On affine symbolic regression trees for the solution of functional problems

M. D. Champneys[1,2], N. Dervilis[2], K. Worden[2]

[1]Industrial Doctorate Centre in Machining Science,
Advanced Manufacturing Research Centre with Boeing,
University of Sheffield, S60 5TZ, UK

[2]Dynamics Research Group,
Department of Mechanical Engineering,
University of Sheffield,
Mappin Street, Sheffield, S1 3JD, UK

Abstract

Symbolic regression has emerged from the more general method of Genetic Programming (GP) as a means of solving functional problems in physics and engineering, where a functional problem is interpreted here as a search problem in a function space. A good example of a functional problem in structural dynamics would be to find an exact solution of a nonlinear equation of motion. Symbolic regression is usually implemented in terms of a tree representation of the functions of interest; however, this is known to produce search spaces of high dimension and complexity. The aim of this paper is to introduce a new representation – the affine symbolic regression tree. The search space size for the new representation is derived and the results are compared to those for a standard regression tree. The results are illustrated by the search for an exact solution to several benchmark problems.

Key words: Symbolic Regression, Genetic Programming, Solutions of Differential Equations, Search Space Analysis

1 Introduction

Differential equations are among the most fundamental tools available for scientific analysis. No other mathematical object has the same ability to describe change in the physical systems. It is therefore no wonder that differential equations have been so extensively studied. Tireless investigation into the specification, existence and uniqueness of solutions to differential equations has borne some of the most eminently useful techniques in engineering. Exact solutions to linear differential equations have enabled *modal analysis*, an invaluable analysis tool for linear dynamic systems. However, as the materials, geometries, demands and capabilities of engineering structures become ever more complex, a linear description of the physics becomes less and less applicable.

By far the majority of nonlinear differential equations are without exact solution. This reality hamstrings the development of powerful analysis tools that might be viewed as nonlinear alternatives to modal analysis. In the place of pure mathematical solutions, powerful heuristic methods have been developed.

Heuristic methods fall into a number of distinct categories based on the form of the yielded solution. Numerical methods provide approximations to the true solution at a finite number of points in the domain. Analytical-approximate methods such as the shooting method [1], or harmonic balance [2], provide analytic approximations, often by the consideration of series expansions.

A third class of approach, referred to hereafter as pure-heuristic methods, offer both the possibility of an approximate and exact solution. Such approaches are often (but not always [3, 4]) based on a standard Symbolic Regression (SR), a specific application of the more general method of Genetic Programming (GP) [5]. A large number of authors have proposed SR for solving differential equations in the last three decades. Highlights include the earliest suggestions of the approach in [5], a reverse-Polish notation-based tree structure in [6], a hybrid-analytical approach in [7], a grammar-based approach with a benchmarking set in [8], and a Cartesian Genetic Programming (CGP) approach in [9].

Despite some progress, no single approach has yet broken through as a significant step forward. A potential reason is that these methods suffer from inherent difficulties arising from the complexity of the underlying search problems. A recent body of work has focussed on ways that the performance of SR algorithms applied to the solution of differential equations might be improved.

This paper opens with a thorough mathematical description (based in part on the work of Seaton et al. in [9]) of the search problem underpinning SR-based solution of differential equations. Of the elements defined in this description, the current study is focused on the encoding space. The main contribution of this paper is a novel scheme for representing expressions in SR - the *affine regression tree.* The sizes of the search spaces for this new representation, as well as two existing schemes, are enumerated. The potential of the new representation is demonstrated with a benchmark case-study.

2 Symbolic regression is a search problem

Symbolic regression casts the specification of an analytic relationship of interest as an optimisation problem. This approach has been undertaken by many authors attempting to develop heuristic solution methods for differential equations. In this section, notation is presented that describes the underlying search problem formally.

The search problem of locating a solution $f^*(x)$ to a general differential equation $\mathbf{\Psi}(f, x)$ is defined as the set of the following elements (the definitions in parentheses are the equivalent terms used in the GP literature).

- An encoding space (Genotype): $\mathbf{E}$.
- A decoding function (Phenotyping function): $P : \mathbf{E} \to \mathbf{\Omega}$.
- A functional interpretation function: $S : \mathbf{\Omega} \to \mathbf{C^r}$.
- An objective function (Fitness function): $J : \mathbf{C^r} \to \mathbb{R}$.
- One or several search operations (Genetic operations): $M_i : \mathbf{E} \to \mathbf{E}$.

A brief definition of each of the above is offered here. The encoding space $\mathbf{E}$ represents the set of trial expressions f. This space can be thought of as the overall search space. Each point in the space $\mathbf{E}$ is a structurally (but not necessarily mathematically) unique representation of a mathematical expression. Points in the encoding space are mapped via a decoding function P into the space of representable expressions (Denoted as $\mathbf{\Omega}$ following [9]).

Each point in $\mathbf{\Omega}$ is a structurally-unique mathematical expression. However, this is still not enough to guarantee mathematical uniqueness. Consider the expressions,

$$\frac{(10x + 2)}{2} \tag{1}$$

and,

$$5x + \cos^2(x) + \sin^2(x) \tag{2}$$

The above expressions can be trivially manipulated to demonstrate equality but they have different expression structures and rely on different bases of operations in their representation. In order to test for mathematical equality, a further projection S is required. Intuitively, this mapping is the semantic realisation of an expression in the function space. Since solutions to differential equations must necessarily be differentiable up to the order of the differential equation r, the relevant function space is denoted $\mathbf{C}^r$ as is conventional. The subspace $\omega \subseteq \mathbf{\Omega}$ is additionally defined by,

$$u \in \mathbf{\Omega}, \quad S(u \in \omega) = f^* \tag{3}$$

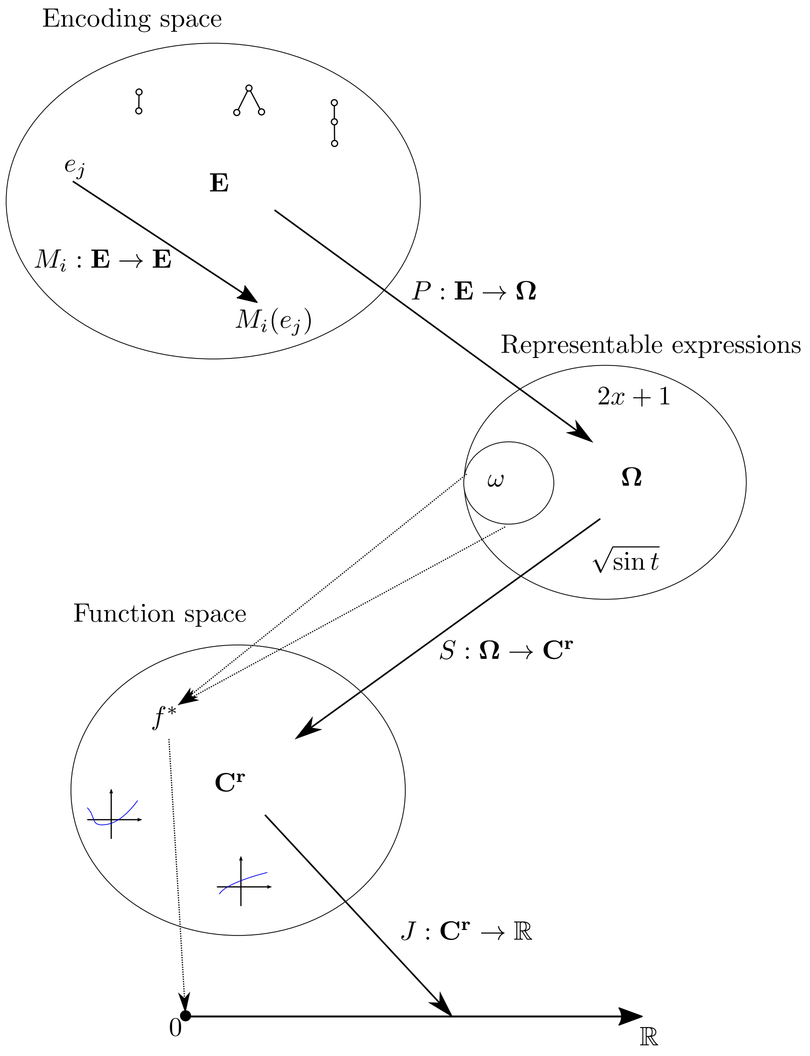

Encoding space
e_j
$\mathbf{E}$
$M_i : \mathbf{E} \to \mathbf{E}$
$M_i(e_j)$
$P : \mathbf{E} \to \mathbf{\Omega}$
Representable expressions
$2x + 1$
ω
$\mathbf{\Omega}$
$\sqrt{\sin t}$
Function space
$S : \mathbf{\Omega} \to \mathbf{C^r}$
f^*
$\mathbf{C^r}$
$J : \mathbf{C^r} \to \mathbb{R}$
0
$\mathbb{R}$

Figure 1: Visualising the underlying search problem.

as the subset of expressions in $\mathbf{\Omega}$ that are mathematically equivalent to the solution to the differential equation. An objective function J maps trial functions to values on the real line. In the case of a minimisation problem, J is constructed such that,

$$J(f^*) = 0, \quad J(f \neq f^*) > 0 \tag{4}$$

The final ingredients in the problem specification are the search operations M_i, these are perturbing functions that move points around in the expression space. The structure of the spaces is summarised in Figure 1. It is clear that the appropriate selection of these elements is instrumental in controlling the overall difficulty of the search problem. An ongoing body of work investigates heuristics for optimal selection of these components. This paper will focus on the selection of the encoding space $\mathbf{E}$. Seaton et al. define a heuristic metric for comparison of encoding spaces termed *unguided complexity* in [9]. The method relies on estimating the ratio,

$$k = \frac{\mu(\mathbf{\Omega})}{\mu(\omega)} \tag{5}$$

by enumerating the search space and sampling from the space of representable expressions by some search method. Here, μ is the counting measure from measure theory used to evaluate the size of the spaces. The quantity k^{-1} can be interpreted as the probability of sampling the solution at random from all representable expressions. However, in the absence of methods for direct enumeration of $\mu(\omega)$ or an efficient method for uniformly sampling from ω, this metric is limited to compact expressions and small encoding spaces. Because of this issue, the analysis conducted in this paper will focus on a direct comparison of $\mu(\mathbf{\Omega})$ for different representation schemes.

3 Enumerating search spaces

In order to evaluate the unguided complexity, one must first be able to evaluate $\mu(\mathbf{\Omega})$. In the current work this is accomplished by considering the encoding space $\mathbf{E}$. A common choice [10, 5, 11] for $\mathbf{E}$ is to use a tree structure. This approach has the nice property that all mathematical expressions can be mapped in an isomorphic way to a tree object,

$$\mathbf{E}_{\text{tree}} \simeq \mathbf{\Omega}_{\text{tree}} \tag{6}$$

and therefore the spaces have the same measure,

$$\mu(\mathbf{E}_{\text{tree}}) = \mu(\mathbf{\Omega}_{\text{tree}}) \tag{7}$$

The size of the search space of trees is discrete but naturally not finite. Practically, one places a restriction on the size of tree structures that are permitted. Several approaches are possible including limits on tree depth or internal nodes [3]. However, the approach here will be to restrict the total number of nodes in the expression tree. This has the advantage of not placing any bias upon either very deep or very wide trees. Another advantage of using the total number of nodes is that it is equivalent to maximum tree depth by the relation $n = m^h$ where n is the number of nodes, m is the maximum number of child nodes connected to any given node (hereafter referred to as the *arity* of that node) and h is the tree depth.

The maximum arity of any node defines the arity of the tree structure. The number of unlabelled m-ary trees with exactly n nodes is given by the Fuss-Catalan numbers [12],

$$C_n = \frac{1}{(m-1)n+1}\binom{mn}{n} \tag{8}$$

However, the nodes in an expression tree representation are not unlabelled. Instead, each node of arity $i \in [0, \ldots, m]$ derives a label from set f_i, the edges are unlabelled. Thus, the overall label set,

$$F = \{f_0, f_1, \ldots, f_i, \ldots, f_m\} \tag{9}$$

is defined as the *search basis*. In the literature, the case $i = 0$ is sometimes considered separately with 0-ary nodes referred to as *terminals* or *leaves* and $f_0 = T$ referred to as the terminal set. For compactness, this notation will not be used here.

Implementations of the search problem must necessarily select a basis that contains all the mathematical objects needed to describe the solution or else only approximate solutions will be possible. For example when considering linear-ODE problems it is essential to include sinusoids and exponential terms.

The size of the tree space clearly depends on the maximum arity m, the maximum number of nodes n and the basis set F. Let,

$$T_j = \mu\left(\mathbf{T}(m, j, F)\right) \tag{10}$$

be the number of possible node-labelled m-ary trees with basis F comprised of exactly j nodes. By considering trees of all sizes up to n, one has,

$$\mu(\mathbf{E}_{\text{tree}}(m, n, F)) = \sum_{j=0}^{n} T_j \tag{11}$$

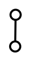
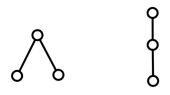
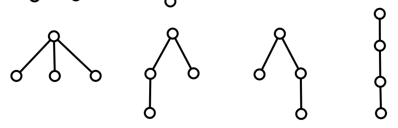
n	m		Possible trees
1	0		f_0
2	1		$f_1 f_0$
3	2		$f_2 f_0^2 + f_1^2 f_0$
4	3		$f_3 f_0^3 + f_2 f_1 f_0^2 + f_2 f_1 f_0^2 + f_1^3 f_0$

Figure 2: Possible labelled trees with up to $n = 4$ nodes.

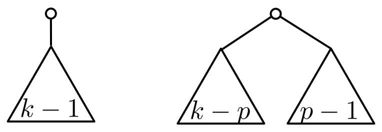


Figure 3: Possible trees with $m = 2, n = k$, $p \in [2, k-1]$.

In order to derive this quantity, consider the case $n = 1$. Since this can only be a single node with no children the number of possible labels (and therefore trees) is,

$$T_1 = f_0 \tag{12}$$

Next, consider the case $n = 2$; there is still only a single possible tree structure (one with one root and one child). The number of such trees is equal to the number of label combinations,

$$T_2 = f_1 f_0 \tag{13}$$

This process is displayed graphically up to $n = 4$ in Figure 2. A recurrence relation can now be derived for the case $n = k$. In order to simplify the notation, the following derivation will continue in the case $m = 2$. For basis functions of analytical expressions this is a realistic restriction as there are few common analytical expression operations with an arity greater than two (an example might be a summation with limits on indices - technically a trinary operation but these are not often seen in exact solutions to differential equations).

In the case $m = 2, n = k$ there are two important cases to consider as shown in Figure 3. In the first, the root node is unary. Its single argument is a subtree with $k - 1$ nodes. In this case there are,

$$T_{k,\text{unary}} = f_1 T_{k-1} \tag{14}$$

possible trees. In the other case the root node is binary and the number of nodes in its arguments sum to $n - 1$. In performing this sum one has,

$$T_{k,\text{binary}} = f_2\{T_{k-2}T_1, T_{k-3}T_2, \ldots, T_1 T_{k-2}\} = f_2 \sum_{p=2}^{k-1} T_{k-p}T_{p-1} \tag{15}$$

The required relation is now the sum of these two possibilities,

$$T_k = T_{k,\text{unary}} + T_{k,\text{binary}} = f_1 T_{k-1} + f_2 \sum_{p=2}^{k-1} T_{k-p}T_{p-1} \tag{16}$$

Given $T_1 = f_0$ and $T_2 = f_1 f_0$ from above, one can construct a recursive scheme to calculate any $\mu(\mathbf{E}_{\text{tree}}(2, n, F))$ and thus the number of structurally unique expressions $\mu(\boldsymbol{\Omega}_{\text{tree}})$,

$$\mu(\mathbf{E}_{\text{tree}}) = \mu(\boldsymbol{\Omega}_{\text{tree}}) = \sum_{k=1}^{n} \left[f_1 T_{k-1} + f_2 \sum_{p=2}^{k-1} T_{k-p} T_{p-1} \right] \tag{17}$$

Another common representation scheme is a grammar-based structure [8, 13]; these are employed by several authors alongside techniques such as modular arithmetic to encode expressions as vectors of integers. The search space of grammar-based representations can conveniently be enumerated by considering the number of unique tree structures. This process gives a better estimate of $\mu(\boldsymbol{\Omega}_{\text{grammar}})$ than simply raising the maximum integer value to the power of the vector length (i.e $\mu(\mathbf{E}_{\text{grammar}})$). This advantage arises because the modular arithmetic removes the need for a maximum integer value resulting in a many-to-one mapping via the decoding function. Practically, this means that unlike the expression tree representation,

$$\mu(\mathbf{E}_{\text{grammar}}) \gg \mu(\boldsymbol{\Omega}_{\text{grammar}}) \tag{18}$$

Another advantage is that an equivalent tree enumeration allows better comparison with other representation schemes. Since grammar-based structures sometimes include trinary operations, an additional term in equation (16) is required. The number of distinct trees with k nodes is now,

$$\mu(\mathbf{T}_{\text{grammar}}(3, k, F)) = f_1 T_{k-1} + f_2 \sum_{p=2}^{k-1} T_{k-p} T_{p-1} + f_3 \sum_{q=2}^{k-2} \left[T_{q-1} \sum_{p=q+1}^{k-1} T_{k-p} T_{p-q} \right] \tag{19}$$

However, there is some subtlety here; several of the nodes included in the grammar do not alter the underlying function mathematically and only act as placeholders for the decoding function P. Such nodes include the ‘expression’ and ‘operation’ nodes. In considering $\mu(\boldsymbol{\Omega}_{\text{grammar}})$, these meta-nodes can be safely collapsed (i.e ignored) in the computation.

Figure 4 is a plot of $\mu(\boldsymbol{\Omega}_{\text{tree}})$, $\mu(\mathbf{T}_{\text{grammar}})$ and $\mu(\boldsymbol{\Omega}_{\text{grammar}})$ with increasing n. The tree-space data are generated with a basis set of operations given by,

$$\begin{aligned} F_{\text{tree}} = \{&\{x, [0, 9]\}, \\ &\{\sin, \cos, \log, \exp\}, \\ &\{+, -, \times, \div\}\} \end{aligned} \tag{20}$$

for the expression trees (indicative of the approach suggested by [10]). The grammar-space data are generated with $m = 3$ and basis set,

$$\begin{aligned} F_{\text{grammar}} = \{&\{x, [0, 9]\}, \\ &\{\sin, \cos, \log, \exp, < \text{expr} >, < \text{digit} >\}, \\ &\{+, -, \times, \div, < \text{func} >\}, \\ &\{< \text{op} >\}\} \end{aligned} \tag{21}$$

for the grammar representation (indicative of the study by Tsoulos et al. in [8]). The meta-operations (in angle brackets) are ignored in the computation of $\mu(\boldsymbol{\Omega}_{\text{grammar}})$ resulting in an identical basis. As can be seen in the Figure 4, the size of these spaces grows extremely quickly with the number of nodes in the tree. The number of trees in grammar-based approaches appears to grow more quickly due to the trinary operation. However it can be seen that the number of expressions representable by these encodings are equivalent.

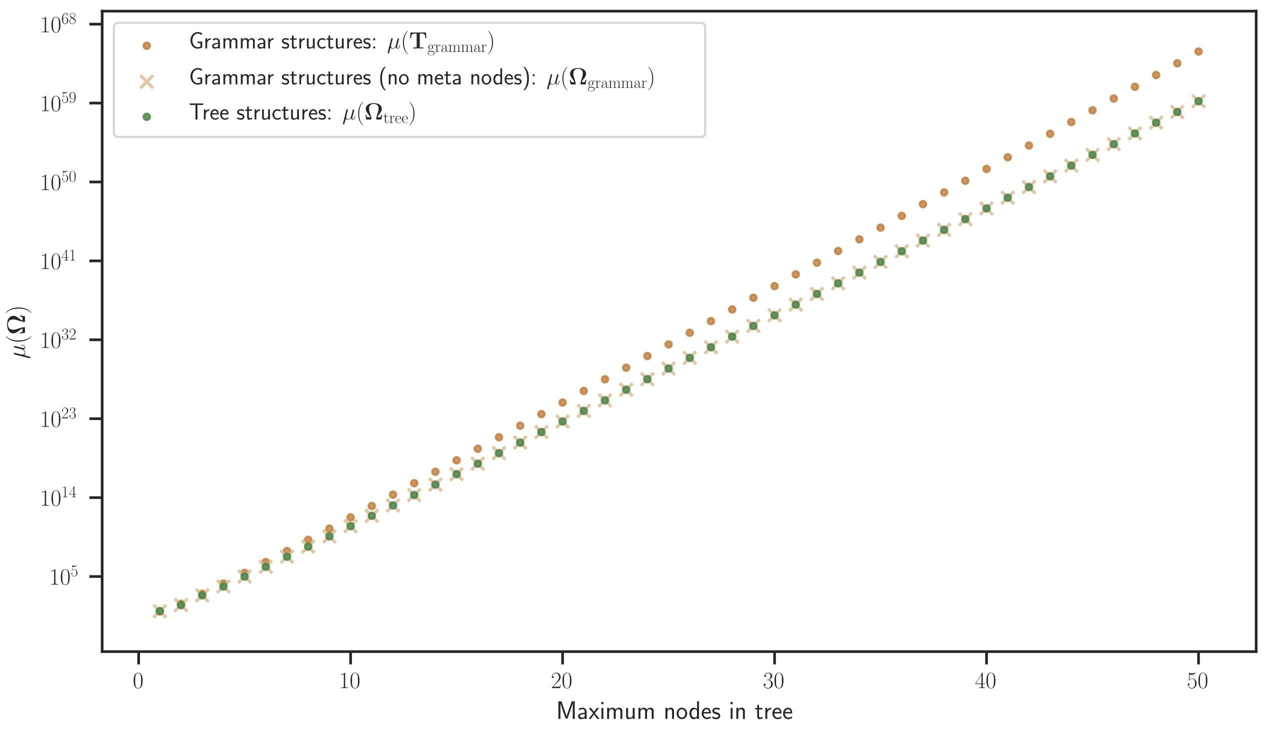


Figure 4: Size of tree space with increasing n.

4 Affine regression trees

A novel encoding representation is proposed here - the *affine symbolic regression tree*. This encoding is essentially an extension of the expression tree with additional structure at each node. The new approach is inspired by two observations. Firstly, the exact discovery of constants in symbolic regression schemes is an open problem with few approaches (there are several notable techniques that specify constants approximately - the reader is directed to [14, 15, 16] for examples). The second observation is that adjacent points in $\mathbf{E}$ are unlikely to be adjacent in $\mathbb{R}$ once projected there via the objective function. Put plainly, it is often not trivial for the SR algorithm to increment towards the true solution despite a high degree of semantic similarity (for example a constant that is wrong by a small integer/rational value).

To this end, the affine symbolic regression tree is defined in the same manner as the expression tree. However each node η is now a 3-tuple,

$$\eta = \{a, f, b\} \tag{22}$$

Where a and b are termed *constants*. During execution, nodes take the affine form,

$$\eta = af + b \tag{23}$$

In this regard the representation bears some similarity to the multiple regression approach in [14] whereby linear combinations of all tree subexpressions are used during a meta-optimisation step. However the current approach differs both in that an affine combination is used and in that the constants are included as a part of the tree structure itself.

Constants have their own structure and are represented by a 4-tuple,

$$\theta = \{s, p, q, r\} \tag{24}$$

Table 1: Constant mutation operations proposed by the current study.

Operation	Action	Description
$m_0(\theta)$	$\theta \to 0$	Zero constant
$m_1(\theta)$	$\theta \to 1$	Unit constant
$m_i(\theta)$	$\theta \to \theta \pm 1$	Increment/decrement
$m_{r1}(\theta)$	$\theta \to \frac{p \pm 1}{q}$	Increment/decrement numerator
$m_{r2}(\theta)$	$\theta \to \frac{p}{q \pm 1}$	Increment/decrement denominator
$m_s(\theta)$	$\theta \to s\theta, s \in S$	Multiply by transcendental constant
$m_r(\theta)$	$\theta \to \theta^r, r \in R$	Raise to exponent
$m_{pm}(\theta)$	$\theta \to -\theta$	flip sign of constant

where p and q are integers in the range $[0, z]$ and $[1, z]$ respectively. r is a member of the set of permitted exponents R with a default value of 1. s is an element from the permitted transcendental constants S, also with a default value of 1. Upon execution constants are evaluated as,

$$\theta = s\left(\frac{p}{q}\right)^r \tag{25}$$

Upon initialisation, 'a' constants are given a value of 1 ($\{1, 1, 1, 1\}$) and 'b' constants a value of zero ($\{0, 1, 1, 1\}$). Initialising the constants in this way provides a bias towards sparse solutions. Care is taken to ensure that $q \neq 0$ so that illegal divisions are avoided by design. A more granular approach is also possible for the specification of the exponents. In this case the constants are represented by the 5-tuple,

$$\theta = \{s, p, q, r_1, r_2\} \tag{26}$$

and are evaluated as

$$\theta = s\left(\frac{p}{q}\right)^{\frac{r_1}{r_2}} \tag{27}$$

However, this increase in granularity comes at the cost of a larger search space and is not the approach considered in this paper.

Constants are mutated during the run of the search by a number of constant mutation operations that are defined in addition to more orthodox tree-based search operations (the reader is directed to [11] for a reference). Several of these constant mutation operations are described in Table 1.

The advantages of this representation are several; consider the expression tree in Figure 5. The affine tree is far more compact requiring only a fraction of the number of nodes to represent the same expression. In fact, with affine trees there is no need to include integers or other constants in the basis set F at all. Constant discovery is handled entirely by the affine constant objects.

The size of the search space can readily be estimated by extending the analysis of expression trees above. The first step is to enumerate the number of possible values an affine constant $\theta \in \mathbf{A}$ can take. This is achieved by considering the elements in the tuple,

$$\mu(\mathbf{A}) = \mu(P)\mu(Q)\mu(R)\mu(S) = z(z+1)\mu(R)\mu(S) \tag{28}$$

Since there are two constants per node the expression for $\mu(\mathbf{\Omega}_{\text{affine}})$ can be written,

$$\mu(\mathbf{\Omega}_{\text{affine}}) = \sum_{k=0}^{n} \left[(z(z+1)\mu(R)\mu(S))^{2k} T_j\right] \tag{29}$$

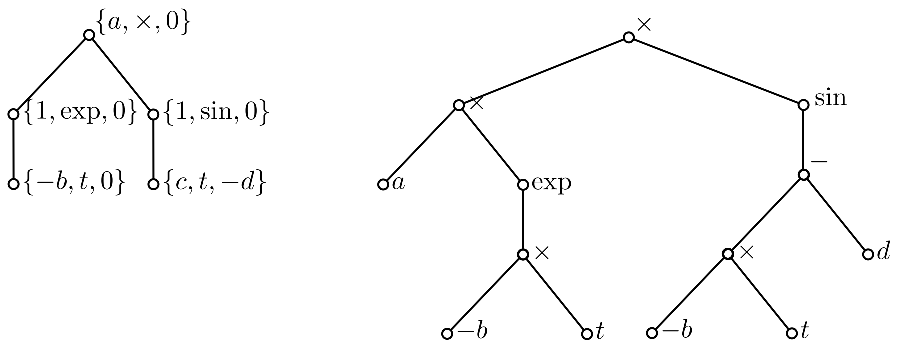


Figure 5: Comparison between compact affine and expression trees for the expression $f = ae^{-bt}\sin(ct - d)$.

or explicitly,

$$\mu(\mathbf{\Omega}_{\text{affine}}) = \sum_{k=0}^{n} \left[(z(z+1)\mu(R)\mu(S))^{2k} \left[f_1 T_{k-1} + f_2 \sum_{p=2}^{k-1} T_{k-p} T_{p+1} \right] \right] \tag{30}$$

Adding this result to the previous data in Figure 4 seems to suggest that this approach is strictly worse (in terms of search space size) than expression trees. However this is a false equivalence. While the affine tree representation is at least as compact as the expression tree representation, in many cases it will be significantly more so. In practice, one is able to select a lower value of n when working with the affine representation and still maintain the same coverage of $\mathbf{C}^r$. Figure 7 compares several scenarios in which the affine representation is more compact and plots them against the number of nodes required for an affine representation. The scenarios shown are illustrated in Figure 6.

The results of Figure 7 indicate that the affine representation has a smaller search space for expressions that require four times as many nodes to represent as an expression tree. For trivial expressions with only a few elementary integer constants, such a ratio is perhaps unrealistic. However, the authors would argue that for more realistic engineering expressions such as the one depicted in Figure 5, a 1:4 ratio is more likely. The authors would note that the tree representations in Figure 5 assume that the constants $a, -b, c, -d$ are specified in the basis. In reality, an expression tree would require further subtrees to represent these values adding yet more complexity.

The advantages of the affine tree representation extend beyond the compactness of the search space. The constant-mutations described in Table 1 permit a pseudo-gradient between expressions of the correct form (only errors in constants) and the true solution. Work is ongoing in developing a formal mathematical argument but a rudimentary explanation is thus. Consider any two affine trees of the same structure α_1 and α_2 (but with different constant values) with objective function values j_1 and j_2 respectively such that $j_2 < j_1$. There must then be a chain of constant mutation operations $[M_1, \ldots, M_v]$ the successive application of which will map α_1 onto α_2. It is furthermore argued that each application of the mutation operations in the chain is more likely to result in a monotonically-decreasing objective score than an equivalent path based on orthodox search operations.

The affine representation has additional advantages in terms of extendibility. As noted in [9], the application of the rules for differentiation tend to result in constants appearing in multiple places in a single expression. A future study examines an extension of the affine tree method such that constants are sampled from a finite pool. The successful outcome of this work would dramatically reduce the size of the required search space.

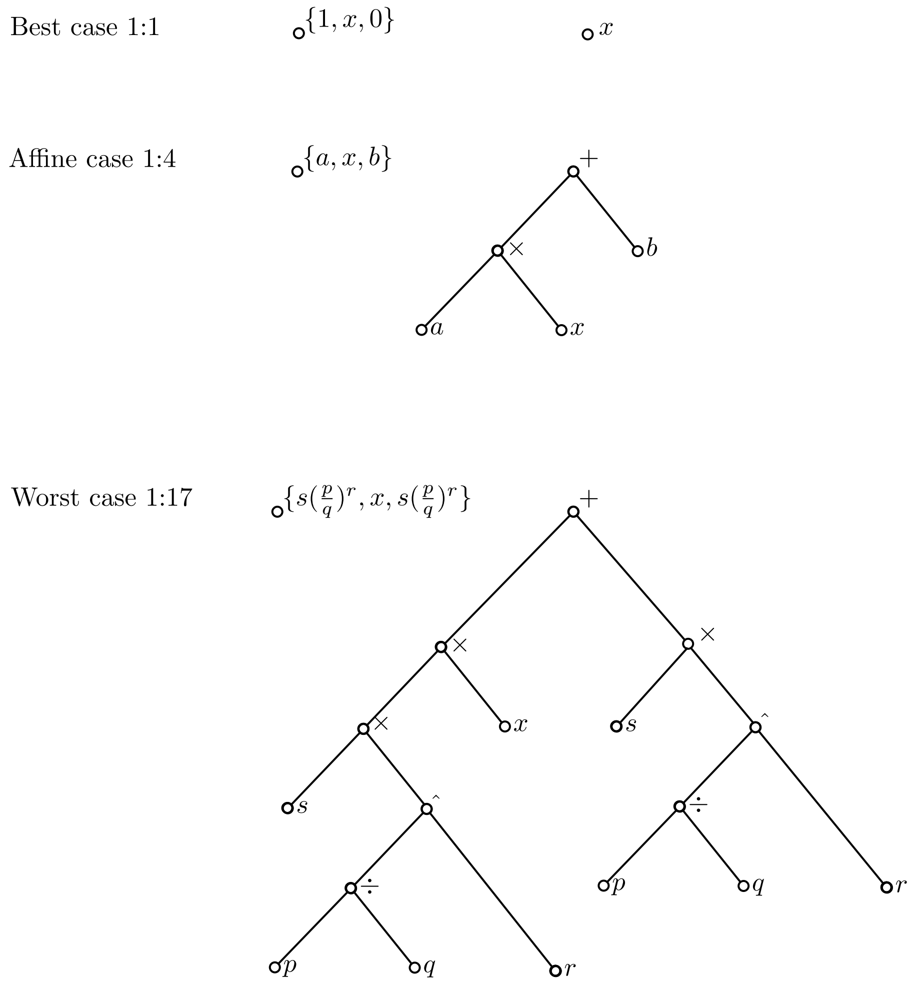


Figure 6: The potential savings offered by an affine structure over a standard expression tree in terms of nodes in the tree structure.

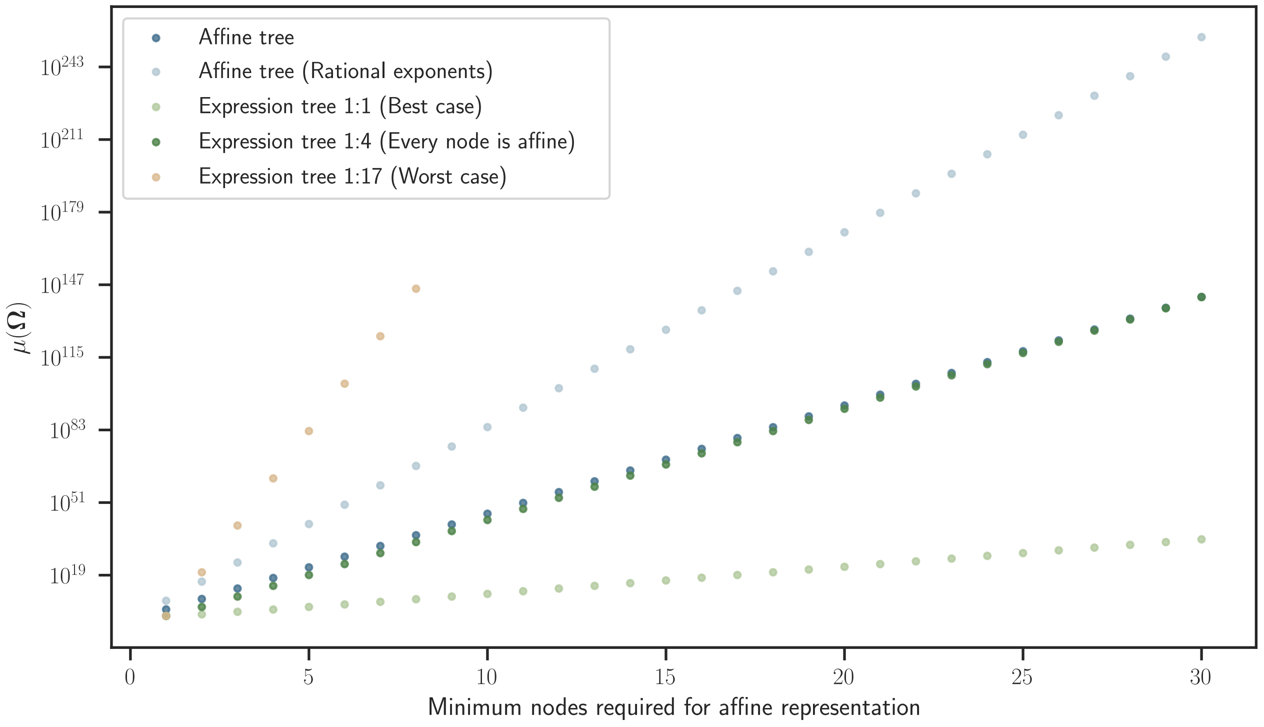


Figure 7: Comparison of affine and expression trees in terms of search space size versus minimum number of nodes required for affine representation.

5 Illustration with a benchmark set

This section illustrates the potential efficacy of the affine tree representation with a simple case study. In order to make comparisons, the benchmark set of ODE problems described in [8] is used. The problem set is outlined in Figure 2.

An affine tree representation with an maximum n value of 10 is compared to a standard regression tree representation with maximum n values of 10 and 40. For all representations, the symbolic regression is undertaken by a genetic algorithm (GA) with properties described in Table 4. An objective function is selected with four terms. The function is defined as the weighted sum of the Mean squared error (MSE) over the ODE, the trial solution f, its first derivative f' and initial or boundary conditions. The target quantities $\hat{f}$ and $\hat{f}'$ are computed in advance of the run by a fixed-step fourth-order Runge-Kutta scheme with a step size determined by the domain of the target problem. For first-order problems, the MSE over f' is not used. The overall objective function is therefore,

Table 2: ODE problems (from [8]) studied in the current investigation.

Problem	ODE	Domain	Subject to	Exact solution
1	$f' - \frac{2t-f}{t} = 0$	$t \in [0.1, 1]$	$f(0.1) = 20.1$	$f = t + \frac{2}{t}$
2	$f' - \frac{1-f\cos(t)}{\sin(t)} = 0$	$t \in [0.1, 1]$	$f(0.1) = \frac{2.1}{\sin(0.1)}$	$f = \frac{t+2}{\sin(t)}$
3	$f' + \frac{1}{5}f - e^{\frac{-t}{5}}\cos(t) = 0$	$t \in [0, 1]$	$f(0) = 0$	$f = e^{\frac{-t}{5}}\sin(t)$
4	$f'' + 100f = 0$	$t \in [0, 1]$	$f(0) = 0, f'(0) = 10$	$f = \sin(10t)$
5	$f'' - 6f' + 9f = 0$	$t \in [0, 1]$	$f(0) = 0, f'(0) = 2$	$f = 2te^{3t}$
6	$f'' + \frac{1}{5}f' + f + \frac{1}{5}e^{\frac{-t}{5}}\cos(t) = 0$	$t \in [0, 2]$	$f(0) = 0, f'(0) = 1$	$f = e^{\frac{-t}{5}}\sin(t)$
7	$f'' + 100f = 0$	$t \in [0, 1]$	$f(0) = \sin(10), f'(0) = 0$	$f = \sin(10t)$
8	$tf'' + (t-1)f' + f = 0$	$t \in [0, 1]$	$f(0) = 1, f(1) = 0$	$f = 1 - t$
9	$f'' + \frac{1}{5}f' + f + \frac{1}{5}e^{\frac{-t}{5}}\cos(t) = 0$	$t \in [0, 1]$	$f(1) = \frac{\sin(0.1)}{e^{\frac{1}{5}}}, f'(0) = 1$	$f = e^{\frac{-t}{5}}\sin(t)$

Table 3: Remaining properties of the search problem in the case study.

Description	Notation	Value
Decoding function	P	'Sympy' Computer algebra system
Functional interpretation	S	$d = 100$ point even sampling over the domain
Objective function	J	See (31)
Search operations	M	Reproduction, Tree crossover, Random tree, Subtree mutation

Table 4: Parameters of the genetic algorithm used in the case study.

Parameter	Value
Population size	400
Maximum generation	500
Initialisation procedure	'grow' [5]
Selection procedure	Tournament ($k = 2$)
Initial maximum tree depth	3
Learning period	5

$$J(f) = \frac{1}{d}\sum_{i}^{d}\left[\lambda_1(\Psi(\hat{f}) - \Psi(f))^2 + \lambda_2(\hat{f}_i - f_i)^2 + \lambda_3(\hat{f}'_i - f'_i)^2 + \right] + \lambda_4\sum_j (I_j(\hat{f}) - I_j(f))^2 \tag{31}$$

where $\Psi(f)$ is the ODE of interest and the I_j are the initial or boundary conditions applicable to that problem. The λ_i are the weights and are all set to unity with the exception of λ_4 which is set to 100. A computer algebra engine (in this case the symbolic library available in the python language *sympy*) is used for the evaluation of derivatives and assessment of exact solutions. The remaining elements of the search problem are described in Table 3.

For both representations, a basis set given by,

$$\begin{aligned} F = \{&\{t,\},\\ &\{\sin, \cos, \log, \exp\},\\ &\{+, -, \times, \div\}\} \end{aligned} \tag{32}$$

is used. For the expression tree representations, the digits $[0, 9]$ are added to f_1. Initialisation, mutation and application of affine constants is as described in the previous section. In order to simplify the search, the sets of exponents R and transcendental constants S are set to $\{1\}$ and the corresponding affine mutations are excluded. The application of affine mutations and search operations is handled by an adaptive process inspired by [17]. The procedure samples search operations randomly according to method probabilities p_i that are initialised as equal but updated every L generations. This interval is referred to as the *learning period.* The probabilities are updated by the rule,

$$p_i = \frac{o_i^2}{s_i} \tag{33}$$

where s_i is the number of times the i^{th} search operation is selected and o_i is the number of successful entrants to the population produced by that method during the last learning period. The raw probabilities are then scaled such that they sum to unity. Adaption of affine mutations is handled separately to search operations. For the affine representation, up to 5 affine mutations are applied during the production of a new individual. The procedure is the following. Firstly, mutations are applied with probability 0.9. If successful, then a number in the range $[1, 5]$ is sampled at random and that many mutation operations are sampled and applied sequentially.

In order to speed up the search, some additional techniques are employed; checks for illegal operations such as infinite terms or division by zero are employed before the evaluation of the objective function. In addition, to restrict the size of the search space, nested transcendental functions (sines, cosines and exponentials) are disallowed. In the case that such a nesting is produced by the action of the search operation, the individual is given a null fitness score and loses tournament selections automatically.

The GA was run 10 times for each problem and representation. In order to compare results, the expected number of function calls required for the solution to be found $E\left[\Lambda^*|\mathbf{\Omega}\right]$ is estimated. Because the GA does not find the exact solution in every run, a right-censored estimator of $E\left[\Lambda^*|\mathbf{\Omega}\right]$ is employed. Assuming an exponential distribution for the underlying probability of finding the solution after any given assessment of the objective function, one retrieves the maximum likelihood estimate of the right-censured mean as,

$$E[\Lambda^*] = \frac{\sum \Lambda}{\sum f^*} \tag{34}$$

which is the sum of all function calls divided by the number of times an exact solution is found. A precision of 400 function evaluations applies to the results, as the exact number of function calls within the final generation is not stored by the current implementation. A 90% confidence interval for the estimates of $E[\Lambda^*|\mathbf{\Omega}]$ is also estimated using a bootstrapping approach with 1000 samples. Although the number of samples is perhaps low for rigorous estimation of confidence intervals, the same approach is used for each case to aid in the visualisation of uncertainty. These data are plotted alongside the results of the individual runs in Figure 8. Uncertainty in the grammar encoding is due to uncertainty regarding the population sizes used in the original study. To ensure the most competitive comparison, the lower bound of these estimates is used for comparison with the proposed approach.

The results show varied performance, the affine method performs better on several problems including ODE1 and ODE8. However there are also several problems for which the affine method was not able to locate the exact solution in any of the 10 runs. Overall, both the affine and expression tree representations perform better than the grammar-based encoding of [8], often dramatically so. However in problems 3, 6 and 9 The tree-based approaches were unable to find the solution with any regularity. It is unlikely to be a coincidence that all three of these problems share the same functional solution, and that this solution has the least compact tree representation of any in the problem set. Investigation of the suitability of the encoding schemes for problems of differing compactness is to be the subject of future work.

The benchmark problems are not without their own limitations. One such limitation is the apparent variation in difficulty of the ODE problems within it. Another is the limited domains over which most of the solutions are monotonic. An interesting avenue for further investigation is the construction of a new benchmarking suite of problems for which the difficulty can be controlled by selecting problems with increasing unguided complexities. Such a benchmark would more readily enable useful comparisons between candidate approaches.

6 Conclusions

In this paper a formal description of the search problem underlying an SR-based approach to the solution of differential equations is presented. This description is used to compare several existing representation schemes for which the sizes of the search spaces are enumerated and compared. A novel expression encoding - the *affine regression tree* is also presented. The search space of this representation is enumerated and found to be at least as small as that of expression trees for problems with an affine structure and considerably more compact than the worst case.

The new encoding is demonstrated in a case study where the optima of interest are the solutions to a benchmark suite of ODE problems. Despite mixed results, the proposed approach shows similar or better performance in a number of problems. However in problems with less compact solutions (such as ODE3, ODE6 and ODE9), both affine and tree-based representations perform poorly. An ongoing body of work seeks to address this and the outstanding issue of exact constant discovery in symbolic regression.

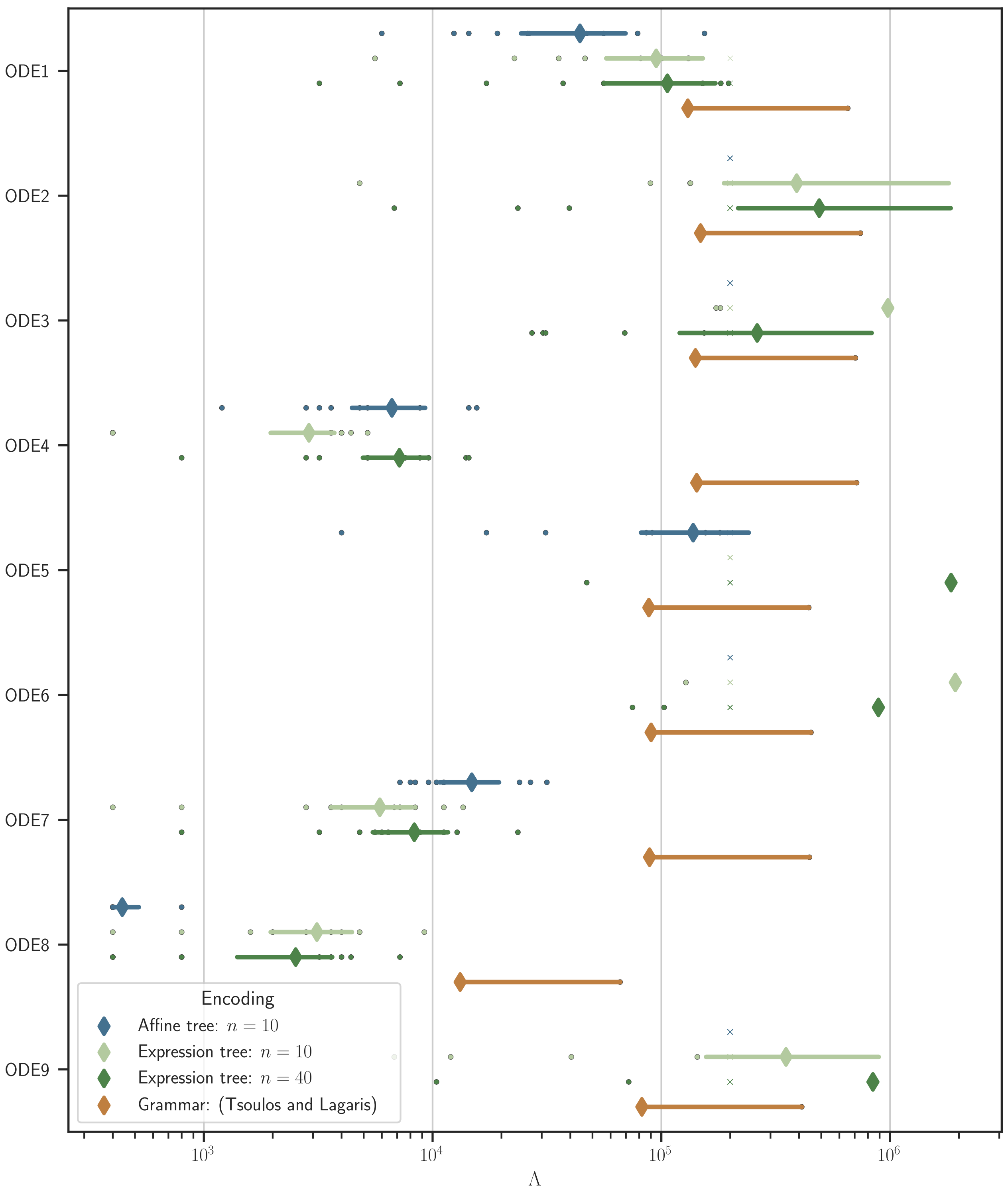


Figure 8: Results of the benchmarking study ODE problem plotted against number of function evaluations before the exact solution is found. Estimates of $E[\Lambda^*|\boldsymbol{\Omega}]$ are displayed as diamonds with estimated 90% confidence intervals. Individual samples of successful runs are plotted as points, unsuccessful runs are plotted as crosses. Results for individual runs [8] are not reported and so these are omitted from the figure.

Acknowledgements

MDC would like to acknowledge the support of the EPSRC grant EP/L016257/1.

References

[1] P. Sundararajan and S.T. Noah. Dynamics of forced nonlinear systems using shooting. *Journal of vibration and acoustics*, 119:9–20, 1997.

[2] R.E. Mickens. Comments on the method of harmonic balance. *Journal of Sound and Vibration*, 94:456–460, 1984.

[3] L. Guillaume and F. Charton. Deep learning for symbolic mathematics. In *Proceedings of the 2019 International Conference on Learning Representations*, 2019.

[4] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

[5] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT press, 1992.

[6] G. Burgess. Finding approximate analytic solutions to differential equations using genetic programming. Technical report, Defense science and technology organisation, Canbera, Australia, 1999.

[7] S.J. Kirstukas, K.M. Bryden, and D.A. Ashlock. A hybrid genetic programming approach for the analytical solution of differential equations. *International Journal of General Systems*, 34:279–299, 2005.

[8] I.G. Tsoulos and I.E. Lagaris. Solving differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 7:33–54, 2006.

[9] T. Seaton, G. Brown, and J.F. Miller. Analytic solutions to differential equations under graph-based genetic programming. In *European Conference on Genetic Programming*, pages 232–243. Springer, 2010.

[10] W.J.A Lobão, D.M. Dias, and M.A.C Pacheco. Genetic programming and automatic differentiation algorithms applied to the solution of ordinary and partial differential equations. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5286–5292. IEEE, 2016.

[11] R. Poli, W.B Langdon, N.F McPhee, and J.R. Koza. *A Field Guide to Genetic Programming.* Lulu Enterprises, 2008.

[12] R.L Graham, D.E. Knuth, O. Patashnik, and S. Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3:106–107, 1989.

[13] L. Mex, C.A. Cruz-Villar, and F. Peñuñuri. Closed-form solutions to differential equations via differential evolution. *Discrete Dynamics in Nature and Society*, 2015, 2015.

[14] I. Arnaldo, K. Krawiec, and U. O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 879–886, 2014.

[15] D.A. Diver. Applications of genetic algorithms to the solution of ordinary differential equations. *Journal of Physics A: Mathematical and General*, 26:3503–3513, 1993.

[16] H. Iba, H. deGaris, and T. Sato. A numerical approach to genetic programming for system identification. *Evolutionary computation*, 3:417–452, 1995.

[17] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1785–1791. IEEE, 2005.