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73, 2001.

European Commission. Artificial intelligence act, 2021. URL `https://artificialintelligenceact.eu/`.

Brian C Falkenhainer and Ryszard S Michalski. Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning*, 1(4):367–401, 1986.

Bengt Fornberg and Julia Zuev. The runge phenomenon and spatially variable shape parameters in rbf interpolation. *Computers & Mathematics with Applications*, 54(3):379–398, 2007.

Donald Gerwin. Information processing, data inferences, and scientific generalization. *Behavioral Science*, 19 (5):314–325, 1974.

Daniel Hein, Steffen Udluft, and Thomas A Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

Alberto Hernandez, Adarsh Balasubramanian, Fenglin Yuan, Simon AM Mason, and Tim Mueller. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *npj Computational Materials*, 5(1):1–11, 2019.

Joseph F Hicklin. *Application of the genetic algorithm to automatic program generation.* PhD thesis, University of Idaho, 1986.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *Advances in Neural Information Processing Systems*, 32, 2019.

Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399, 2019.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Lukas Kammerer, Gabriel Kronberger, Bogdan Burlacu, Stephan M. Winkler, Michael Kommenda, and Michael Affenzeller. *Symbolic Regression by Exhaustive Search: Reducing the Search Space Using Syntactical Constraints and Efficient Semantic Structure Deduplication*, pp. 79–99. Springer International Publishing, 2020.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. Introduction to NP-Completeness of knapsack problems. In *Knapsack Problems*, pp. 483–493. Springer, 2004.

John R Koza. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, volume 34. Stanford University, 1990.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

John R Koza et al. Hierarchical genetic algorithms operating on populations of computer programs. In *International Joint Conference on Artificial Intelligence*, volume 89, pp. 768–774, 1989.

Gabriel Kronberger, Michael Kommenda, Andreas Promberger, and Falk Nickel. Predicting friction system performance with symbolic regression and genetic programming with factor variables. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1278–1285, 2018.

William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason H Moore. Learning concise representations for regression by evolving networks of trees. In *International Conference on Learning Representations*, 2018.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. Contemporary symbolic regression methods and their relative performance. In *Advances in Neural Information Processing Systems — Datasets and Benchmarks Track*, 2021.

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

Pablo Lemos, Niall Jeffrey, Miles Cranmer, Shirley Ho, and Peter Battaglia. Rediscovering orbital mechanics with machine learning. *arXiv preprint arXiv:2202.02306*, 2022.

Haoran Li, Yang Weng, and Hanghang Tong. CoNSoLe: Convex neural symbolic learning. *arXiv preprint arXiv:2206.00257*, 2022.

Qiang Lu, Jun Ren, and Zhiguang Wang. Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational Intelligence and Neuroscience*, 2016.

Marcus Märtens and Dario Izzo. Symbolic regression for space applications: Differentiable cartesian genetic programming powered by multi-objective memetic algorithms. *arXiv preprint arXiv:2206.06213*, 2022.

Yoshitomo Matsubara, Naoya Chiba, Ryo Igarashi, Tatsunori Taniai, and Yoshitaka Ushiku. Rethinking symbolic regression datasets and benchmarks for scientific discovery. *arXiv preprint arXiv:2206.10540*, 2022.

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

Robert I McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3):365–396, 2010.

David J Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

Fabrício Olivetti de França. A greedy search tree heuristic for symbolic regression. *Information Sciences*, 442-443:18–32, 2018. ISSN 0020-0255.

Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

Clemens Otte. Safe and interpretable machine learning: A methodological review. *Computational Intelligence in Intelligent Data Analysis*, pp. 111–122, 2013.

Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

Brenden K Petersen, Mikel Landajuela Larma, Terrell N Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2020.

Ricardo Poli, William B Langdon, and Nicholas F McPhee. *A Field Guide to Genetic Programming*. Lulu Press, 2008.

Daniel Rivero, Enrique Fernandez-Blanco, and Alejandro Pazos. DoME: A deterministic technique for equation development and symbolic regression. *Expert Systems with Applications*, 198:116712, 2022.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324 (5923):81–85, 2009.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Martin Vastl, Jonáš Kulhánek, Jirí Kubalík, Erik Derner, and Robert Babuška. SymFormer: End-to-end symbolic regression using transformer-based architecture. *arXiv preprint arXiv:2205.15764*, 2022.

Sergiy Verstyuk and Michael R. Douglas. Machine learning the gravity equation for international trade. *Available at SSRN 4053795*, 2022.

Marco Virgolin, Andrea De Lorenzo, Eric Medvet, and Francesca Randone. Learning a formula of interpretability to learn interpretable formulas. In *International Conference on Parallel Problem Solving from Nature*, pp. 79–93. Springer, 2020a.

Marco Virgolin, Ziyuan Wang, Tanja Alderliesten, and Peter AN Bosman. Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction. *Journal of Medical Imaging*, 7(4):046501, 2020b.

Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter A N Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary Computation*, 29(2):211–237, 2021a.

Marco Virgolin, Andrea De Lorenzo, Francesca Randone, Eric Medvet, and Mattias Wahde. Model learning with personalized interpretability estimation (ML-PIE). In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1355–1364, 2021b.

Tony Worm and Kenneth Chiu. Prioritized grammar enumeration: Symbolic regression by dynamic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1021–1028, 2013.

Ivan Zelinka, Zuzana Oplatkova, and Lars Nolle. Analytic programming–symbolic regression by means of arbitrary evolutionary algorithms. *International Journal of Simulation: Systems, Science and Technology*, 6(9):44–56, 2005.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.