Symbolic Regression by Exhaustive Search – Reducing the Search Space using Syntactical Constraints and Efficient Semantic Structure Deduplication.

Lukas Kammerer and Gabriel Kronberger and Bogdan Burlacu and Stephan M. Winkler and Michael Kommenda and Michael Affenzeller

Abstract Symbolic regression is a powerful system identification technique in industrial scenarios where no prior knowledge on model structure is available. Such scenarios often require specific model properties such as interpretability, robustness, trustworthiness and plausibility, that are not easily achievable using standard approaches like genetic programming for symbolic regression. In this chapter we introduce a deterministic symbolic regression algorithm specifically designed to address these issues. The algorithm uses a context-free grammar to produce models that are parameterized by a non-linear least squares local optimization procedure. A finite enumeration of all possible models is guaranteed by structural restrictions as well as a caching mechanism for detecting semantically equivalent solutions. Enumeration order is established via heuristics designed to improve search efficiency. Empirical tests on a comprehensive benchmark suite show that our approach is competitive with genetic programming in many noiseless problems while maintaining desirable properties such as simple, reliable models and reproducibility.

Key words: symbolic regression, grammar enumeration, graph search

Lukas Kammerer[1,2,3] e-mail: `lukas.kammerer@fh-hagenberg.at` · Gabriel Kronberger[1,3] · Bogdan Burlacu[1,3] · Stephan M. Winkler[1,2] · Michael Kommenda[1,3] · Michael Affenzeller[1,2]

[1] Heuristic and Evolutionary Algorithms Laboratory (HEAL), University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria
[2] Department of Computer Science, Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria
[3] Josef Ressel Center for Symbolic Regression, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria

1 Introduction

Symbolic regression is a task that we can solve with genetic programming (GP) and a common example where GP is particularly effective in practical applications. Symbolic regression is a machine learning task whereby we try to find a mathematical model represented as a closed-form expression that captures dependencies of variables from a dataset. Genetic programming has been proven to be well-suited for this task especially when there is little knowledge about the data-generating process. Even when we have a good understanding of the underlying process, GP can identify counterintuitive or unexpected solutions.

1.1 Motivation

GP has some practical limitations when used for symbolic regression. One limitation is that—as a stochastic process—it might produce highly dissimilar solutions even for the same input data. This can be very helpful to produce new "creative" solutions. However, it is problematic when we try to integrate symbolic regression in carefully engineered solutions (e.g. for automatic control of production plants). In such situations we would hope that there is an optimal solution and the solution method guarantees to identify the optimum. Intuitively, if the data changes only slightly, we expect that the optimal regression solution also changes only slightly. If this is the case we know that the solution method is trustworthy (cf. [15, 31]) and we can rely on the fact that the solutions are optimal at least with respect to the objective function that we specified. Of course this is only wishful thinking because of three fundamental reasons: (1) the symbolic regression search space is huge and contains many different expressions which are algebraically equivalent, (2) GP has no guarantee to explore the whole search space with reasonable computational resources and (3) the "optimal solution" might not be expressible as a closed-form mathematical expressions using the given building blocks. Therefore, the goal is to find an approximately optimal solution.

1.2 Prior Work

Different methods have been developed with the aim to improve the reliability of symbolic regression. Currently, there are several off-the-shelf software solutions which use enhanced variants of GP and are noteworthy in this context: the Data-Modeler package[1] [16] provides extensive capabilities for symbolic regression on top of Mathematica™. Eureqa™ is a commercial software tool[2] for symbolic re-

---

[1] `http://www.evolved-analytics.com/`

[2] `https://www.nutonian.com/products/eureqa/`

gression based on research described in [27, 28, 29]. The open-source framework HeuristicLab[3] [36] is a general software environment for heuristic and evolutionary algorithms with extensive functionality for symbolic regression and white-box modeling.

In other prior work, several researchers have presented non-evolutionary solution methods for symbolic regression. Fast function extraction (FFX) [22] is a deterministic method that uses elastic-net regression [39] to produce symbolic regression solutions orders of magnitudes faster than GP for many real-world problems. The work by Korns toward "extremely accurate" symbolic regression [12, 13, 14] highlights the issue that baseline GP does not guarantee to find the optimal solution even for rather limited search spaces. They give a useful systematic definition of increasingly larger symbolic regression search spaces using abstract expression grammars [10] and describes enhancements to GP to improve it's reliability. The work by Worm and Chiu on prioritized grammar enumeration [38] is closely related. They use a restricted set of grammar rules for deriving increasingly complex expressions and describe a deterministic search algorithm, which enumerates the search space for limited symbolic regression problems.

1.3 Organization of this Chapter

Our contribution is conceptually an extension of prioritized grammar enumeration [38], although our implementation of the method deviates significantly. The most relevant extensions are that we cut out large parts of the search space and provide a general framework for integrating heuristics in order to improve the search efficiency. Section 2 describes how we reduce the size of the search space which is defined by a context-free grammar:

1. We restrict the structure of solution to prevent too complicated solutions.
2. We use grammar restrictions to prevent semantic duplicates—solutions with different syntax but same semantics, such as algebraic transformations. With these restrictions, most solutions can only be generated in exactly one way.
3. We efficiently identify remaining duplicates with semantic hashing, so that (nearly) all solutions in the search space are semantically unique.

In Section 3, we explain the algorithm that iterates all these semantically unique solutions. The algorithm sequentially generates solutions from the grammar and keeps track of the most accurate one. For very small problems, it is even feasible to iterate the whole search space [19]. However, our goal in larger problems is to find accurate and concise solutions early during the search and to stop the algorithm after a reasonable time. The search order is determined with heuristics, which estimate the quality of solutions and prioritize promising ones in the search. A simple heuristic is proposed in Section 4. Modeling results in Section 5 show that this first version of our algorithm can already solve several difficult noiseless benchmark problems.

---

[3] `https://dev.heuristiclab.com`

2 Definition of the Search Space

The search space of our deterministic symbolic regression algorithm is defined by a context-free grammar. Production rules in the grammar define the mathematical expressions that can be explored by the algorithm. The grammar only specifies possible model structures whereby placeholders are used for numeric coefficients. These are optimized separately by a curve-fitting algorithm (e.g. optimizing least squares with an gradient-based optimization algorithm) using the available training data.

In a general grammar for mathematical expressions—as it is common in symbolic regression with GP for example—the same formula can be derived in several forms. These duplicates inflate the search space. To reduce them, our grammar is deliberately restricted regarding the possible structure of expressions. Remaining duplicates that cannot be prevented by a context-free grammar are eliminated via a hashing algorithm. Using both this grammar and hashing, we can generate a search space with only semantically unique expressions.

2.1 Grammar for Mathematical Expressions

In this work we consider mathematical expressions as list of symbols which we call *phrases* or *sentences*. A phrase can contain both *terminal* and *non-terminal* symbols and a sentence only terminal symbols. Non-terminal symbols can be replaced by other symbols as defined by a grammar's *production rules* while terminal symbols represent parts of the final expression like functions or variables in our case.

Our grammar is very similar to the one by Kronberger et al. [19]. It produces only rational polynomials which may contain linear and nonlinear terms, as outlined conceptually in Equation 1. The basic building blocks of terms are linear and nonlinear functions $\{+, \times, \text{inv}, \exp, \log, \sin, \text{square root}, \text{cube root}\}$. Recursion in the production rules represents a strategy for generating increasingly complex solutions by repeated nesting of expressions and terms.

$$
\begin{aligned}
Expr &= c_1 Term_1 + c_2 Term_2 + \ldots + c_n \\
Term &= Factor_0 \times Factor_1 \times \ldots \\
Factor &\in \{variable, \log(variable), \exp(variable), \sin(variable)\}
\end{aligned}
\tag{1}
$$

We explicitly disallow nested non-linear functions, as we consider such solutions too complex for real-world applications. Otherwise, we allow as many different structures as possible to keep accurate and concise models in the search space. We prevent semantic duplicates by generating just one side of mathematical equality relations in our grammar, e.g. we allow $xy + xz$ but not $x(y + z)$. Since each function has different mathematical identities, many different production rules are necessary to cover all special cases. Because we scale every term including function argu-

ments, we also end up with many placeholders for coefficients in the structures. All production rules are detailed in Listing 1 and described in the following.

Listing 1 Context-free grammar for generating mathematical expressions

```
G(Expr):
// Expressions and terms for polynomial structure
    Expr        -> "const" "*" Term "+" Expr    |
                   "const" "*" Term "+" "const"

    Term        -> RecurringFactors "*" Term    |
                   RecurringFactors             |
                   OneTimeFactors

    RecurringFactors -> VarFactor | LogFactor |
                        ExpFactor | SinFactor

    VarFactor  -> <variable>
    LogFactor  -> "log" "(" SimpleExpr ")"
    ExpFactor  -> "exp" "(" "const" "*" SimpleTerm ")"
    SinFactor  -> "sin" "(" SimpleExpr ")"

// Factors which can occur at most once per term
    OneTimeFactors -> InvFactor "*" SqrtFactor "*" CbrtFactor |
                      InvFactor "*" SqrtFactor               |
                      InvFactor "*"              CbrtFactor |
                                 SqrtFactor "*" CbrtFactor |
                      InvFactor                             |
                                 SqrtFactor                 |
                                            CbrtFactor

    InvFactor  -> "1/" "(" InvExpr ")"
    SqrtFactor -> "sqrt" "(" SimpleExpr ")"
    CbrtFactor -> "cbrt" "(" SimpleExpr ")"

// Function arguments
    SimpleExpr -> "const" "*" SimpleTerm "+" SimpleExpr  |
                  "const" "*" SimpleTerm "+" "const"

    SimpleTerm -> VarFactor "*" SimpleTerm | VarFactor

    InvExpr -> "const" "*" InvTerm "+" InvExpr |
               "const" "*" InvTerm "+" "const"

    InvTerm -> RecurringFactors "*" InvTerm  |
               RecurringFactors "*" SqrtFactor "*" CbrtFactor |
               RecurringFactors "*" SqrtFactor               |
               RecurringFactors "*"              CbrtFactor |
                                SqrtFactor "*" CbrtFactor |
               RecurringFactors                             |
                                SqrtFactor                 |
                                           CbrtFactor
```

We use a polynomial structure as outlined in Equation 1 to prevent a factored form of solutions. The polynomial structure is enforced with the production rules

`Expr` and `Term`. We restrict the occurrence of the multiplicative inverse ($= \frac{1}{\cdots}$), the square root and cube root function to prevent a factored form such as $\frac{1}{x+y}\frac{1}{x+z}$. This is necessary since we want to allow sums of simple terms as function arguments (see non-terminal symbol `SimpleExpr`). Therefore, these three functions can occur at most once time per term. This is defined with symbol `OneTimeFactors` and one production rule for each combination. The only function in which we do not allow sums as argument is exponentiation (see `ExpFactor`), since this form is substituted by the overall polynomial structure (e.g. we allow $e^x e^y$ but not e^{x+y}). Equation 2 shows some example identities and which forms are supported.

$$
\begin{array}{ccc}
\text{in the search space:} & & \text{not in the search space:} \\[4pt]
c_1 xy + c_2 xz + c_3 & \equiv & x(c_4 y + c_5 z) + c_6 \\[6pt]
c_1 \dfrac{1}{c_2 x + c_3 xx + c_4 xy + c_5 y + c_6} + c_7 & \equiv & c_8 \dfrac{1}{c_9 x + c_{10}}\; \dfrac{1}{c_{11} x + c_{12} y + c_{13}} + c_{14} \\[6pt]
c_1 \exp(c_2 x)\exp(c_3 y) + c_4 & \equiv & c_5 \exp(c_6 x + c_7 y) + c_8
\end{array}
\tag{2}
$$

We only allow (sums of) terms of variables as function arguments, which we express with the production rules `SimpleExpr` and `SimpleTerm`. An exception is the multiplicative inverse, in which we want to include the same structures as in ordinary terms. However, we disallow compound fractions like in Equation 3. Again, we introduce separate grammar rules `InvExpr` and `InvTerm` which cover the same rules as `Term` except the multiplicative inverse.

$$
\begin{array}{ccc}
\text{in the search space:} & & \text{not in the search space:} \\[4pt]
c_1 \dfrac{1}{c_2 \log(c_3 x + c_4) + c_5} + c_6 & \equiv & c_7 \dfrac{1}{c_8 \frac{1}{c_9 \log(c_{10} x + c_{11}) + c_{12}} + c_{13}} + c_{14}
\end{array}
\tag{3}
$$

In the simplest case, the grammar produces an expression $E_0 = c_0 x + c_1$, where x is a variable and c_0 and c_1 are coefficients corresponding to the slope and intercept. This expression is obtained by considering the simplest possible `Term` which corresponds to the derivation chain `Expr` $\rightarrow$ `Term` $\rightarrow$ `RecurringFactors` $\rightarrow$ `VarFactor` $\rightarrow$ x. Further derivations could lead for example to the expression $E_1 = c_0 x + (c_1 x + c_2)$, produced by nesting E_0 into the first part of the production rule for `Expr`, where the `Term` is again substituted with the variable x.

However, duplicate derivations can still occur due to algebraic properties like associativity and commutativity. These issues cannot be prevented with a context-free grammar because a context-free grammar does not consider surrounding symbols of the derived non-terminal symbol in its production rules. For example the expression $E_1 = c_0 x + (c_1 x + c_2)$ contains two coefficients c_0 and c_1 for variable x which could be folded into a new coefficient $c_{new} = c_0 + c_1$. This type of redundancy becomes even more pronounced when `VarFactor` has multiple productions (corresponding to multiple input variables), as it becomes possible for multiple derivation paths to

produce different expressions which are algebraically equivalent, such as $c_1 x + c_2 y$, $c_3 x + c_4 x + c_5 y$, $c_6 y + c_7 x$ for corresponding values of $c_1 \ldots c_7$. Another example are $c_1 xy$ and $c_2 yx$ which are both equivalent but derivable from the grammar.

To avoid re-visiting already explored regions of the search space, we implement a caching strategy based on expression hashing for detecting algebraically equivalent expressions. The computed hash values are the same for algebraically equivalent expressions. In the search algorithm we keep the hash values of all visited expressions and prevent re-evaluations of expressions with identical hash values.

2.2 Expression Hashing

We employ expression hashing by Burlacu et al. [3] to assign hash values to subexpressions within phrases and sentences. Hash values for parent expressions are aggregated in a bottom-up manner from the hash values of their children using any general-purpose hash function. We then simplify such expressions according to arithmetic properties such as commutativity, associativity, and applicable mathematical identities. The resulting canonical minimal form and associated hash value are then cached in order to prevent duplicated search effort.
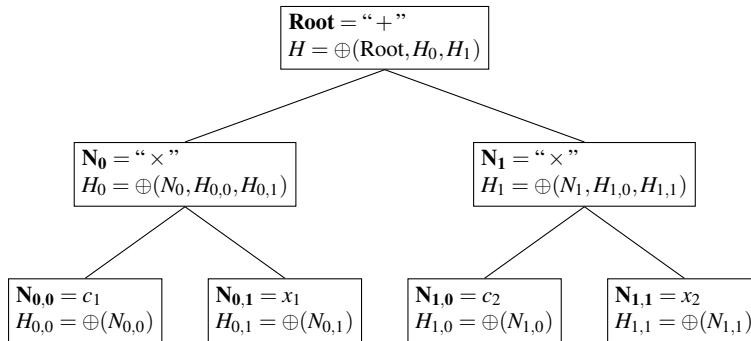

Fig. 1 Hash tree example, in which the hash values of all nodes are calculated from both their own node content and the has value of their children [3].

Expression hashing builds on the idea of Merkle trees [23]. Figure 1 shows how hash values propagate towards the tree root (the topmost symbol of the expression) using hash function $\oplus$ to aggregate child and parent hash values. Expression hashing considers an internal node's own symbol, as well as associativity and commutativity properties. To account for these properties, each hashing step must be accompanied by a corresponding sorting step, where child subexpressions are reordered according to their type and hash value. Algorithm 1 ensures that child nodes are sorted and hashed before parent nodes, such that calculated hash values are consistent towards the root symbol.

Algorithm 1: Expression hashing [3]

---

input : An expression E
output: The corresponding sequence of hash values

1 hashes $\leftarrow$ empty list of hash values;
2 symbols $\leftarrow$ list of symbols in E;
3 **foreach** *symbol s in symbols* **do**
4 $\quad$ $H(s) \leftarrow$ an initial hash value;
5 $\quad$ **if** *s is a terminal function symbol* **then**
6 $\quad\quad$ **if** *s is commutative* **then**
7 $\quad\quad\quad$ Sort the child nodes of s;
8 $\quad\quad$ child hashes $\leftarrow$ hash values of s's children;
9 $\quad\quad$ $H(n) \leftarrow \oplus(\text{child hashes}, H(s))$;
10 $\quad$ hashes.append($H(n)$);

11 **return** hashes;

An expression's hash value is then given by the hash value of its root symbol. After sorting, sub-expressions with the same hash value are considered isomorphic and are simplified according to arithmetic rules. The simplification procedure is illustrated in Figure 2 and consists of the following steps:

1. **Fold**: Apply associativity to eliminate nested symbols of the same type. For example, postfix expression `a b + c +` consists of two nested additions where each addition symbol has arity 2. Folding flattens this expression to the equivalent form `a b c +` where the addition symbol has arity 3.
2. **Simplify**: Apply arithmetic rules and mathematical identities to further simplify the expressions. Since expression already include placeholders for numerical coefficients, we eliminate redundant subexpressions such as `a a b +` which becomes `a b +`, or `a a +` which becomes `a`.
3. Repeat steps 1 and 2 until no further simplification is possible.

Nested $+$ and $\times$ symbols in Figure 2 are folded in the first step, simplifying the tree structure of the expression. Arithmetic rules are then applied for further simplification. In this example, the product of exponentials

$$\exp(c_1 \times x_1) \times \exp(c_2 \times x_1) \equiv \exp((c_1 + c_2) \times x_1)$$

is simplified since from a local optimization perspective, optimizing the coefficients of the expression yields better results for a single coefficient $c_3 = c_1 + c_2$, thus it makes no sense to keep both original factors. Finally, the sum $c_4 x_1 + c_5 x_1$ is also simplified since one term in the sum is redundant.

After simplification, the hash value of the simplified tree is returned as the hash value of the original expression. Based on this computation we are able to identify already explored search paths and avoid duplicated effort.
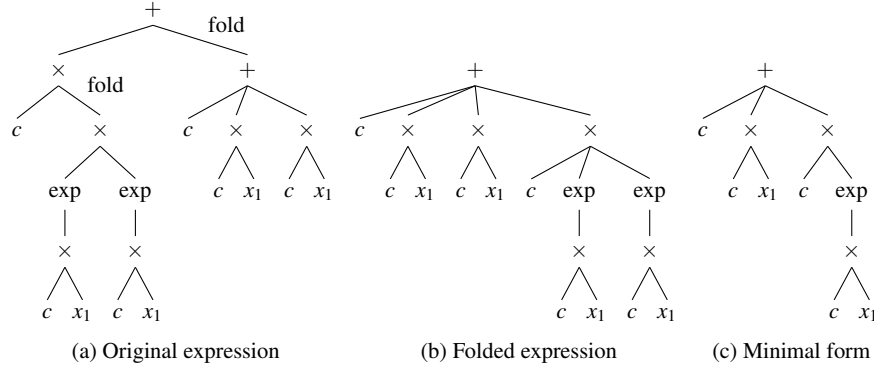
(a) Original expression   (b) Folded expression   (c) Minimal form

Fig. 2 Simplification to canonical minimal form during hashing

3 Exploring the Search Space

By limiting the size of expressions, the grammar and the hashing scheme produce a large but finite search space of semantically unique expressions. In an exhaustive search, we iterate all these expressions and search for the best fitting one. Thereby, we derive sentences with every possible derivation path. An expression is rejected if another expression with the same semantic—according to hashing—has already been generated during the search. When a new, previously unseen sentence is derived, the placeholders for coefficients are replaced with real values and optimized separately. The best fitting sentence is stored.

Algorithm 2 outlines how all unique expressions are derived: We store unfinished phrases—expressions with non-terminal symbols—in a data structure such as a stack or queue. We fetch phrases from this data structure one after another, derive new phrases, calculate their hash values and compare these hash values to previously seen ones. To derive new phrases, we always replace the *leftmost* non-terminal symbol in the old phrase with the production rules of this non-terminal symbol. If a derived phrase becomes a sentence with only terminal symbols, its coefficients are optimized and its fitness is evaluated. Otherwise, if it still contains derivable non-terminal symbols, it is put back on the data structure.

We restrict the length of a phrase by its number of variable references—e.g. xx and $\log(x) + x$ have two variable references. Phrases that exceed this limit are discarded in the search. Since every non-terminal symbol is eventually derived to at least one variable reference, non-terminal symbols count as variable references. In our experiments, a limit on the complexity has been found to be the most intuitive way to estimate an appropriate search space limit. Other measures, e.g. the number of symbols are harder to estimate since coefficients, function symbols and the non-factorized representation of expression quickly inflate the number of symbols in a phrase.

Algorithm 2: Iterating the Search Space

---

    input : Data set *ds*, max. number of variable references *maxVariableRefs*
    output: Best fitting expression

1   *openPhrases* ← empty data structure;
2   *seenHashes* ← empty set;
3   Add *StartSymbol* to *openPhrases*;
4   *bestExpression* ← constant symbol;

5   **while** *openPhrases* is not empty **do**
6   *oldPhrase* ← fetch and remove from *openPhrases*;
7   *nonTerminalSymbol* ← leftmost nonterminal symbol in *oldPhrase*;
8   **foreach** production *prod* of *nonTerminalSymbol* **do**
9   *newPhrase* ← apply *prod* on copy of *oldPhrase*;
10   **if** VariableRefs(*newPhrase*) ≤ *maxVariableRefs* **then**
11   *hash* ← Hash(*newPhrase*);
12   **if** *seenHashes* not contains *hash* **then**
13   Add *hash* to *seenHashes*;
14   **if** *newPhrase* is sentence **then**
15   Fit coefficients of *newPhrase* to *ds*;
16   Evaluate *newPhrase* on *ds*;
17   **if** *newPhrase* is better than *bestExpression* **then**
18   *bestExpression* ← *newPhrase*;

19   **else**
20   Add *newPhrase* to *openPhrases*;

21 **return** *bestExpression*

3.1 Symbolic Regression as Graph Search Problem

Without considering the semantics of an expression, we would end up exploring a search tree like in Figure 3, in which semantically equivalent expressions are derived multiple times (e.g. $c_1x + c_2x$ and $c_1x + c_2x + c_3x$). However, hashing turns the search tree into a directed search graph in which nodes (derived phrases) are reachable via one or more paths, as shown in Figure 4. Thus, hashing prevents the search in a graph region that was already visited. From this point of view, Algorithm 2 is very similar to simple graph search algorithms such as depth-first or breadth-first search.

3.2 Guiding the Search

In Algorithm 2, the order in which expressions are generated is determined by the data structure used. A stack or a queue would result in a depth-first or a breadth-first search respectively. However, as the goal is to find well-fitting expressions quickly and efficiently, we need to guide the traversal of a search graph towards promising phrases.
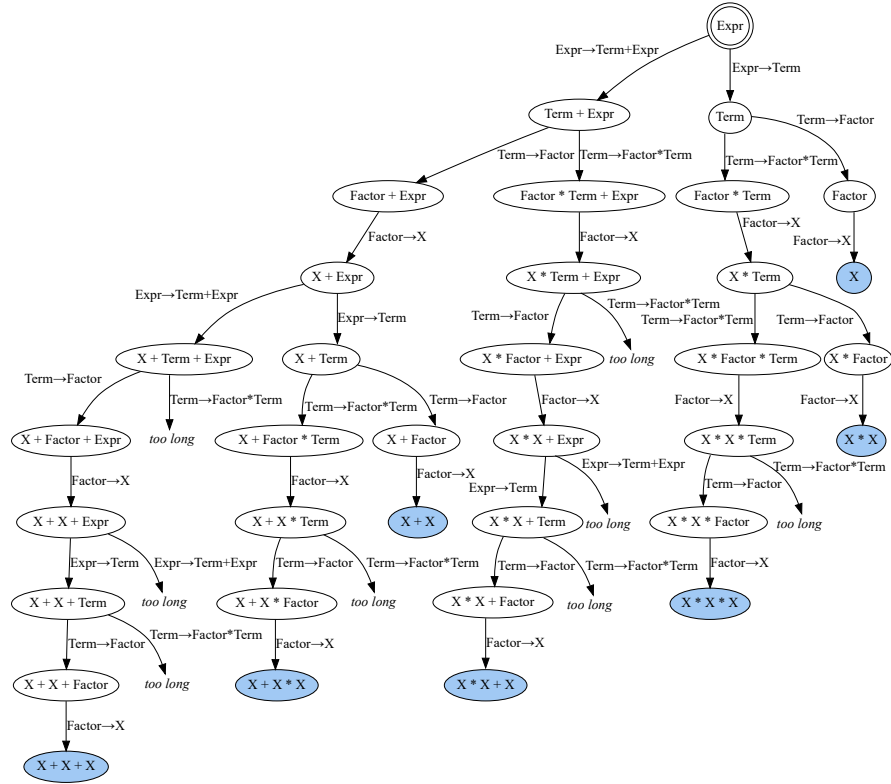
Fig. 3 Search tree of expression generation without semantic hashing.

Our general framework for guiding the search is very similar to the idea used in the A* algorithm [5]. We use a priority queue as data structure and assign a priority value to each phrase, indicating the expected quality of sentences which are derivable from that phrase. Phrases with high priority are derived first in order to discover well-fitting sentences, steering the algorithm towards good solutions.

Similar to the A* algorithm, we cannot make a definite statement about a phrase's priority before actually deriving all possible sentences from it. Therefore, we need to estimate this value with problem-specific heuristics. The calculation of phrase priorities provides us a generic point for integrating heuristics for improving the search efficiency and extending the algorithm's capabilities in future work.

4 Steering the Search

We introduce a simple heuristic for guiding the search and leave more complex and efficient heuristics for future work. The proposed heuristic makes a pessimistic
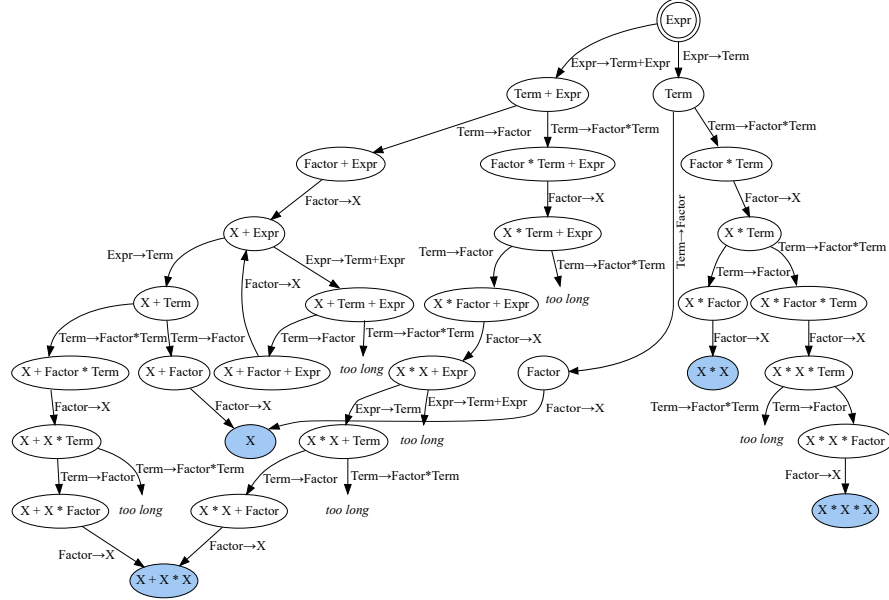
Fig. 4 Search graph with loops caused by semantic hashing.

estimation of the quality of a phrase's derivable sentences. This is done by evaluating phrases before they are derived to sentences. With the goal of finding short and accurate sentences quickly, the priority value considers both the expected quality and the length of a phrase.

4.1 Quality Estimation

Estimating the expected quality of an unfinished phrase is possible due to the polynomial structure of sentences and the derivation of the leftmost non-terminal symbol in every phrase. Since expressions are sums of terms ($c_1 Term_1 + c_2 Term_2 + ...$), repeated expansion of the leftmost non-terminal symbol derives one term after another. This results in phrases such as in Equation 4, in which the first two terms $c_1 \log(c_2 x + c_3)$ and $c_4 xx$ contain only terminal symbols and the last non-terminal symbol is *Expr*.

$$\underbrace{c_1 \log(c_2 x + c_3)}_{\textit{finishedTerm}_1} \quad + \quad \underbrace{c_4 xx}_{\textit{finishedTerm}_2} \quad + \quad \underbrace{Expr}_{\text{Treat as coefficient}} \tag{4}$$

Phrases where the only non-terminal symbol is `Expr` are evaluated as if they were full sentences by treating `Expr` as a coefficient during the local optimization phase. We get a pessimistic estimate of the quality of derivable sentences, since de-

rived sentences with more terms can only have better quality. The quality can only improve with more terms because of separate coefficient optimization and one scaling coefficient per term, as shown in Equation 5. If a term which does not improve the quality is derived, the optimization of coefficients will cancel it out by setting the corresponding scaling coefficient to zero (e.g. c_5 in Equation 5).

$$\textit{finishedTerm}_1 \quad + \quad \textit{finishedTerm}_2 \quad + \quad \underbrace{c_5\textit{Term}}_{\text{Can only improve quality}} \tag{5}$$

This heuristic works only for phrases in which *Expr* is the only non-terminal symbol. For sentences with different non-terminal symbols, we reuse the estimated quality from the last evaluated parent phrase. The estimate is updated when a new term with only terminal symbols is derived and again only one *Expr* remains. For now, we do not have a reliable estimation method for terms that contain non-terminal symbols and leave this topic for future work.

4.2 Priority Calculation

To prevent arbitrary adding of badly-fitting terms that are eventually scaled down to zero, our priority measure considers both a phrase's length and its expected accuracy. To balance these two factors, these two measures need to be in the same scale. We use the normalized mean squared error (NMSE) as quality measure which is in the range $[0, 1]$ for properly scaled solutions. This measure corresponds to $1 - R^2$ (coefficient of determination). As length measure we use the number of symbols relative to the maximum sentence length.

Since we limit the search space to a maximum number of variable references of a phrase, we cannot exactly calculate the maximum possible length of a phrase. Therefore, we estimate this maximum length with a greedy procedure: Starting with the grammar's start symbol *Expr*, we iteratively derive a new phrase using the longest production rule. If two production rules have the same length, we take the one with least non-terminal symbols and variable references.

Phrases with *lower* priority values are expanded first during the search. The priority for steering the search from Section 3 is the phrase's NMSE value minus its weighted relative length, as shown in Equation 6. The weight *w* controls the greediness and allows corrections of over- or underestimations of the maximum length. However, in practice this value is not critical.

$$\text{priority}(p) \ = \ \text{NMSE}(p) \ - \ w\frac{\text{len}(p)}{\textit{length}_{\text{max}}} \tag{6}$$

Table 1 OSGP experiment settings

Parameter	Setting
Population size	500
Max. selection pressure	300
Max. evaluated solutions	200 000
Mutation probability	15%
Selection	Gender-specific selection (random and proportional)
Crossover operator	Subtree swapping
Mutation operator	Point mutation, tree shaking, changing single symbols, replacing/removing branches
Max. tree size	Number of nodes: 30, depth: 50
Function set	$+, -, \times, \div$, exp, log, sin, cos, square, sqrt, cbrt

5 Experiments

We run our algorithm on several synthetic benchmark datasets to show that the search space defined by our restricted grammar is powerful enough to solve many problems in feasible time. As benchmark datasets, we use noiseless datasets from physical domains [4] and Nguyen-, Vladislavleva- and Keijzer-datasets [37] as defined and implemented in the HeuristicLab framework.

The search space was restricted in the experiments to include only sentences with at most 20 variable references. We evaluate at most 200 000 sentences. Coefficients are randomly initialized and then fitted with the iterative gradient-based Levenberg-Marquardt algorithm [20, 21] with at most 100 iterations. For each model structure, we repeat the coefficient fitting process ten times with differently initialized values to reduce the chance of finding bad local optima.

As a baseline, we also run symbolic regression with GP on the same benchmark problems. Therefore, we execute GP with strict offspring selection (OSGP) [1] and explicit optimization of coefficients [9]. The OSGP settings are listed in Table 1. The OSGP experiments were executed with the *HeuristicLab* software framework[4] [36]. Since this comparison focuses only on particular weaknesses and strengths of our proposed algorithm over state of the art-techniques, we use the same OSGP settings for all experiments and leave out problem-specific hyper parameter-tuning.

5.1 Results

Both the exhaustive search and OSGP were repeated ten times on each dataset. All repetitions of the exhaustive search algorithm led to the exact same results. This underlines the determinism of the proposed methods, even though we rely on stochasticity when optimizing coefficients. Also the OSGP results do not differ much. Ta-

---

[4] https://dev.heuristiclab.com

Table 2 Median NMSE results for Keijzer instances.

	Problem	Exhaustive Search		OSGP	
		Train.	Test	Train.	Test
1 [6, 7]	$0.3x\sin(2\pi x); x \in [-1, 1]$	3e-27	**2e-27**	1e-30	**8e-31**
2 [6]	$0.3x\sin(2\pi x); x \in [-2, 2]$	5e-22	**5e-22**	5e-18	**4e-18**
3 [6]	$0.3x\sin(2\pi x); x \in [-3, 3]$	6e-32	**3e-31**	4e-30	**3e-30**
4 [6, 26]	$x^3\exp(-x)\cos(x)\sin(x)(\sin(x)^2\cos(x) - 1)$	1e-04	2e-04	1e-06	1e-06
5 [6]	$(30xz)/((x - 10)y^2)$	3e-08	3e-08	3e-20	**3e-20**
6 [6, 32]	$\sum_{i=1}^{x} \frac{1}{i}$	8e-13	**6e-09**	5e-14	**5e-13**
7 [6, 32]	$\ln(x)$	2e-31	**3e-31**	1e-30	**2e-30**
8 [6, 32]	$\sqrt{x}$	2e-14	**8e-10**	5e-21	**1e-21**
9 [6, 32]	$\operatorname{arcsinh}(x)$ i.e. $\ln(x + \sqrt{x^2 + 1})$	5e-14	1e-05	5e-17	**6e-16**
10 [6, 32]	x^y	4e-04	1e-01	6e-32	2e-04
11 [6, 33]	$xy + \sin((x - 1)(y - 1))$	7e-04	7e-01	2e-22	9e-02
12 [6, 33]	$x^4 - x^3 + y^2/2 - y$	5e-32	**1e-31**	7e-22	**8e-18**
13 [6, 33]	$6\sin(x)\cos(y)$	2e-32	**2e-31**	3e-32	**3e-32**
14 [6, 33]	$8/(2 + x^2 + y^2)$	4e-32	**2e-31**	1e-17	**1e-17**
15 [6, 33]	$x^3/5 + y^3/2 - y - x$	1e-22	**2e-21**	2e-11	**6e-10**

bles 2-5 show the achieved NMSE values for the exhaustive search and the median NMSE values of all OSGP repetitions. NMSE values in the Tables 2-5 smaller than 10^{-8} are considered as exact or good-enough approximations and emphasized in bold. The exhaustive search found a good solution (NMSE $< 10^{-8}$) within ten minutes for all datasets. If no such solution was found, the algorithm runs until it reaches the max. number of evaluated solutions, which can take days for larger datasets.

The experimental results show, that our algorithm struggles with problems with complex terms—for example with Keijzer data sets 4, 5 and 11 in Table 2. This is probably because our heuristic works "term-wise"—our algorithm searches completely broad without any guidance within terms which still contain non-terminal symbols. This issue becomes even more pronounced when we have to find long and complex function arguments. It should also be noted that our algorithm only finds non-factorized representations of such arguments, which are even longer and therefore even harder to find in a broad search.

For the Nguyen datasets in Table 3 and the Keijzer datasets 12-15 in Table 2, we find the exact or good approximations in most cases with our exhaustive search. Especially for simpler datasets, the results of our algorithm surpasses the one of OSGP. This is likely due to the datasets' low number of training instances, which makes it harder for OSGP to find good approximations.

Some problems are not contained in the search space, thus we do not find any good solution for them. This is the case for Keijzer 6, 9 and 10 in Table 2, for which we do not support the required function symbols in our grammar. Also all Vladislavleva datasets except 6 and 7 in Table 4 and the problems "Fluid Flow" and "Pagie-1" in Table 5 are not in the hypothesis space as they are too complex.

Another issue is the optimization of coefficients. Although several problems have a simple structure and are in the search space, we do not find the right coefficients

for arguments of non-linear functions, for example in Nguyen 5-7. The issue hereby is that we iterate over the actually searched model structure but determine bad co-efficients. As we do never look again at the same model structure, we can only find an approximation. This is a big difference to symbolic regression with genetic programming, in which we might find the same structure again in next generations.

6 Discussion

Among the nonlinear system identification techniques, symbolic regression is characterized by its ability to identify complex nonlinear relationships in structured numerical data in the form of interpretable models. The combination of the power of nonlinear system identification without a priori assumptions about the model structure with the white-box ability of mathematical formulas represents the unique selling point of symbolic regression. If tree-based GP is used as search method, the ability to interpret the found models is limited due to the stochasticity of the GP search. Thus, at the end of the modeling phase, several similarly complex models of approximately the same quality can be produced, which have completely different structures and use completely different subsets of features. These last-mentioned limitations due to ambiguity can be countered using a deterministic approach in which only semantically unique models may be used. This approach, however, re-quires a lot of restrictions regarding search space complexity in order to specify a subspace in which an exhaustive search is feasible. On the other hand, the exhaus-tive claim enables the approach to generate extensive model libraries already in the offline phase, through which as soon as a concrete task is given in the online phase, it is only necessary to navigate in a suitable way.

Table 3 Median NMSE results for Nguyen instances.

Problem		Exhaustive Search		OSGP	
		Train.	Test	Train.	Test
1 [34]	$x^3 + x^2 + x$	5e-34	**3e-33**	8e-30	**2e-29**
2 [34]	$x^4 + x^3 + x^2 + x$	3e-33	**4e-33**	5e-30	**1e-28**
3 [34]	$x^5 + x^4 + x^3 + x^2 + x$	1e-33	**7e-33**	2e-16	**2e-15**
4 [34]	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	6e-12	**6e-11**	2e-12	3e-08
5 [34]	$\sin(x^2)\cos(x) - 1$	9e-14	**3e-13**	3e-18	**4e-18**
6 [34]	$\sin(x) + \sin(x + x^2)$	2e-17	**2e-12**	6e-14	6e-08
7 [34]	$\log(x+1) + \log(x^2+1)$	4e-13	**5e-12**	5e-13	**1e-09**
8 [34]	$\sqrt{x}$	6e-32	**2e-31**	7e-32	**1e-31**
9 [34]	$\sin(x) + \sin(y^2)$	2e-13	**2e-12**	8e-31	**8e-31**
10 [34]	$2\sin(x)\cos(y)$	5e-32	**1e-31**	1e-28	**8e-29**
11 [34]	x^y	2e-06	1e-02	6e-30	**3e-30**
12 [34]	$x^4 - x^3 + y^2/2 - y$	2e-31	**2e-31**	7e-18	**5e-17**

Table 4 Median NMSE results for Vladislavleva instances.

Problem		Exhaustive Search		OSGP	
		Train.	Test	Train.	Test
1 [30]	$\exp(-(x_1-1)^2)/(1.2+(x_2-2.5)^2)$	3e-03	3e-01	1e-09	9e-07
2 [26]	$\exp(-x)x^3\cos(x)\sin(x)(\cos(x)\sin(x)^2-1)$	3e-04	1e-02	3e-06	2e-03
3 [35]	$f_2(x_1)(x_2-5)$	1e-02	2e-01	3e-05	6e-04
4 [35]	$10/(5+\sum_{i=1}^{5}(x_i-3)^2)$	1e-01	2e-01	7e-03	1e-02
5 [35]	$30((x_1-1)(x_3-1))/(x_2^2(x_1-10))$	2e-03	9e-03	8e-16	**9e-15**
6 [35]	$6\sin(x_1)\cos(x_2)$	8e-32	**4e-31**	6e-31	**3e-19**
7 [35]	$(x_1-3)(x_2-3)+2\sin((x_1-4)(x_2-4))$	1e-30	**9e-31**	5e-29	**4e-29**
8 [35]	$((x_1-3)^4+(x_2-3)^3-(x_2-3))/((x_2-2)^4+10)$	1e-03	2e-01	5e-05	2e-02

Table 5 Median NMSE results for other instances.

Problem	Exhaustive Search		OSGP	
	Train.	Test	Train.	Test
Poly-10 [25] $x_1x_2+x_3x_4+x_5x_6+x_1x_7x_9+x_3x_6x_{10}$	2e-32	**1e-32**	7e-02	1e-01
Pagie-1 (Inverse Dynamics) [24] $1/(1+x^{-4})+1/(1+y^{-4})$	1e-03	6e-01	9e-07	5e-05
Aircraft Lift Coefficient [4] $C_{L\alpha}(\alpha-\alpha_0)+C_{L\delta_e}\delta_e S_{HT}/S_{ref}$	3e-31	**3e-31**	2e-17	**2e-17**
Fluid Flow [4] $V_\infty r\sin(\theta)(1-R^2/r^2)+\Gamma/(2\pi)\ln(r/R)$	3e-04	4e-04	9e-06	2e-05
Rocket Fuel Flow [4] $p_0 A^\star/\sqrt{T_0}\sqrt{\gamma/R(2/(\gamma+1))^{(\gamma+1)/(\gamma-1)}}$	3e-31	**3e-31**	1e-19	**1e-19**

In a very reduced summary, one could characterize the classical tree-based symbolic regression using GP and the approach of deterministically and exhaustively generating models in such a way that the latter enables a complete search in an incomplete search space while the classical approach performs an incomplete search in a rather complete search space.

6.1 Limitations

The approach we have described in this contribution also has several limitations. For the identification of optimal coefficient values we rely on the Levenberg-Marquardt method for least squares, which is a local search routine using gradient information. Therefore, we can only hope to find global optima for coefficient values. Finding bad local optima for coefficients is less of a concern when using GP variants with a similar local improvement scheme because there are implicitly many restarts through the evolutionary operations of recombination and mutation. In the proposed method

we visit each structure only once and therefore risk to discard a good solution when we are unlucky to find good coefficients.

We have worked only with noiseless problem instances yet. We observed in first experiments with noisy problems instances that the algorithm might get stuck trying to improve non-optimal partial solutions due to its greedy nature. Therefore, we need further investigations before we move on with the development of our algorithm to noisy real-world problems.

Another limitation is the poor scalability of grammar enumeration when increasing the number of features or the size of the search space. When increasing these parameters we can not expect to explore a significant part of the complete search space and must increasingly rely on the power of heuristics to hone in on relevant subspaces. Currently, we have only integrated a single heuristic which evaluates terms in partial solutions and prioritizes phrase which include well-fitting terms. However, the algorithm has no way to prioritize incomplete terms and is inefficient when trying to find complex terms.

7 Outlook

Even when considering the above mentioned limitations of the currently implemented algorithm we still see significant potential in the approach of more systematic and deterministic search for symbolic regression and we already have several ideas to improve the algorithm and overcome some of the limitations.

The integration of improved heuristics for guided search is our top-priority. An advantage of the concept is that it is extremely general and allows to experiment with many different heuristics. Heuristics can be as simple as prioritizing shorter expressions or less complex expressions. More elaborate schemes which guide the search based on prior knowledge about the data-generating process are easy to imagine. Heuristics could incorporate syntactical information (e.g. which variables already occur within the expression) as well as information from partial evaluation of expressions. We also consider dynamic heuristics which are adjusted while the algorithm is running and learning about the problem domain. Potentially, we could even identify and learn heuristics which are transferable to other problem instances and would improve efficiency in a transfer learning setting.

Getting trapped in local optima is less of a concern when we apply global search algorithms for coefficient values such as evolution strategies, differential evolution, or particle swarm optimization (cf. [11]). Another approach would be to reduce the ruggedness of the objective function through regularization of the coefficient optimization step. This could be helpful to reduce the potential of overfitting and getting stuck in sub-optimal subspaces of the search space.

Generally, we consider grammar enumeration to be effective only when we limit the search space to relatively short expressions—which is often the case in our industrial applications. Therein lies the main potential compared to the more general approach of genetic programming. In this context we continue to explore potential

for segmentation of the search space [19] in combination with grammar enumeration in an offline phase for improving later search runs. Grammar enumeration with deduplication of structures could also be helpful to build large offline libraries of sub-expressions that could be used by GP [2, 8, 17, 18].

References

1. Affenzeller, M., Winkler, S., Wagner, S., Beham, A.: Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications, *Numerical Insights*, vol. 6. CRC Press, Chapman & Hall (2009)
2. Angeline, P.J., Pollack, J.: Evolutionary module acquisition. In: Proceedings of the Second Annual Conference on Evolutionary Programming, pp. 154–163. La Jolla, CA, USA (1993)
3. Burlacu, B., Kammerer, L., Affenzeller, M., Kronberger, G.: Hash-based Tree Similarity and Simplification in Genetic Programming for Symbolic Regression. In: Computer Aided Systems Theory, EUROCAST 2019 (2019)
4. Chen, C., Luo, C., Jiang, Z.: A multilevel block building algorithm for fast modeling generalized separable systems. Expert Systems with Applications **109**, 25–34 (2018)
5. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
6. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Genetic Programming, Proceedings of EuroGP'2003, *LNCS*, vol. 2610, pp. 70–82. Springer-Verlag, Essex (2003)
7. Keijzer, M., Babovic, V.: Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In: Genetic Programming, Proceedings of EuroGP'2000, *LNCS*, vol. 1802, pp. 76–90. Springer-Verlag, Edinburgh (2000)
8. Keijzer, M., Ryan, C., Murphy, G., Cattolico, M.: Undirected training of run transferable libraries. In: Proceedings of the 8th European Conference on Genetic Programming, *Lecture Notes in Computer Science*, vol. 3447, pp. 361–370. Springer, Lausanne, Switzerland (2005)
9. Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M., Wagner, S.: Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '13 Companion, pp. 1121–1128. ACM (2013)
10. Korns, M.F.: Symbolic regression using abstract expression grammars. In: GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, pp. 859–862. ACM, Shanghai, China (2009)
11. Korns, M.F.: Abstract expression grammar symbolic regression. In: Genetic Programming Theory and Practice VIII, Genetic and Evolutionary Computation, vol. 8, chap. 7, pp. 109–128. Springer, Ann Arbor, USA (2010)
12. Korns, M.F.: Extreme accuracy in symbolic regression. In: Genetic Programming Theory and Practice XI, Genetic and Evolutionary Computation, chap. 1, pp. 1–30. Springer, Ann Arbor, USA (2013)
13. Korns, M.F.: Extremely accurate symbolic regression for large feature problems. In: Genetic Programming Theory and Practice XII, Genetic and Evolutionary Computation, pp. 109–131. Springer, Ann Arbor, USA (2014)

14. Korns, M.F.: Highly accurate symbolic regression with noisy training data. In: Genetic Programming Theory and Practice XIII, Genetic and Evolutionary Computation, pp. 91–115. Springer, Ann Arbor, USA (2015)
15. Kotanchek, M., Smits, G., Vladislavleva, E.: Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models. In: Genetic Programming Theory and Practice V, Genetic and Evolutionary Computation, chap. 12, pp. 201–220. Springer, Ann Arbor (2007)
16. Kotanchek, M.E., Vladislavleva, E., Smits, G.: Symbolic Regression Is Not Enough: It Takes a Village to Raise a Model, pp. 187–203. Springer New York, New York, NY (2013)
17. Krawiec, K., Pawlak, T.: Locally geometric semantic crossover. In: GECCO Companion '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion, pp. 1487–1488. ACM, Philadelphia, Pennsylvania, USA (2012)
18. Krawiec, K., Swan, J., O'Reilly, U.M.: Behavioral program synthesis: Insights and prospects. In: Genetic Programming Theory and Practice XIII, Genetic and Evolutionary Computation, pp. 169–183. Springer, Ann Arbor, USA (2015)
19. Kronberger, G., Kammerer, L., Burlacu, B., Winkler, S.M., Kommenda, M., Affenzeller, M.: Cluster analysis of a symbolic regression search space. In: Genetic Programming Theory and Practice XVI. Springer, Ann Arbor, USA (2018)
20. Levenberg, K.: A method for the solution of certain non-linear problems in least squares. Quarterly of Applied Mathematics **2**(2), 164–168 (1944)
21. Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics **11**(2), 431–441 (1963)
22. McConaghy, T.: FFX: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, Genetic and Evolutionary Computation, chap. 13, pp. 235–260. Springer, Ann Arbor, USA (2011)
23. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology — CRYPTO '87, pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
24. Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary Computation **5**(4), 401–418 (1997)
25. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In: Genetic Programming, Proceedings of EuroGP'2003, *LNCS*, vol. 2610, pp. 204–217. Springer-Verlag, Essex (2003)
26. Salustowicz, R.P., Schmidhuber, J.: Probabilistic incremental program evolution. Evolutionary Computation **5**(2), 123–141 (1997)
27. Schmidt, M., Lipson, H.: Co-evolving fitness predictors for accelerating and reducing evaluations. In: Genetic Programming Theory and Practice IV, Genetic and Evolutionary Computation, vol. 5, pp. 113–130. Springer, Ann Arbor (2006)
28. Schmidt, M., Lipson, H.: Symbolic regression of implicit equations. In: Genetic Programming Theory and Practice VII, Genetic and Evolutionary Computation, chap. 5, pp. 73–85. Springer, Ann Arbor (2009)
29. Schmidt, M., Lipson, H.: Age-fitness pareto optimization. In: Genetic Programming Theory and Practice VIII, Genetic and Evolutionary Computation, vol. 8, chap. 8, pp. 129–146. Springer, Ann Arbor, USA (2010)
30. Smits, G., Kotanchek, M.: Pareto-front exploitation in symbolic regression. In: Genetic Programming Theory and Practice II, chap. 17, pp. 283–299. Springer, Ann Arbor (2004)
31. Stijven, S., Vladislavleva, E., Kordon, A., Kotanchek, M.: Prime-time: Symbolic regression takes its place in industrial analysis. In: Genetic Programming Theory and Practice XIII, Genetic and Evolutionary Computation, pp. 241–260. Springer, Ann Arbor, USA (2015)
32. Streeter, M.J.: Automated discovery of numerical approximation formulae via genetic programming. Master's thesis, Computer Science, Worcester Polytechnic Institute, MA, USA (2001)
33. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pp. 155–162. Morgan Kaufmann, San Francisco, California, USA (2001)

34. Uy, N.Q., Hoai, N.X., O'Neill, M., McKay, R.I., Galvan-Lopez, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines **12**(2), 91–119 (2011)
35. Vladislavleva, E.J., Smits, G.F., den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. IEEE Transactions on Evolutionary Computation **13**(2), 333–349 (2009)
36. Wagner, S., Affenzeller, M.: HeuristicLab: A generic and extensible optimization environment. In: Adaptive and Natural Computing Algorithms, pp. 538–541. Springer (2005)
37. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., O'Reilly, U.M., Luke, S.: Better GP benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines **14**(1), 3–29 (2013)
38. Worm, T., Chiu, K.: Prioritized grammar enumeration: symbolic regression by dynamic programming. In: GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference, pp. 1021–1028. ACM, Amsterdam, The Netherlands (2013)
39. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal of the royal statistical society: series B (statistical methodology) **67**(2), 301–320 (2005)