On the Existence of Feature Bundles and their Effect on Symbolic Regression Algorithms

Kourosh Neshatian*, Lucianne Varn
*Department of Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
kourosh.neshatian@canterbury.ac.nz

***Abstract*—In this paper, we consider a special subset of the features in a regression learning problem that, while being relevant to the problem, its strict subsets do not show any sign of relevance. We discuss the presence of such subsets of features, which we call 'feature bundles', and examine the challenges they pose in feature selection and learning. We demonstrate the effect of these feature bundles on the performance of commonly-used symbolic regression algorithms.**

***Index Terms*—symbolic regression, feature selection, genetic programming**

I. Introduction

The regression problem is concerned with finding a computable model that describes the relationship between a group of input variables (or features) and an output variable. The model is often used for the prediction of the output variable on unseen data. Finding the right model often involves a search, or more specifically solving an optimization problem. The search space is a set (or collection) of functions, and the job of the learning algorithm is to find the right function from this set.

The representation of the search space varies depending on the algorithm. In linear regression, the search space is a set of weight (or coefficient) vectors, and the job of the learning algorithm is to find the optimal weight vector. A good algorithm for learning linear regression would exploit the properties of the search space (for example, its convexity) in order to find a solution efficiently.

Symbolic regression is another approach to regression where the search space is a set of expressions (that can be constructed from a set of variables and functions). Like other methods, the job of the symbolic regression learning algorithm is to find the right expression. However, because the search space in symbolic regression is complex (and many notions, such as distance and locality, are not defined), the learning algorithm has to resort to (meta-)heuristic techniques, such as evolutionary search. Prominent symbolic regression learning algorithms, in order of commonality, are *genetic programming* and *gene expression programming*.

Genetic programming (GP) is a hyper-heuristic search algorithm that evolves computer programs [1]–[3]. GP is capable of dynamically building logical and mathematical expressions [4], and classification models [5], [6]. GP is the most prominent tool used in symbolic regression [7]—it has been used in many application areas, ranging from industry [8], [9] to finance [10], [11].

Gene Expression Programming (GEP) is another evolutionary paradigm for symbolic regression [12]. GEP is population-based and the candidate solutions (or individuals) are represented as fixed-length chromosomes. As in GP, the components of the chromosomes are terminals and primitive functions, and like many other evolutionary search mechanisms, various operators (including the crossover and mutation operators) are applied to the members of the population in order to direct the search towards better candidate solutions [13].

A. Related work and research goals

There have been many efforts in improving GP-based symbolic regression learning. Various capabilities have been introduced, including generalization [14], parallelism [15], and autonomy [16].

Recent advances have also been made in using GP-based symbolic regression on high-dimensional problems. In [17], an embedded feature construction mechanism is introduced to dynamically augment the terminal set. In [18], the authors propose a multi-stage feature selection mechanism that can be used to solve high-dimensional symbolic regression problems.

There are also limitations or difficulties in using evolutionary symbolic regression learning. These include high computational cost, bloating, finding suitable values for search parameters (including various evolutionary operators, population size, maximum number of generations, limits on the size of individuals, etc.) [19], [20]. More specific to symbolic regression is the problem of finding appropriate numeric constants in an efficient way [12].

Another concern is the scalability of learning algorithms. In [21], the scalability of GP with respect to the number of examples is studied analytically and with respect to the number of features is studied empirically. The results show that while GP scales very well (linearly) with respect to the number of training examples, it scales poorly with respect to the number of features on some problems. In the experiments conducted in the paper, it was shown that even on some linearly separable classification problems, the amount of computation required to find a solution increases exponentially with the number of features.

In this paper, we focus on the presence of *feature bundles*— subsets of relevant features with a special property— in symbolic regression, and their effect on the performance of the learning algorithm. We will show that despite the power of some algorithms (such as GP) in automatically and implicitly selecting features [22], in the presence of large *feature bundles*, canonical symbolic regression algorithms (namely GP and GEP) cannot perform very well.

B. Organization

This paper is organized as follows. In Section II, we provide an overview of the problem and introduce the concept of *feature bundles*. In Section III, we describe the design of the experiments, including the simulation and search processes. In Section IV, we present the results of our simulation experiments and discuss the observed trends. We conclude the paper in Section V.

II. Feature Bundles and Their Effect on Regression Learning

A. Regression Problems

Regression learning involves finding a function $h^* : \mathcal{X} \to \mathcal{Y}$—which maps an input space $\mathcal{X}$ to an output space $\mathcal{Y}$—that minimizes some *risk* function R, i.e.

$$h^* = \arg\min_{h \in \mathcal{H}} R(h) \quad , \tag{1}$$

where $\mathcal{H}$ is the set of possible predictor functions or candidate solutions. The learning algorithm is tasked with finding the optimal function h^* in the set $\mathcal{H}$, which given an input vector $\mathbf{x} \in \mathcal{X}$, predicts the best possible output $y \in \mathcal{Y}$.

The nature of the elements of $\mathcal{H}$ depends on the regression method being used. In linear regression, the elements are weight vectors; each element specifies a line (or hyperplane) in $\mathcal{X} \times \mathcal{Y}$. In symbolic regression, the elements of $\mathcal{H}$ are expression trees made of primitive functions and terminals.

The risk function in (1) is defined as the expected *loss* over all points in the space $\mathcal{X} \times \mathcal{Y}$, where "loss" is some measure of the deviations of the predicted values from the actual values [23]. Let $l(h(\mathbf{X}), Y)$ denote the loss function for a given predictor function h, such that $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, where $\mathbf{X} \in \mathcal{X}$ denotes the vector of input variables, and $Y \in \mathcal{Y}$, the output variable. Here, $\mathbf{X}$ and Y are assumed to be random variables with some joint probability distribution $F_{\mathbf{X}Y}$. Then, for a given predictor function h, the risk or expected loss function is

$$R(h) = \mathbf{E}[l(h(\mathbf{X}), Y)] = \int_{\mathbf{x},y} l(h(\mathbf{x}), y)\, dF_{\mathbf{X}Y}(\mathbf{x}, y) \quad . \tag{2}$$

A commonly-used loss function in linear regression is the *squared error* loss function $l(f(\mathbf{X}), Y) = (f(\mathbf{X}) - Y)^2$, which has desirous properties such as being infinitely divisible and convex. For the squared error loss function, the predictor function or expression that yields the minimum risk is the conditional expected value of Y, given the input $\mathbf{X}$, i.e. $h^* = E[Y \mid \mathbf{X} = \mathbf{x}]$ [24].

To compute the risk, the joint distribution $F_{\mathbf{X}Y}$ must be known, which is rarely the case, and therefore, an approximation of the risk function, known as the *empirical risk* function, is minimized to find the optimal predictor function. The empirical risk function, denoted by R_m, is defined as the estimated expected loss:

$$R_m(h) = \frac{1}{m} \sum_{i=1}^{m} l(h(\mathbf{X_i}), Y_i) \quad , \tag{3}$$

computed over the training dataset, denoted by $\{(\mathbf{X_1}, Y_1), \ldots, (\mathbf{X_m}, Y_m)\}$, which is assumed to be an independent and identically distributed random sample from the probability distribution $F_{\mathbf{X}Y}$. The predictor function obtained by minimizing the empirical risk function is denote by h^*_m to emphasize its dependence on a training dataset of cardinality m. For a given predictor function $h \in \mathcal{H}$, the difference between the empirical risk $R_m(h)$ in (3) and the true risk $R(h)$ in (2) decreases as the cardinality m of the training dataset increases [25].

B. Partial Solutions in Symbolic Regression

All prominent symbolic regression algorithms are iterative search algorithms. The idea is that an initial, randomly-generated candidate solution (or a population of candidate solutions) is gradually improved during the search. Throughout the search different *operators* are applied to the existing candidate(s) in order to derive new (and hopefully better) candidates. In a genetic search (e.g. GP and GEP), common choices of these operators are mutation and crossover.

For most non-trivial symbolic regression problems, it is extremely unlikely for an algorithm to arrive at the solution purely by chance and without progressively going through some intermediate *partial* solutions. All symbolic regression leaning algorithms rely on gradually improving these partial solutions.

Definition 1. A function h (or an expression representing it) is a partial solution iff $R(h) < R(\mathrm{E}[Y])$.

A partial solution performs better than a constant model that is merely the expected value of the target variable Y, completely independent of any input features. Candidates that are not partial solutions receive low fitness and are not likely to survive during the execution of the learning algorithm. Partial solutions are important because they act as stepping stones towards better solutions. For a sequence of partial solutions to the learning problem, the corresponding risks are in decreasing order. That is, the risks associated with a sequence of partial solutions leading to the final and best solution $h^\star$ satisfy the following inequality:

$$R(h_1) > R(h_2) > \cdots > R(h_k) = R(h^\star) \quad ,$$

for $1 \leq k < \infty$.

A partial solution (or expression) that is not a perfect solution (i.e. any partial solution that is not $h^\star$) falls in one of the following two categories:

1) the partial solution contains all the features that must appear in the final solution, but the primitive functions (i.e. the internal nodes of the expression tree) are not of the correct type or in a correct hierarchy; or
2) the partial solution does not contain all the necessary features.

In this section, we will show that there are regression problems for which there is no partial solution in the second category—that is, any partial solution to the problem must include all the features that need to appear in the final solution.

We begin with an example where a partial solution can belong to any of the two categories, and then present an example where the partial solution can only belong to the first category, which leads us to the concept of a *feature bundle* (introduced in the next section).

Example 1. *A regression problem that allows partial solutions in both categories*

Consider a regression problem with n mutually independent and identically distributed input features $X_1, X_2, \ldots, X_n$, where $X_i \sim \mathcal{U}(-1,1)$, for all $i \in \{1,\ldots,n\}$, and the target function is $Y = \sum_{i=1}^{n} X_i$. Since noise and irrelevant features are absent in this example, the perfect solution is $h^*(\mathbf{x}) = \sum_{i=1}^{n} x_i$.

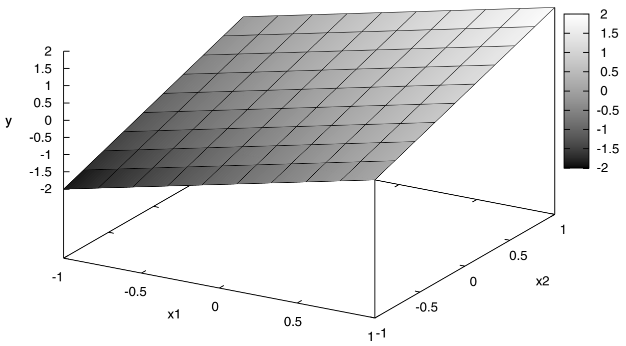
For the case where $n = 2$, the function $y = x_1 + x_2$ has been plotted in Fig. 1 (top). The best prediction in the absence of any features is $\mathrm{E}[Y] = 0$, which is the zero-height plane, the x_1-x_2 plane, cutting across the middle of the target surface. Half of the target plane is above this plane and half of it below.

The risk of this constant prediction (using the squared error loss function) is

$$R(0) = \mathrm{E}[(Y-0)^2] = \mathrm{E}[(X_1+X_2)^2] = \int_{-1}^{1}\int_{-1}^{1} (x_1+x_2)^2 \frac{1}{4} dx_1 dx_2 = \frac{2}{3} \tag{4}$$

where $f_{X_1X_2}(x_1,x_2) = \frac{1}{2}\,\frac{1}{2} = \frac{1}{4}$, $\forall x_1, x_2 \in (-1,1)$, is the uniform joint density function of the two independent input features X_1 and X_2, and is computed as the product of the individual marginal densities. Note that, the risk function in (4) is the variance of $Y = X_1 + X_2$, which reduces to the sum of the variances of X_1 and X_2 (since they are independent, and therefore, uncorrelated).

When the only available feature is X_1, the projection of the target plane (onto the x_2-y plane) is plotted in Fig. 1 (bottom) over the range of x_1. Note that, this is no longer a function, since for each value of x_1, multiple values of y are possible, all equally likely. The best prediction (or solution) in this case is $h(\mathbf{x}) = x_1$. This is a line that goes through the middle of



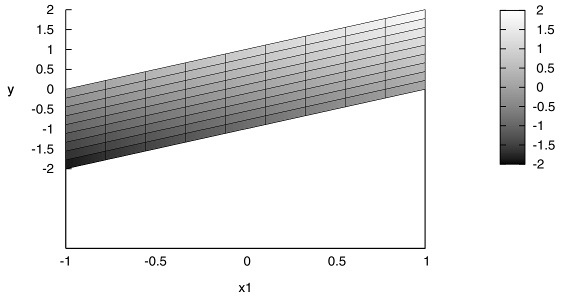


Fig. 1. The target plane $x_1 + x_2$ in $[-1,1]^2$ (top), and its projection where the only available input feature is x_1 (bottom).

the projected plane. Its risk is given by

$$R(h) = \mathrm{E}[(Y-X_1)^2] = \mathrm{E}[(X_1+X_2-X_1)^2] = \mathrm{E}[X_2^2] = \int_{-1}^{1} x_2^2 \,\frac{1}{2} dx_2 = \frac{1}{3} \ ,$$

which is the variance of X_2, since $\mathrm{E}[X_2] = 0$.

The predictor function $h(\mathbf{x}) = x_1$ is a partial solution, since $R(h) < R(0) = R(\mathrm{E}[Y])$. It is not a perfect solution, but it performs better than the constant prediction $\mathrm{E}[Y] = 0$. Since its error or risk is lower than that of constant or other irrelevant functions, it is more likely to survive during the search (or evolution), and since it has one of the necessary features of the perfect solution $h^*(\mathbf{x}) = x_1 + x_2$, it is likely to contribute towards finding the best solution.

The prediction function $h(\mathbf{x}) = x_1$ is in fact not the only function of x_1 that is a partial solution. Many other functions of x_1, such as $\frac{1}{5}x_1$ and $x_1 + \frac{1}{2}$, perform better than a constant prediction, and can all contribute towards finding the best solution.

Example 2. *A regression problem that allows partial solutions only in the first category*

We now turn our attention to a regression problem where there is no partial solution of proper (or strict) subsets of features (that is, partial solutions belonging to the second category).

Consider a regression problem with n mutually independent and identically distributed input features $X_1, X_2, \ldots, X_n$, where $X_i \sim \mathcal{U}(-1, 1)$, for all $i \in \{1, \ldots, n\}$, and the target function is $Y = \prod_{i=1}^{n} X_i$. This example is identical to the previous example, except that summation has been replaced by product. Here, the perfect solution (since noise and irrelevant features are absent) is $h^*(\mathbf{x}) = \prod_{i=1}^{n} x_i$.

For the case where $n = 2$, the function $y = x_1 x_2$ has been plotted in Fig. 2 (top). Again, the best prediction in the absence of any features is $\mathrm{E}[Y] = 0$. The risk of this constant prediction, using the squared error loss function, is given by

$$R(0) = \mathrm{E}[(Y - 0)^2] = E[(X_1 X_2)^2] \\ = \int_{-1}^{1} \int_{-1}^{1} (x_1 x_2)^2 \frac{1}{4} dx_1 dx_2 = \frac{1}{9} \ .$$

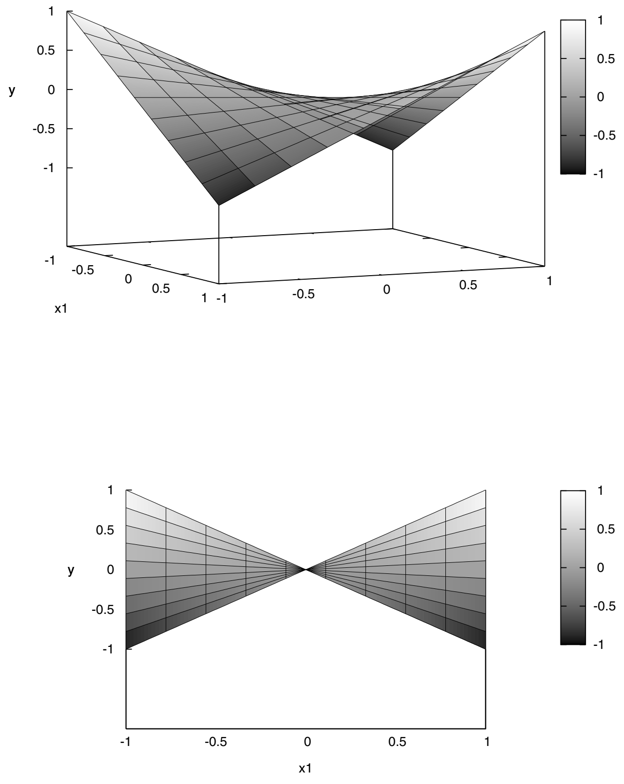


Fig. 2. The plane $x_1 x_2$ in $[-1, 1]^2$ (top), and its projection where the only available input feature is x_1.

When the only available feature is X_1, the projection of the target plane (onto the x$_2$-y plane) is plotted in Fig. 2 (bottom) over the range of x_1. In this example, the projection is symmetric with respect to the line $y = 0$, and therefore, the best predictor when only the feature X_1 is available, is still the constant function $h(\mathbf{x}) = 0$.

In fact, any function of x_1 that slightly deviates from 0 will have a higher risk (or error) than $h(\mathbf{x}) = 0$. For instance, the risk for the function $h(\mathbf{x}) = x_1$ is

$$R(h) = \mathrm{E}[(Y - X_1)^2] = \mathrm{E}[(X_1 X_2 - X_1)^2] \\ = \int_{-1}^{1} \int_{-1}^{1} (x_1 x_2 - x_1)^2 \frac{1}{4} dx_1 dx_2 = \frac{4}{9} \ .$$

Therefore, $R(h) > R(0) = R(E[Y])$ in this example, which implies that the prediction function $h(\mathbf{x}) = x_1$ is not a partial solution. This is counter-intuitive, because in a symbolic regression setting, the expression x_1 is one step closer to the best solution, but its error is higher than that of the constant prediction $\mathrm{E}[Y] = 0$, which does not use any of the input features. In fact, if this problem had an irrelevant feature x_{noise}, then the symbolic expression $x_{noise} - x_{noise}$ would perform better than x_1. This may guide the search (or evolution) in a wrong direction. In conditions like this, partial solutions are a subset of functions that use all the relevant features. For instance, the prediction function $h(\mathbf{x}) = \frac{1}{2} x_1 x_2$, having risk $R(h) = \frac{1}{36}$, is a partial solution in this problem, since $R(h) < R(E[Y])$.

From a syntactic point of view, $\sum_{i=1}^{n} x_i$ and $\prod_{i=1}^{n} x_i$ are very similar. Changing the $+$ to $\times$ in the expression tree that solves the first problem, will solve the second problem. In fact, the number of expression trees whose phenotype is the first expression (the sum problem), is equal to the number of expression trees whose phenotype is the second expression (the product problem). However, the second problem is much harder to learn because partial solutions are sparse.

One must note that, this is not simply the difference between linear and non-linear problems. Many non-linear problems do allow partial solutions form the second category. In fact, in the product example, if the ranges of the features are, for instance, all $[0, 1]$, the projections become non-symmetric, and therefore, expressions containing smaller numbers of relevant features can contribute towards reducing the risk.

C. Feature Bundles

This section formalizes the phenomenon that was observed in the examples of the previous section.

Let $\mathcal{H}$ be the hypothesis space in a symbolic regression problem, i.e. the set of all expressions constructed from a set of terminals and primitive functions (to a maximum depth). Let $\mathcal{F}$ be the set of available features and let $\mathrm{symvars} : \mathcal{H} \to \mathcal{F}$ be a function that takes an expression as input, and returns the set of features (or variable terminals) that are used in that expression. For instance, $\mathrm{symvars}(x_1^2 - x_3) = \{x_1, x_3\}$. The set of relevant features, which we denote by $\mathcal{V}$, can then be defined as

$$\mathcal{V} = \text{symvars}(h^\star) \ ,$$

where $h^\star$ is the best solution and is obtained from (1).

Definition 2. A *feature bundle* is a set $\mathcal{B} \subseteq \mathcal{V}$, such that

$$\min_{h \in \mathcal{H}_\mathcal{S}} R(h) \geq R(\text{E}[Y]) \ \ , \forall \mathcal{S} \subset \mathcal{B} \ \ ,$$

where $\mathcal{H}_\mathcal{S} = \{h \in \mathcal{H} : \text{symvars}(h) = \mathcal{S}\}$.

That is, the smallest risk associated with the candidate solutions $h \in \mathcal{H}_\mathcal{S}$, constructed from any strict subset $\mathcal{S}$ of the feature bundle $\mathcal{B}$, is at least as large as the risk associated with the constant function $\text{E}[Y]$. Therefore, by definition, there are no partial solutions that use only a proper subset of a feature bundle. All the elements of a feature bundle must be present in an expression for the expression to have an error lower than that of the constant prediction, and therefore, have a chance of survival in the search for the best solution.

A search process (e.g. an evolutionary symbolic learning algorithm) will behave in a manner similar to a random search until it (accidentally) finds a partial solution that contains all the features in the feature bundle. This implies that problems with large feature bundles can become difficult to solve.

In the following section, we will demonstrate how the size of a feature bundle can affect the chance (or probability) of finding a solution.

III. Experimental Design

In the previous section, we showed that in symbolic regression, in the presence of feature bundles, the search algorithm has to first (randomly) arrive at an entire feature bundle in order to have a partial (or complete) solution. We saw that there is no fitness gain for *any* function that uses only a strict subset of a feature bundle.

In this section, we describe a set of experiments that we used to examine how the likelihood of finding a feature bundle is affected by the number of features and the size of the feature bundle in the problem.

A. Problems

For most "real" regression problems (or datasets), the correct solution, relevant features, size of feature bundles, and other properties are not known. Therefore, since the goal of our experiments is to see how an evolutionary process for finding symbolic expressions performs with respect to the size of feature bundles, we use a number of "synthetic" problems in which the sizes of the feature bundles are known.

In each problem, we have $|\mathcal{F}|$ features (see Table I for the specified values). All the relevant features in the problem are part of a single feature bundle of size $|\mathcal{B}|$. Various values of $|\mathcal{F}|$ and $|\mathcal{B}|$ yield a total of 20 problems. The problems are constructed such that there is no redundancy between the relevant features in each problem. Also, all the relevant features in a given problem are "noise"-free; that is, the target function is a deterministic function of the input features, and thus, their discovery is not hindered by anything other than the search strategy. We also assume that the target variable is independent of features in the set $\mathcal{F} \setminus \mathcal{B}$, which is the set of available features in the problem that are not in the feature bundle; that is, all features not included in the feature bundle are pure "noise" and irrelevant.

B. Search Process

Since the most common way of conducting the search for symbolic expressions is evolution (both in genetic programming and gene expression programming), we have adopted an evolutionary search over the expression trees. The average number of terminals in expression trees is correlated with the average depth of trees. For this reason, we have increased the maximum depth for problems with larger numbers of features.

We assume that all the primitive functions required to construct the perfect solution are known and available in the function set. The arity of the functions and the maximum depth are set well above the level that is required to construct a perfect solution.

We have used three standard operators: the crossover operator, the mutation operator, and the reproduction operator. The rate of these operators and various other settings of the experiments are available in Table I.

TABLE I
Simulation settings

Parameter	Symbol	Value(s)
Number of features	$\|\mathcal{F}\|$	$\{10, 20, 50, 100\}$
Size of feature bundle	$\|\mathcal{B}\|$	10%, 30%, 50%, 80%, and 100% of the number of features
Number of problems		possible numbers of features $\times$ possible sizes of feature bundles $= 20$
Population size		1000
Max num. of generations		500
Max depth		set based on the number of features in the problem
Crossover rate		0.9
Mutation rate		0.01
Reproduction rate		$1 - 0.9 - 0.01 = 0.09$
Runs		50

The search (or evolution) process is terminated when either the maximum number of generations is reached or as soon as an expression is found that contains all the features in the feature bundle (or more). Note that, finding all the features in a feature bundle does not necessarily mean that the regression problem is solved, since the internal nodes (or primitive functions) of the expression must also be of the correct type and order. However, finding the feature bundle makes it possible to have partial solutions which, due to higher fitness (i.e. lower error/risk), will have a higher chance of survival, and therefore, a higher chance of leading to the final solution.

The simulations were repeated 50 times in order to obtain a reliable enough estimate of the number of evaluations required in order to arrive at a feature bundle.

C. Remarks

We have made a number of assumptions in our experiments for the purpose of simplification. However, since the goal of this research is to highlight some limitations of canonical symbolic regression methods in special situations (i.e. in the presence of feature bundles), the simplifying assumptions do not weaken our conclusions.

The experiments are designed in a way that the results provide a notion of upper bound for the probability of finding a solution. In more "real" scenarios and applications where some of these conditions are not met, the actual performance of canonical symbolic regression algorithms will be worse than what is being reported here (and hence, our bounds will still hold).

IV. Results and Discussion

The results of our simulation experiments, for various numbers of features ($|\mathcal{F}|$) and various sizes of feature bundles ($|\mathcal{B}|$), are presented in Table II. The maximum depth has been set manually to increase as the number of features in the problem increases. Deeper expression trees allow more leaves (or variable terminals), but they also allow more internal nodes (or primitive functions), which adds to the complexities of the search space.

TABLE II
Problems and Simulation Results

$\|\mathcal{F}\|$	$\|\mathcal{B}\|$	depth	$<$ evaluations $>$	success rate
10	1	4	1.82	1.0
10	3	4	5.28	1.0
10	5	4	23.14	1.0
10	8	4	330.72	1.0
10	10	4	2168.76	1.0
20	2	5	3.2	1.0
20	6	5	38.18	1.0
20	10	5	889.78	1.0
20	16	5	129395.98	1.0
20	20	5	-	0.14
50	5	6	38.06	1.0
50	15	6	204294.58	0.98
50	25	6	-	0.0
50	40	6	-	0.0
50	50	6	-	0.0
100	10	7	2118.24	1.0
100	30	7	-	0.0
100	50	7	-	0.0
100	80	7	-	0.0
100	100	7	-	0.0

The fourth column in Table II is the average number of evaluations required to reach a solution that contains all the features in the feature bundle. Since any partial or complete solution must contain all of the features in a feature bundle, these numbers are lower bounds for the number of evaluations required for an optimal solution to be found.

Note that, the average number of evaluations can be viewed as the estimated expected value or mean of a random variable Z having a geometric distribution, i.e. Z counts the number of Bernoulli (success or failure) trials until the first success is observed. This geometric distribution has probability mass function $\mathrm{P}(Z = z) = (1-p)^{z-1}p$, for $z \in \mathbb{N}_+$, with expected value $\mu = \frac{1}{p}$, where $p \in [0, 1]$ is the probability of success in each of the independent Bernoulli trials. Therefore, the average number of evaluations required to find the first subset of features that includes the desired feature bundle is $\hat{\mu} = \bar{Z}$, an estimator of the mean μ.

The last column of Table II indicates the proportion of runs in which the algorithm managed to find the desired feature bundle.

Even though the averages (of the numbers of evaluations) reported in Table II are very optimistic, note that, as the number of features and the size of the feature bundle in a problem grow, the amount of computation required to find a (partial) solution soon moves beyond reach. The chance of discovering a feature bundle with over 20 features is extremely low. In almost all the cases, none of the runs were able to find one. Note that, in Table II, for the case where all the 20 features in the problem were part of the feature bundle, only 14% of the runs were successful in finding the feature bundle, and therefore, a reliable estimate of the average number of evaluations could not be obtained. It can also be seen that in all cases as the ratio $\frac{|\mathcal{B}|}{|\mathcal{F}|}$ approaches 1 the computational effort required to find the feature bundle increases exponentially.

The average number of evaluations in Table II, can be used to estimate the probability of success—that is, the probability of finding a set of features that contains a feature bundle—for each problem, for a given number of evaluations. The estimated probability for each Bernoulli trial, which is defined as whether or not a set containing the feature bundle was found in a given evaluation (or trial), is $\hat{p} = \frac{1}{\hat{\mu}}$, where $\hat{\mu}$ is the average number of evaluations. The probability of achieving success within t evaluations is computed using the cumulative probability distribution function of the geometric distribution, which is given by

$$\begin{aligned} \mathrm{P}\{\text{success}\} &= \mathrm{P}\{Z \leq t\} = 1 - \mathrm{P}\{Z > t\} \\ &= 1 - (1-p)^t \ , \end{aligned} \tag{5}$$

for $t \in \mathbb{N}_+$, where $(1-p)^t$ represents the probability that all t evaluations end in failure. Note that, for a fixed t, this probability is an increasing function of the probability p of success in each evaluation, and for a fixed p, it is an increasing function of the number of evaluations t.

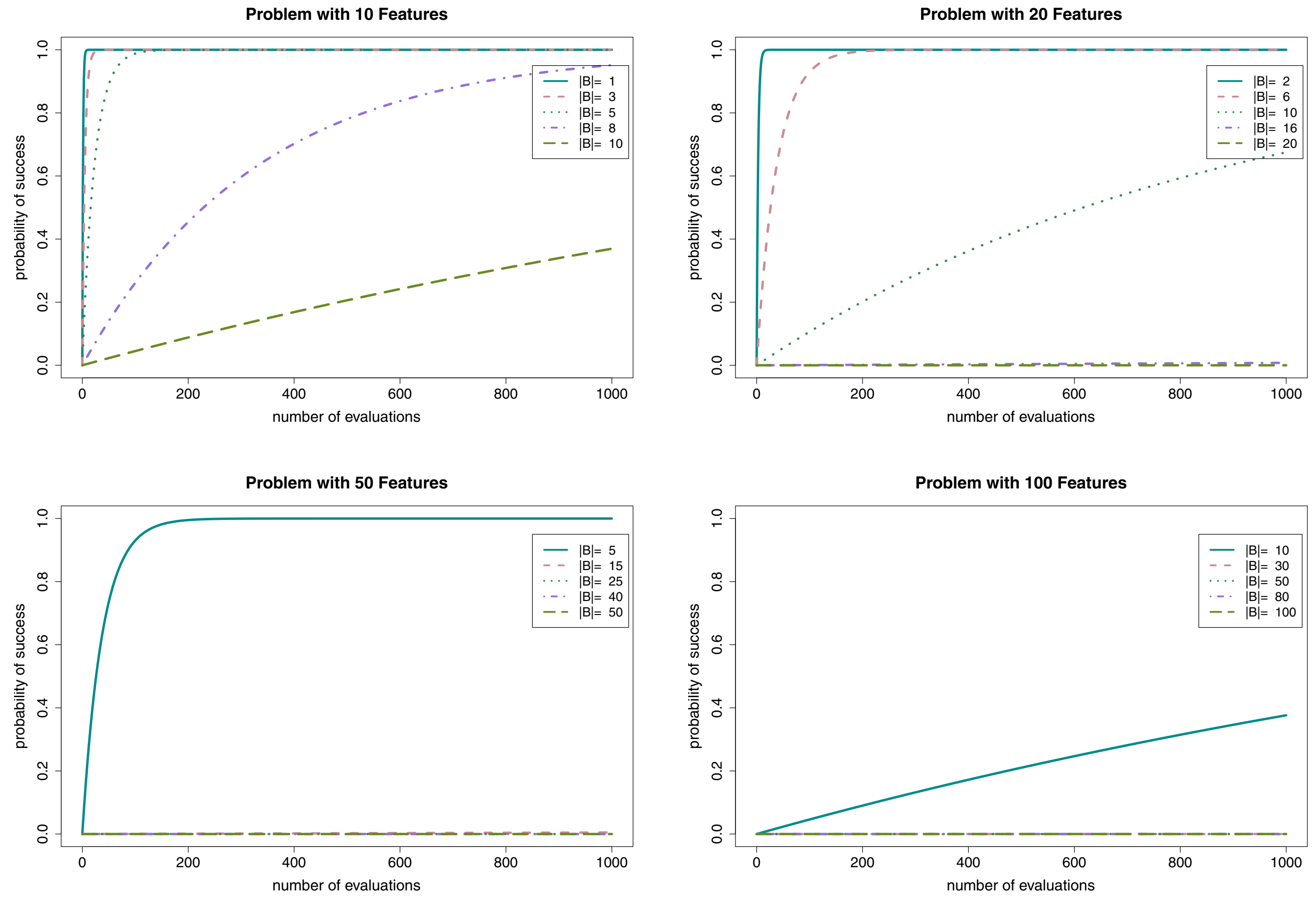


Fig. 3. The estimated probability of success in finding feature bundles, plotted as a function of the number of evaluations, for various numbers $|\mathcal{F}|$ of features and various sizes $|\mathcal{B}|$ of feature bundles, where $|\mathcal{F}| \in \{10, 20, 50, 100\}$; and $|\mathcal{B}| \in \{r|\mathcal{F}| : r \in \{10\%, 30\%, 50\%, 80\%, 100\%\}\}$.

To estimate the probability of success in (5), we use $\hat{p}$ derived from the results in Table II. The estimated probability $\hat{\mathrm{P}}\{\text{success}\} = 1 - (1 - \hat{p})^t$ is, as a function of the number of evaluations t, plotted in Fig. 3, for various values of $|\mathcal{F}|$ and $|\mathcal{B}|$ (which yield different $\hat{p}$ values). The four plots in the figure correspond to the different numbers of features (i.e. values of $|\mathcal{F}|$ from Table II), and the curves within each plot correspond to the different sizes of the feature bundles (i.e. corresponding values of $|\mathcal{B}|$ from Table II).

Notice that, within each plot in Fig. 3, the probability of success at any t, is a decreasing function of $|\mathcal{B}|$; that is, the probability curve corresponding to a lower value of $|\mathcal{B}|$ is, for all t, greater than the probability curve corresponding to a higher value of $|\mathcal{B}|$. Also, for each $|\mathcal{F}|$, the probabilities are decreasing functions of the ratio $\frac{|\mathcal{B}|}{|\mathcal{F}|}$.

For higher numbers of features, for instance, $|\mathcal{F}| = 50$ or $|\mathcal{F}| = 100$, the success probabilities for most sizes of the feature bundles are effectively zero—this can be seen in the cluster of probability curves close to the bottom of the corresponding plots. (Note that, although the number of evaluations is a discrete variable, i.e. $t \in \mathbb{N}_+$, we have plotted the probability functions as continuous curves, in order to reduce clutter and more clearly demonstrate the trends.)

As stated earlier, these numbers are upper bounds. That is, the actual probabilities of success can be lower than what is presented in the plots, as the actual numbers of evaluations required to reach a feature bundle can be higher than those averaged in Table II.

V. Conclusions

In this paper, we showed, through construction, that there are problems where there is no function of subsets of a set of relevant features that would perform any better than the constant prediction, the expected value of the target variable. We called these subsets of features 'feature bundles' and showed that as the size of a feature bundle increases, the probability of finding it becomes increasingly difficult.

It has been known that noise and irrelevant features are challenging issues in learning, but we showed that even in the absence of such issues, and in cases where the target function is completely deterministic and noise-free, finding a feature bundle is still challenging.

Finding feature bundles is important, because without them, there is no survivable partial solution and the evolutionary process (or any other meta-heuristic algorithm) reduces to a random search.

To illustrate this, we showed how finding a symbolic regression model that is as simple as the product of input features, becomes very challenging when there is no partial solution before finding (or rather randomly arriving at) all the relevant features.

The limitations presented in this paper provide opportunities for new research in this area. One possible research direction is to identify a classification of symbolic regression problems and their hardness.

REFERENCES

[1] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[2] ——, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press, May 1994.

[3] ——, *Genetic programming III: Darwinian invention and problem solving*. Morgan Kaufmann Pub, 1999.

[4] M. F. Korns, "Large-Scale, Time-Constrained symbolic Regression-Classification," in *Genetic Programming Theory and Practice V*. Springer US, 2008, pp. 53–68.

[5] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 242–258, 2000.

[6] G. Patterson and M. Zhang, "Fitness functions in genetic programming for classification with unbalanced data," in *AI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 769–775.

[7] D. A. Augusto and H. J. Barbosa, "Symbolic regression via genetic programming," in *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*. IEEE, 2000, pp. 173–178.

[8] F. Castillo, A. Kordon, and C. Villa, "Genetic programming transforms in linear regression situations," *Genetic Programming Theory and Practice VIII*, pp. 175–194, 2011.

[9] T. McConaghy, T. Eeckelaert, and G. Gielen, "Caffeine: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming," in *Design, Automation and Test in Europe, 2005. Proceedings*. IEEE, 2005, pp. 1082–1087.

[10] M. F. Korns, "Abstract expression grammar symbolic regression," *Genetic Programming Theory and Practice VIII*, pp. 109–128, 2011.

[11] R. Riolo, T. Soule, and B. Worzel, *Genetic programming theory and practice vi*. Springer Science & Business Media, 2008.

[12] C. Ferreira, "Function finding and the creation of numerical constants in gene expression programming," in *Advances in soft computing*. Springer, 2003, pp. 257–265.

[13] ——, "Gene expression programming in problem solving," in *Soft computing and industry*. Springer, 2002, pp. 635–653.

[14] Q. Chen, M. Zhang, and B. Xue, "Improving generalisation of genetic programming for symbolic regression with structural risk minimisation," in *Genetic and Evolutionary Computation Conference, 2016. Proceedings*. ACM Press. Dever, Colorado, USA., 2016, pp. 709–716.

[15] G. F. Smits, E. Vladislavleva, and M. E. Kotanchek, "Scalable symbolic regression by continuous evolution with very small populations," in *Genetic Programming Theory and Practice VIII*. Springer, 2011, pp. 147–160.

[16] M. Oltean and L. Dioşan, "An autonomous gp-based system for regression and classification problems," *Appl. Soft Comput.*, vol. 9, pp. 49–60, January 2009.

[17] Q. Chen, M. Zhang, and B. Xue, "Genetic programming with embedded feature construction for high-dimensional symbolic regression," in *The 20th Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*. Springer, 2016, pp. 87–102.

[18] ——, "Feature selection to improve generalisation of genetic programming for high-dimensional symbolic regression," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 792–806, 2017. [Online]. Available: https://doi.org/10.1109/TEVC.2017.2683489

[19] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *Trans. Sys. Man Cyber Part C*, vol. 40, pp. 121–144, March 2010.

[20] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evol. Comput.*, vol. 14, pp. 309–344, September 2006.

[21] R. Hunt, K. Neshatian, and M. Zhang, "Scalability analysis of genetic programming classifiers," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*, 2012, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CEC.2012.6256520

[22] B. Xue, M. Zhang, W. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *Evolutionary Computation, IEEE Transactions on*, vol. 20, no. 4, pp. 606–626, 2016. [Online]. Available: https://doi.org/10.1109/TEVC.2015.2504420

[23] R. T. Trevor Hastie and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, Springer New York Inc., 2001.

[24] A. C. Rencher and G. B. S. (2nd ed.), *Linear Models in Statistics*. John Wiley & Sons, Inc., 2008.

[25] V. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.