The Inefficiency of Genetic Programming for Symbolic Regression – Extended Version*

Gabriel Kronberger[1][0000−0002−3012−3189], Fabricio Olivetti de Franca[2][0000−0002−2741−8736], Harry Desmond[3][0000−0003−0685−9791], Deaglan J. Bartlett[4][0000−0001−9426−7723], and Lukas Kammerer[1][0000−0001−8236−4294]

[1] University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria `gabriel.kronberger@fh-ooe.at`
[2] Federal University of ABC, Santo Andre, SP, Brazil `folivetti@ufabc.edu.br`
[3] Institute of Cosmology & Gravitation, University of Portsmouth, Dennis Sciama Building, Portsmouth, PO1 3FX, UK
[4] CNRS & Sorbonne Université, Institut d'Astrophysique de Paris (IAP), UMR 7095, 98 bis bd Arago, F-75014 Paris, France

Abstract. We analyse the search behaviour of genetic programming for symbolic regression in practically relevant but limited settings, allowing to exhaustively enumerate all solutions. This enables us to quantify the success probability of finding the best possible expressions, and to compare the search efficiency of genetic programming to random search in the space of semantically unique expressions. This analysis is made possible by improved algorithms for equality saturation, which we use to improve an exhaustive symbolic regression algorithm to produce the set of semantically unique expression structures that is orders of magnitude smaller than the SR search space. We compare the efficiency of random search in the set of unique expressions and genetic programming. For our experiments we use two real-world datasets, where symbolic regression has been used to produce well-fitting univariate expressions. The results show that genetic programming in such limited settings explores only a small fraction of all unique expressions, and evaluates expressions repeatedly that are congruent to already visited expressions.

Keywords: Symbolic Regression · Genetic Programming · Search space.

1 Introduction

Symbolic regression (SR) is a machine learning task where the goal is to find an expression representing a function of the parametric functions family $f(x;\theta)$ that accurately fits a dataset $(x_i, y_i)_{i=1}^n$ with an adjusted value of θ [31,32]. This is commonly performed using search algorithms that explore a search space composed of mathematical expressions.

* This is an extended version of the article submitted to Parallel Problem Solving from Nature (PPSN) Conference 2024. Changes to the submitted version are marked in blue.

Search algorithms [51] can be classified as complete and optimal if they are guaranteed to find a solution and, also, returning the best possible solution. A random search, in which a random solution is sampled at every time step, is both complete and optimal, provided an unlimited runtime. Another complete and optimal search is the enumeration of the whole search space, since the algorithm will traverse all of the possible solutions. This is only feasible if the search space is finite and small enough to process all elements in limited time. On the other hand, a local search exploits the neighbourhood of an initial point and focuses the search on a limited region of the search space. This approach is neither complete nor optimal as it depends on the size of the explored neighbourhood and the starting point. In evolutionary algorithms (EA) [15] a set of solution candidates (here called population) is used in a parallel search, where parts of solution candidates are recombined, mutated, evaluated repeatedly, to gradually evolve improvements. Processes observed in natural evolution, most importantly inheritance of traits from parents to children, and selection pressure, are simulated. Genetic programming (GP) [31] is an evolutionary algorithm and is a popular algorithm for SR. In tree-based GP for SR the search space of genotypes (i.e., expression trees that can be generated up to a given length limit), can be distinguished from the (much smaller) set of distinct expressions after the conversion to a canonical form, as many different expressions represent the same function (e.g., $x\,(x + p_2)$ and $p_1\,x + x^2$). Many modern GP systems use a solver for more efficient parameter optimisation [30,6], which means expressions may contain placeholders $p_1 \ldots p_k$ for real-valued fitting parameters, and reparameterised forms are possible to represent the same function (e.g., $p_1\,(x + p_2)$ and $p_1\,x + p_2'$) .

The solution space is the space of functions that can be produced by fitting the elements from the search space to a given dataset. The elements of the solution space include the best-fit parameter values and the fitting objective value (e.g. log-likelihood for maximum likelihood, or the mean of squared errors for least-squares fitting). As a consequence of local optima when fitting parameters, GP systems may produce different solutions for the same expression.

The efficiency of a search algorithm can be quantified via the probability of reaching a solution of a certain quality by the number of evaluated solution candidates. An efficient search algorithm samples solution candidates with minimal repetition, and invests more time evaluating more promising solution candidates. Ideally, it enumerates the solution candidates following an order from the most promising to the least promising, thus hopefully reaching an acceptable solution quickly. Particularly for SR, this can be challenging since there are many equivalent representations for the same function, either because they evaluate to the same fitness, or because they are isomorphic to other solutions modulo parameter values. Whether this is helpful to traverse SR's search space is open to debate as some authors point out that it can degrade the overall performance, while others show that this may play an important role in reaching a global optimum, as neutral steps may be necessary to reach promising search regions[14,26,27,1].

In this paper, we analyse the efficiency of GP for SR with parameter optimisation [30], whereby we define efficiency as the success probability after a number of evaluated solution candidates. We propose a new method which allows us to calculate explicitly how many of the evaluated solution candidates are unique expressions — or alternatively: how often GP revisits expressions that are isomorphic to previously evaluated expressions. In contrast to previous work, we do not use the fitness values for the detection of semantic duplicates because they depend on the optimised parameter values, and therefore can only detect semantically identical expressions when they are parameterised equivalently. Instead, we use symbolic simplification to a canonical form, which allows us to detect duplicate expression forms regardless of the parameterisation. We use equality saturation [20,56], adapted to simplify and rewrite equivalent expressions into the same form, so we can measure how many isomorphic solutions are visited throughout the search. The analysis is done for a limited search space, for which we fully enumerate all solutions using Exhaustive Symbolic Regression (ESR) [3] for two practically relevant datasets from the physical sciences. In this way we are able to calculate the success probability for finding the best solution or e.g. one of the top 100 solutions with GP.

The results show a worrying inefficiency of GP for SR with a very low rate of unique solutions and a success probability smaller than an idealised random search when limiting the search space to short expressions.

In the following sections we will describe some of the related work (Sec. 2), followed by a brief explanation of our research methods (Sec. 3). In Sec. 4, we analyse the obtained results about the efficiency of SR followed by a brief comment on the limitations of our experiments (Sec. 5) and final discussions (Sec. 6).

2 Related work

The characteristics of the GP search space and the biases of GP when exploring the search space have been extensively studied in earlier work.

In early work, Ebner [14] investigated the redundancy in the solution encoding in GP for SR and argued that, even though the search space of SR is much larger than commonly seen in genetic algorithms, this redundancy is important to guarantee that one of such equivalent solutions is reached. Later, Daida et al. [12,11] asked the question "What makes a GP problem hard?" and studied the consequences of using an iterative growth mechanism for trees in GP. They showed that certain tree structures are much more likely to be visited during the exploration of the search space than others. If a solution lies within the unlikely region of the search, GP has a lower success rate.

Gustafson et al. [24] observed a high frequency of offspring sharing the same fitness values as one of their parents, even with structural differences, chiefly when both parents had a similar fitness value. They suggested to disallow the recombination of two parents with the same fitness value. With this simple mechanism, they observed a significant increase in improved and worsened offspring, and a high decrease of no-change offspring.

Neutrality and redundancy have further been studied by Hu, Banzhaf, and Ochoa [26,27,1] for linear GP for Boolean SR problems with the help of search trajectory networks. They found that more complex phenotypes are harder for evolution to discover as they are represented by fewer genotypes and overrepresented phenotypes are easier to find.

In [45], it was shown that caching fitness values and simplification of solution candidates can significantly reduce the evaluation time for graph-based GP. This observation implies that GP generates a significant number of expressions that can be simplified to the same expression. Similarly, McPhee et al. [41] analysed the possible semantic outcomes of subtree crossover and found that for Boolean GP, most crossover events (over 75 %) produced no immediately useful semantic changes, drawing the attention to the need of investigating new crossover operators for GP.

More recently, Langdon [34] that function evaluation values and opcodes are similar or identical in expanding regions from the root node for SR, large trees, and over thousands of GP generations. Nevertheless fitness continued to evolve even though most crossover events had near zero disruption.

Simplification of SR expressions during the evolution or creation of the expressions have been shown to be beneficial [50,48,7,52] as well as an important mechanism for diversity control [4,5].

Several alternative algorithms for SR have been proposed, which try to improve efficiency by preventing re-evaluations of redundant expressions. This includes algorithms that enumerate a restricted search space, such as [57,28,3] or algorithms that decompose the problem and build the solutions incrementally [17,50]. Others use an evolutionary approach, but limit the search space to find less complex models models [54,19,18,29].

Exhaustive Symbolic Regression [3] (ESR) is an SR algorithm that affords enumeration of the full SR solution space as defined by the function and terminal sets and a length limit. The algorithm as described by [3] has three phases. First it generates all valid expression trees. In the second phase the expression trees are simplified using SymPy, a Python software library for symbolic computation, and only unique expressions are kept. Finally, the parameters of the unique expressions are optimized to fit the expressions to a given dataset. The first two phases only need to be executed once for a given terminal and function set as they are independent of the dataset. Only the last phase of parameter optimization has to be done for each dataset individually.

Compared to GP, ESR guarantees to find the best solution – assuming the optimal parameters are found. It fits and evaluates only simplified expressions to prevent evaluations of structurally different but semantically identical expressions. However, the runtime grows exponentially with the number of variables and the maximum length limit.

3 Methods

3.1 Improved Exhaustive Symbolic Regression

To allow the enumeration of the search space, we limit the set of operators to $\{+, -, \cdot, \div, x^{-1}, \text{powabs}(x)\}$ and the length of expressions, defined as the number of nodes of the expression tree of operators and operands, to maximally $\{10, 12, 20\}$. For the length limits of 10 and 12 nodes, we exhaustively generate the solution space with ESR, the limit of 20 nodes is only used with GP.

Based on the original Python implementation of ESR[5] we have prepared a new implementation in Julia which includes several improvements. In our improved implementation, expressions are generated by repeatedly applying derivation rules from a formal grammar using a breadth-first search procedure. The grammar $G(\mathrm{E})$ used for this work is:

$$\mathrm{E} \rightarrow x \;|\; p \;|\; \mathrm{inv}\,\text{`('}\,\mathrm{E}\,\text{`)'} \;|\; \mathrm{powabs}\,\text{`('}\,\mathrm{E},\ \mathrm{E}\,\text{`)'} \;|\; \text{`('}\,\mathrm{E}\ (+|-)\ \mathrm{E}\,\text{`)'} \;|\; \mathrm{E}\ (\times|\div)\ \mathrm{E}$$

whereby $\mathrm{inv}(a) \equiv a^{-1}$, $\mathrm{powabs}(a, b) \equiv |a|^b$, and p is a placeholder for free parameters.

Breadth-first search starts with a sequence composed of only the root symbol. In each iteration a sequence is taken from the queue, and the first occurrence of the non-terminal symbol E for expressions is replaced with all possible alternatives as defined in the grammar and if possible within the length limit. If the so generated derivations of the sequence contain only terminal symbols, the expression is complete. All other derivations are enqueued again for later processing – after checking for semantic duplicates. Sequences that are semantically equivalent to an earlier visited sequence are detected and removed. This semantic de-duplication is implemented via simplification of expressions to a canonical form using equality graphs (e-graph), which is a data structure that allows compact storage of expressions that are congruent with respect to a given set of rules [44,56]. E-graphs have been originally developed for automated theorem provers, but recently the equality saturation (eq-sat) algorithm that allows to generate all possible expressions from a set of rules has been improved [56] and provided as an easy-to-use Rust library, which has led to impressive efficiency improvements in several applications [56]. In the context of SR, equality saturation (eq-sat) provides a mechanism for simplifying expressions to a canonical form, which opens many new pathways for more detailed study of SR algorithms and their improvement. For example, we have used eq-sat for simplifying solutions produced by GP to remove redundant parameters [20,33].

In this work, we use the implementation for Julia provided in the package *MetaTheory.jl*, and the rule set shown in Table 1. Eq-sat provides functionality for tracking semantic analysis values for equality classes which we use to fold constant expressions and expressions that do not contain variables, i.e. those that contain only parameters and constants and therefore can be simplified to a single parameter.

[5] `https://github.com/DeaglanBartlett/ESR`

After a limited set of eq-sat iterations we extract the expression from the e-graph with the minimum number of parameters, minimum number of tree nodes and is the first expression with respect to a recursively defined order relation for expressions. The extracted expression is the canonical form representative of all congruent expressions as defined by the rule set. We calculate a hash value for this expression and use the hash value for semantic de-duplication which allows us to cut redundant branches of the search tree. The efficiency of eq-sat enables us to generate all unique expressions for the grammar up to a maximum length of 12 within 5 to 10 hours of runtime (single-core) on a desktop PC. We implemented common algebraic rules involving commutativity, associativity, distributivity, and other well known properties of the operator set used in ESR. Table 1 shows the algebraic rules used in this paper to generate unique expressions.

Table 1: Rule set used for equality saturation. Symbols a, b, c, d represent any expression.

$0 + a$	$\rightarrow$	a		$\text{abs}(a)$	$\rightarrow$	a	$a \leq 0$
$0 \times a$	$\rightarrow$	0		$\text{abs}(-1 \times a)$	$\rightarrow$	$\text{abs}(a)$	
$0/a$	$\rightarrow$	0		$\text{abs}(a - b)$	$=$	$\text{abs}(b - a)$	
$1 \times a$	$\rightarrow$	a					
$a - a$	$\rightarrow$	0		$a + b \times a$	$\rightarrow$	$(1 + b) \times a$	
$a + a$	$\rightarrow$	$2\,a$		$b \times a + c \times a$	$=$	$a \times (b + c)$	
a/a	$\rightarrow$	1					
1^a	$\rightarrow$	1		$a + b \times c$	$\rightarrow$	$(a/b + c) \times b$	
a^1	$\rightarrow$	a		$a \times c + b \times d$	$\rightarrow$	$b \times (a/b \times c + d)$	
a^0	$\rightarrow$	1					
0^a	$\rightarrow$	0	$a > 0$	$a^b \times a^c$	$\rightarrow$	a^{b+c}	
				$(a^b)^c$	$\rightarrow$	$a^{b \times c}$	is_integer(c)
$a + b$	$=$	$b + a$		$(a \times b)^c$	$\rightarrow$	$a^c \times b^c$	is_integer(c)
$a \times b$	$=$	$b \times a$		$a^c \times b^c$	$\rightarrow$	$(a\,b)^c$	is_integer(c)
$a + (b + c)$	$=$	$(a + b) + c$		$a \times a$	$=$	a^2	
$a \times (b \times c)$	$=$	$(a \times b) \times c$		$(a^b) \times a$	$\rightarrow$	a^{1+b}	
a/b	$=$	$a \times b^{-1}$					
$(1/a) \times b$	$\rightarrow$	b/a					
$-(-a)$	$\rightarrow$	a					
$-a$	$=$	$(-1) \times a$					
$a - b$	$=$	$a + (-1) \times b$					

We have adapted the implementation of eq-sat as described in [20] to simplify the expressions while minimizing the number of parameters and returning a canonical representation of the expression. By this, we can map isomorphic expressions into a single hash value, helping in the detection of redundant genotypes, similar to [4]. Notice that eq-sat only guarantees the detection if we have a sufficient set of rewriting rules and the e-graphs are saturated. For the purpose

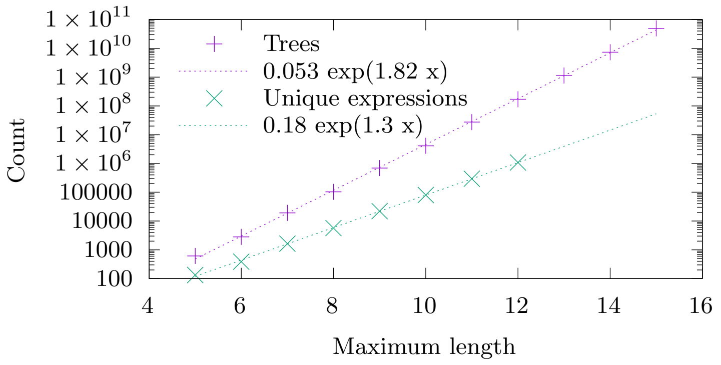


Fig. 1: Growth of the search space and solution space sizes for the used function set. After simplification we found 80 407 unique expressions with maximum length 10, and 1 083 803 unique expressions with maximum length 12. At maximum length 10 there are approximately 50× more trees than unique expressions. This factor grows exponentially with maximum length.

of this work, we stop after a fixed number of eq-sat iterations which allows us to detect most of the duplicates.

Figure 1 shows the exponential growth of the number of expression trees and unique simplified expressions with the expression length limit and the grammar $G(E)$. The gap between the two values also grows exponentially, which implies that the redundancy problem becomes more extreme for longer length limits.

The final step of ESR is fitting all unique expressions to the dataset, which means finding optimal values for all parameter placeholders in the expressions. We use the "shifted limited-memory variable-metric method" with rank-1 update (VAR1) [55] as implemented in *NLOpt*[6] because we found it found solutions with the same quality but was slightly faster than limited-memory BFGS [16] for the RAR dataset described below. ESR does up to 340 random restarts ($\sim$ unif$(-3, 3)$) to have a high chance of finding a global optimum. The optimizer is stopped only after convergence (relative and absolute tolerance $< 10^{-8}$) or when reaching the maximum runtime of five minutes for each restart. Therefore, the final step of parameter optimization requires most of the computational effort of ESR. Parameter optimization took approximately 5 days when distributed to 400 cores on our HPC cluster, for the RAR dataset.

While ESR was designed to be an exhaustive method, we may also iterate through the unique expressions in random order, fit them to the dataset at hand, and stop at any time returning the best solution found so far. We will refer to this algorithm as *random search* (RS) in the following, and use it as a benchmark to quantify the success probability and efficiency of GP. It should be

[6] `https://nlopt.readthedocs.io/en/latest/`

noted, however, that this is an idealized random search in a much smaller search space, as it only visits unique expressions.

3.2 Genetic programming

We adapted the TinyGP [53] implementation by Moshe Sipper, a Koza-style genetic programming system for symbolic regression [31] that is easy to understand and can be easily extended to include parameter optimization for non-Gaussian likelihoods, such as the MNR likelihood described below. TinyGP implements a ramped half-and-half initialization and follows the usual GP process. For a population of size P, P new individuals are generated by: 1) selecting two parents using tournament selection; 2) applying crossover between these parents with probability p_{cx}, otherwise returning the first parent; 3) applying subtree mutation at a random node with probability p_{mut}; 4) replacing the current population with the newly generated solutions; 5) replacing the worst solution in the population with the best of all time (elitism).

In ramped half-and-half, half of the population is generated using the grow method and half with full method. The maximum depth during the initialization is varied between 3 and maximum depth hyperparameter. For tournament selection, n solution candidates are sampled from the population and the fittest solution is chosen. If there are more than one fittest solution, we pick one at random.

After selecting two parents, the algorithm chooses whether to apply crossover with a probability p_{cs}, if it chooses not to, it will simply return the first parent. Otherwise, it will pick a random node from the first parent, and replace with a random subtree from the second parent.

The mutation operator traverses the tree returned by the crossover operator and decides whether to apply the operator at that point with probability p_{mut}. Once it picks a mutation point, it replaces that node with a random subtree generated using the grow method with a maximum depth of 2.

Finally, the current population is replaced by the new generated solutions and the worst generated solution is replaced by the best solution found so far (elitism).

We have adapted the TinyGP code to support univariate functions (inv, powabs), multiple variables, parameter nodes (θ), and parameter optimization using SciPy minimize [7] and the default BFGS optimizer. The partial derivatives of expressions w.r.t. θ (and x for the MNR likelihood) are calculated via forward-mode automatic differentiation (cf. [37]).

Additionally, for the initial population, we ensure that there is no expression that throws an error during evaluation (i.e., Inf, NaN) by resampling the solution until it has a common fitness value. Whenever we generate an expression that is longer than the length limit, we assign a bad value of fitness so it will be discarded to enforce the length limit.

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

Table 2: List of the hyperparameters used in the experiments.

Parameter	Value		
Max. length	10	12	20
Pop. size	100	100	500
Generations	250	250	250
Min. depth	2	2	2
Max. depth	4	4	4
Tournament size	2	2	4
Cx. prob.	1.0	1.0	1.0
Mut. prob.	25 %	25 %	25 %
Objective (Nikuradse)	min. MSE	min. MSE	min. MSE
Objective (RAR)	max. MNR logL	max. MNR logL	max. MNR logL
Optim. iterations (L-BFGS)	10	10	10
Function set	$+, -, \div, \times$, powabs, inv		

The hyperparameter values used in this work are listed in Table 2. We ran three experiments for each dataset varying the maximum length limit whereby we used a larger population size for length 20. We collected the logs with all evaluated expressions from 50 independent runs for both datasets and used these for our analysis.

3.3 Datasets

Flow in rough pipes – Nikuradse We extracted the dataset from Nikuradse on the flow in rough pipes from [46]. Symbolic regression results for this dataset predicting the relationship between turbulent friction λ and scaled roughness have been reported in [23,49]. Reichart et al. [49] stressed that "Over eight decades later, and despite the fundamental and practical importance of the problem" the functional dependency of friction from the Reynolds number and the relative roughness is still unknown. They found that their system called *Bayesian machine scientist* prefers Prandtl's collapse over other models proposed later. We use ESR and GP to find an expression for the collapse $y = f(x)$ using the scaled variables $y = \lambda^{-1/2} - 2 \log \frac{r}{k}$ and $x = \log \frac{v_* k}{\nu})$ as reported in [46].

Radial acceleration relation (RAR) We also explore a real-world dataset from the field of galactic astrophysics. The Radial Acceleration Relation (RAR) describes the link between the acceleration sourced by visible stars and gas ("baryons" in astrophysicists' jargon), g_{bar}, and the total dynamical acceleration as traced by the trajectories of dynamical tracers in galaxies, g_{obs} [36]. The relation is a key, although somewhat contentious, piece of evidence in the debate over the meaning of the "missing mass problem," namely that galaxy dynamics appears to require significantly more mass than is visible (e.g. [39]). In particular, the tightness of the relation (it has small intrinsic scatter, σ_{int}) is often used as evidence that in fact it is not mass that is missing (the "dark matter" of the standard cosmological model), but rather that gravity behaves differently on galactic scales than in the Solar System. Modified Newtonian Dynamics (MOND) accommodates the relation naturally, and in fact predicted it at the theory's inception [43,42].

The RAR has been measured in a range of galaxy types, as well as galaxy groups [21,40,38,8,9,22,47]. We use here the data from the SPARC sample, the flagship dataset for precision analysis of the RAR [35]. This is a compilation of 175 late-type galaxies with resolved HI and Hα rotation curves from the literature and *Spitzer* 3.6 μm photometry to determine bayonic mass distributions. We remove galaxies with poor measurement quality (flag 3) and those with inclination <30 deg, and velocity measurements with $> 10\%$ uncertainty, leaving 2696 points from 147 galaxies. This follows the analysis of [13] (in turn following [36]), who were the first to apply symbolic regression—in particular ESR—to galaxy dynamics, finding many new functions that fit that data better than MOND functions.

We implement one methodological improvement over [13]. That study used a likelihood function that integrated over the true values of the independent variables (g_{bar}) in the dataset using a uniform prior. However, it was shown in [2] that this can lead to significant bias in the best-fit parameter values; these biases are removed by instead marginalising over the true g_{bar} using a Gaussian hyperprior whose mean, μ, and width, ω, are free parameters to be inferred simultaneously (themselves with uniform priors) with those of the function being fitted. In particular, given a function $y = f(x, \boldsymbol{\theta})$ with free parameters $\boldsymbol{\theta}$, the log-likelihood of data $\mathcal{D}$ is

$$\begin{aligned}\log \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}, \mu, \omega, \sigma_{\text{int}}) = &-\frac{1}{2}\sum_i \frac{\omega^2\left(\mathcal{A}_i x_i + \mathcal{B}_i - y_i\right)^2 + \sigma_{x_i}^2\left(\mathcal{A}_i\mu + \mathcal{B}_i - y_i\right)^2}{\mathcal{A}_i^2\omega^2\sigma_{x_i}^2 + \left(\sigma_{y_i}^2 + \sigma_{\text{int}}^2\right)\left(\omega^2 + \sigma_{x_i}^2\right)} \\ &-\frac{1}{2}\sum_i \frac{\left(\sigma_{y_i}^2 + \sigma_{\text{int}}^2\right)\left(x_i - \mu\right)^2}{\mathcal{A}_i^2\omega^2\sigma_{x_i}^2 + \left(\sigma_{y_i}^2 + \sigma_{\text{int}}^2\right)\left(\omega^2 + \sigma_{x_i}^2\right)} \\ &-\frac{1}{2}\sum_i \log\left(\mathcal{A}_i^2\omega^2\sigma_{x_i}^2 + \left(\sigma_{y_i}^2 + \sigma_{\text{int}}^2\right)\left(\omega^2 + \sigma_{x_i}^2\right)\right) + \text{const.},\end{aligned} \tag{1}$$

where

$$\mathcal{A}_i \equiv \frac{\partial}{\partial x} f(x, \boldsymbol{\theta})\bigg|_{x=x_i}, \quad \mathcal{B}_i \equiv f(x_i, \boldsymbol{\theta}) - \mathcal{A}_i x_i. \tag{2}$$

i indexes the data points and σ_{x_i} and σ_{y_i} are the uncertainties on the x and y variables. For RAR, x is g_{bar} and y is g_{obs}. We have verified separately to [2] that MNR leads to unbiased parameter recovery on mock RAR-like datasets, while the uniform prior method used in [13] does not. We employ MNR here by maximising Eq. 1 simultaneously in $\boldsymbol{\theta}, \mu, \omega$ and σ_{int}.

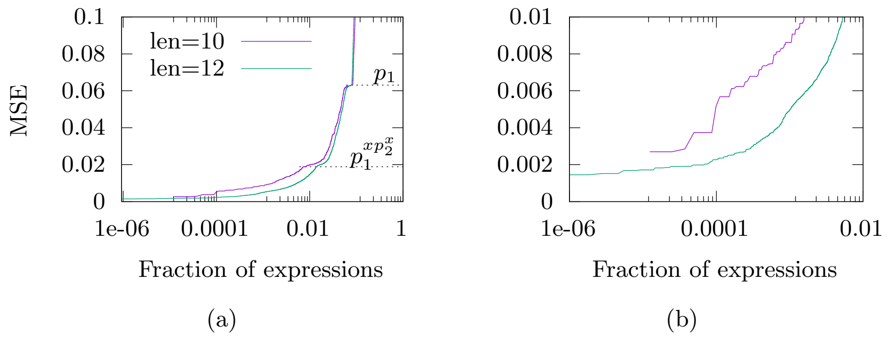


Fig. 2: Distribution of MSE values for all possible expressions with the Nikuradse dataset (a) and a zoomed region (b). The constant model p_1 has an MSE $= 0.063$ and only 10 % of the solutions have a better MSE. The expression $p_1^{x p_2^x}$ reaches MSE=0.019 which only around 1 % of the expressions surpass. The subplot on the right hand side shows that MSE less than 0.002 is reached only by about the 100 best expressions with length 12.

4 Results

4.1 Characterization of the solution space

Figures 2a and 2b show the distribution of MSE values for the Nikuradse dataset for the solution spaces with unique expressions with maximum length 10 and 12. The best expression for length 10 has MSE=$2.7 \cdot 10^{-3}$ and the best expression for length 12 has MSE=$1.46 \cdot 10^{-3}$. We can see from this plot that just about 0.01 % of all expressions have an MSE close to the minimum, a majority of the expressions have an MSE about 10× higher. It is also worth noticing that only 10 % of the expressions are better than the baseline model p_1 and only 1 % are better than $p_1^{x\,p_2^x}$.

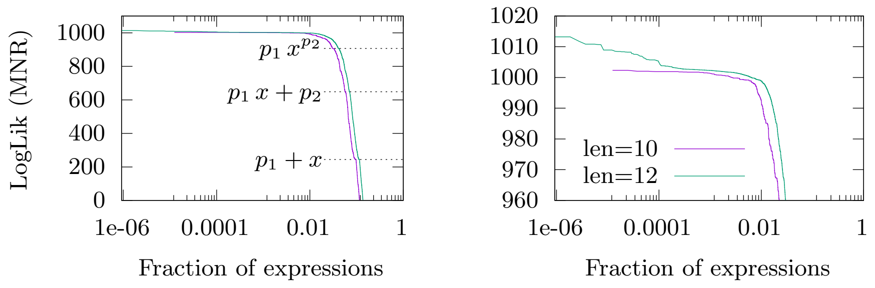
We can see the top-5 expressions for size 10 and 12 in Table 3. The top expressions share some similarities in their form as the adjustable parameters can compensate for any smaller differences. The list shows semantic duplicates with equivalent likelihood values that have not been detected as a consequence of the limited number of eq-sat iterations or missing rules.

Figures 3a and 3b show the distribution of MNR log-likelihood values for the RAR dataset for the unique expressions with maximum length 10 and 12. The distribution shows a large subset of solutions (approximately 1 % of all solutions) with a log-likelihood around 1000. The best solution for length 10 has log-likelihood 1002.34, the best for length 12 has log-likelihood 1013.24.

The Top-5 solutions for the RAR dataset are listed in Table 4. For the top expressions the MNR likelihood parameters are consistently estimated as $\sigma_{\text{int}} \approx 0.08, \mu \approx -0.50, \omega \approx 0.74$.

Table 3: Top-5 expressions of size 12 and 10 for the Nikuradse dataset

Max. len.	Expression	LogLik	Parameters ($[p_1 \ldots p_5]$)	MSE
12	$p_1/(1.0/(p_2 - x) - {p_3}^{{p_4}^x})$	668.65	$[-2.222, -0.253, 3.433 \cdot 10^{-5}, 0.137]$	$1.456 \cdot 10^{-3}$
12	$p_1/(1.0/(p_2 + x) + {p_3}^{{p_4}^x})$	668.65	$[2.222, 0.253, -3.433 \cdot 10^{-5}, -0.137]$	$1.456 \cdot 10^{-3}$
12	$p_1/(\lvert x\rvert^{-x} - {p_2}^{{p_3}^x})$	661.19	$[-1.928, -4.043, 0.463]$	$1.517 \cdot 10^{-3}$
12	$p_1/({p_2}^{{p_3}^x} - \lvert 1.0/x\rvert^x)$	661.19	$[1.928, 4.043, 0.463]$	$1.517 \cdot 10^{-3}$
12	$p_1/(\lvert 1.0/x\rvert^x - {p_2}^{{p_3}^x})$	661.20	$[-1.928, -4.043, -0.463]$	$1.517 \cdot 10^{-3}$
10	$p_1/(1.0/(p_2 + x) - {p_3}^x)$	556.94	$[0.301, 0.673, -0.453]$	$2.699 \cdot 10^{-3}$
10	$p_1/(1.0/(p_2 - x) + {p_3}^x)$	556.94	$[-0.301, -0.673, 0.453]$	$2.699 \cdot 10^{-3}$
10	$1.0/(p_1 + \lvert p_2 + {p_3}^x\rvert^{p_4})$	547.04	$[0.456, -0.191, -0.177, 1.216]$	$2.851 \cdot 10^{-3}$
10	$p_1 - \lvert p_2 + -1.0/(p_3 + x)\rvert^{p_4}$	497.93	$[2.262, 0.773, 0.301, 0.677]$	$3.739 \cdot 10^{-3}$
10	$p_1 - \lvert p_2 + 1.0/(p_3 + x)\rvert^{p_4}$	497.93	$[2.262, -0.7735, 0.301, 0.6767]$	$3.739 \cdot 10^{-3}$



(a) (b)

Fig. 3: Distribution of log-likelihood values for all possible expressions for the RAR dataset (a) and a zoomed region (b). Around 10 % of all solutions reach a good log-likelihood ≈ 1000. The zoomed plot shows that approximately only the 100 best solutions with length 12 have a log-likelihood above 1005.

Table 4: Top expression for the RAR dataset

Max. len.	Expression	NegLogLik	Parameters ($[p_1 \ldots p_4, \sigma_{\text{int}}, \mu, \omega]$
12	$p_1\, x^{-1.0/(p_2 + \lvert p_3 + x\rvert^{p_4})}$	-1013.24	$[-1.73, -2.41, -0.0179, 0.0513, 0.0792, -0.502, 0.74]$
12	$p_1\, x^{1.0/(p_2 - \lvert p_3 + x\rvert^{p_4})}$	-1013.24	$[-1.73, 2.41, -0.0179, 0.0513, -0.0792, -0.502, 0.74]$
12	$1.0/({p_1}^{\lvert p_2 + x\rvert^{p_3}} - x^{p_4})$	-1011.76	$[0.407, -0.0175, 0.221, -0.502, 0.0796, -0.502, 0.74]$
12	$p_1 + \lvert p_2 + -1.0/(p_3 - x)\rvert^x - x$	-1010.84	$[-1.025, -0.775, -1.272, -0.0791, -0.503, 0.74]$
12	$p_1 + \lvert p_2 + 1.0/(p_3 - x)\rvert^x - x$	-1010.84	$[-1.025, 0.775, -1.272, -0.0791, -0.503, 0.74]$
10	$1.0/(p_1 + \lvert p_2 + x^{p_3}\rvert^{p_4})$	-1002.34	$[0.018, -0.276, -0.297, 1.752, 0.0806, -0.503, 0.74]$
10	$1.0/(p_1 - \lvert p_2 + x^{p_3}\rvert^{p_4})$	-1002.34	$[-0.018, -0.276, 0.297, 1.752, 0.0806, -0.503, 0.74]$
10	${p_1}^{p_2 - x^{p_3}} - x$	-1002.06	$[1.39 \cdot 10^{-5}, 0.911, 0.0723, 0.0806, -0.503, 0.74]$
10	${p_1}^{p_2 - (1.0/x)^{p_3}} - x$	-1002.06	$[-1.39 \cdot 10^{-5}, 0.911, -0.0722, 0.0806, -0.503, 0.74]$
10	$x - {p_1}^{p_2 - x^{p_3}}$	-1002.06	$[1.39 \cdot 10^{-5}, 0.911, 0.0722, 0.0806, -0.503, 0.74]$

4.2 Quality of GP solutions

Table 5 shows the results obtained by GP with the different maximum length limits for each dataset. We report the objective values of the best solution found over 50 independent runs, as well as the average and standard deviation of the objective values of each of the 50 solutions. For both datasets and both length limits, GP was not able to find the global optimum. Only when allowing longer expressions, GP was capable of finding a better (but longer) solution for the Nikuradse dataset. When looking at the average objective, we can see that only when we set a larger length limit we find solutions with similar objective values are closer to the best solutions in the more restricted search spaces.

Table 5: Objective values of the best solutions found by GP compared to the global optimum in the search space found by ESR. Smaller values are better. For the Nikuradse dataset the MSE is reported, for the RAR dataset the negative MNR log-likelihood. For the Nikuradse dataset the results for Operon are similar to the results achieved with TinyGP. Neither TinyGP nor Operon find the global optimum

Dataset	Max. length	Optimum	Best	Mean	StdDev
Nikuradse	10	$2.70 \cdot 10^{-3}$	$4.80 \cdot 10^{-3}$	$7.56 \cdot 10^{-3}$	$2.97 \cdot 10^{-3}$
Nikuradse	12	$1.46 \cdot 10^{-3}$	$2.01 \cdot 10^{-3}$	$5.39 \cdot 10^{-3}$	$1.49 \cdot 10^{-3}$
Nikuradse	20		$1.30 \cdot 10^{-3}$	$1.33 \cdot 10^{-3}$	$0.03 \cdot 10^{-3}$
RAR	10	−1002.34	−1000.66	−999.23	0.225
RAR	12	−1013.24	−1001.65	−999.44	0.630
RAR	20		−1007.08	−1002.31	0.934
Nikuradse (Operon)	10	$2.70 \cdot 10^{-3}$	$2.86 \cdot 10^{-3}$	$8.57 \cdot 10^{-3}$	$2.86 \cdot 10^{-3}$
Nikuradse (Operon)	12	$1.46 \cdot 10^{-3}$	$1.68 \cdot 10^{-3}$	$2.51 \cdot 10^{-3}$	$0.58 \cdot 10^{-3}$
Nikuradse (Operon)	20		$1.24 \cdot 10^{-3}$	$1.28 \cdot 10^{-3}$	$0.02 \cdot 10^{-3}$

4.3 Success probability

For a better insight into the reasons why GP does not find the global optimum for both datasets, we analyse the empirical cumulative density function (ECDF) for the number of visited expressions until finding a solution (cf. [25]). To compare the efficiency of search algorithms we plot the ECDF and show multiple curves for different objective value thresholds and search algorithms in a single plot. The reported success probability is calculated as the fraction of the 50 independent runs, where the threshold is reached and the x-axis shows the number of expressions visited until the value was reached in the runs.

We use ESR-based RS as an (idealized) benchmark that ignores the effort for parameter optimization and the problem of local optima, and allows us to focus on the search efficiency of expression trees.

In Fig. 4 we can see the ECDF-curves for multiple MSE thresholds for the Nikuradse dataset. Perhaps surprisingly, RS has a higher success probability and needs to visit fewer expressions to find solutions for all thresholds. However, it is clear that our RS always finds the best solution in the finite solution space as it does not resample solutions. The first threshold of 0.02 is deliberately high,

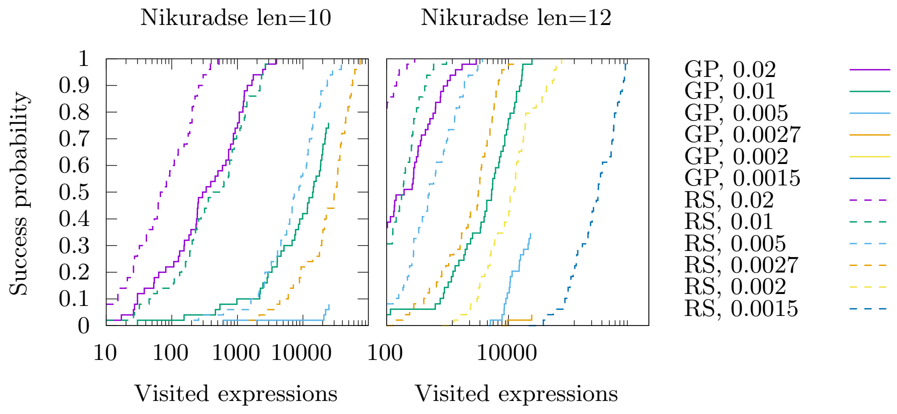


Fig. 4: Success probability of GP and RS over number of visited expressions for length=10 and length=12 for the Nikuradse dataset. For length=10, GP has a high probability to find solutions with MSE below 0.2 and 0.1, but success rate drops below 10 % for a threshold of 0.005. For length=12, the success rates are higher but GP did not find the best solutions in any of the 50 runs.

as we know that approximately 10 % of the solutions have a better MSE. For length= 10, GP requires approximately 600 visited expressions to find a solution in 50 % of the runs and found a solution in all runs after visiting 4000 expressions. RS, finds solutions in 50 % of the runs after visiting 100 expressions and in all runs after fewer than 1000 expressions. For lower thresholds the algorithms require more time to find solutions and success probability of GP decreases as we have limited the GP runs to visit a maximum of 25000 expression (population size: 100 with 250 generations). GP finds no solutions below the thresholds 0.002 and 0.0015. This pattern is repeated for maximum length 12.

Instead of showing the success probability over the number of visited expressions one could also make a more realistic comparison that incorporates the effort spent for parameter optimization in ESR by showing the success rate over the number of function evaluations. In Figure 5 it is clear that from this point of view GP is much faster as it does only a single run of BFGS with 10 iterations for each of the visited expressions and we did a large number of restarts and ran to convergence for ESR. We deliberately ignore the effort of parameter optimization since we focus on the efficiency of sampling expression structures in this work.

Fig. 6 shows the ECDF for RS and GP for the RAR dataset with a similar result. GP does not find the best solutions, and has to visit more expressions than RS until it finds solutions. The success rate of reaching a solution with neg. log-likelihood (nll) below -1000 is only approximately 15 % even for maximum length 12.

With these results, it remains the question of whether expanding the search space of GP enables it to find expressions with the same MSE as RS and with

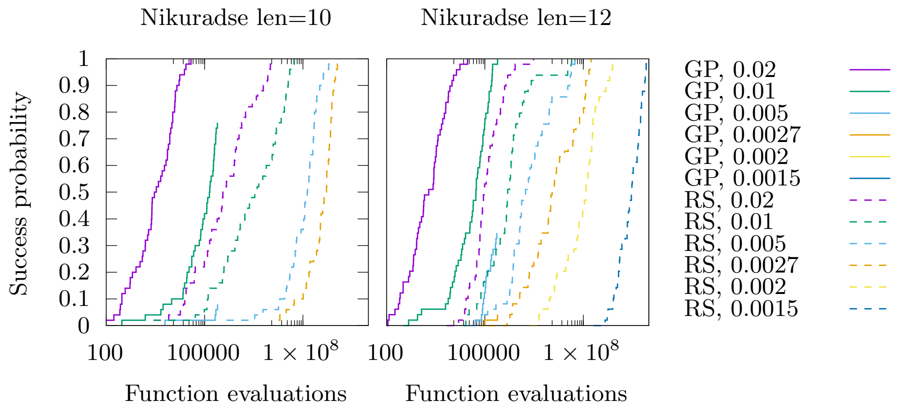


Fig. 5: Success probability of GP and RS over the number of function evaluations for length=10 and length=12 for the Nikuradse dataset. The success probabilities are the same as in Figure 4 but GP is more efficient than RS when counting the number of function evaluations.

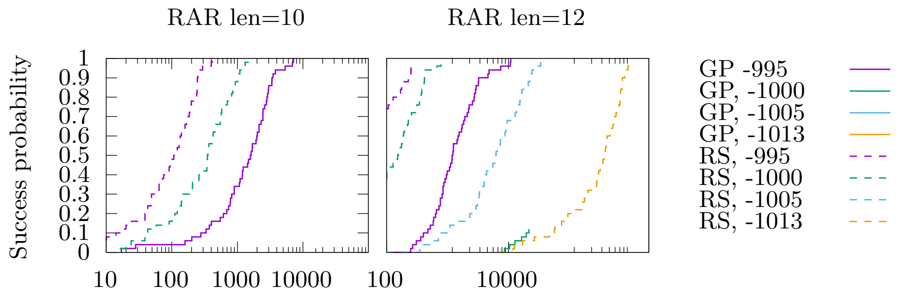


Fig. 6: Success probability over number of visited expressions for RAR for length=10 and length=12. For length=10, GP finds solutions with nll below -995 easily but found a solution with nll < -1000 in only a single run out of 50. For length=12, GP has a success rate of ≈ 15 % for nll < -1000, but fails to find one of the ≈ 100 best solutions with nll < -1005.

the same length. Fig. 10 compares the success rate of RS with maximum length 12 with the success rate of GP with maximum length 20 (but limited to solutions of the same length as RS), surprisingly RS is still more efficient, being capable of reaching a perfect success rate for MSE smaller than 0.002, where GP success rate is capped 50%.

4.4 Semantic duplicates

The previous plots highlight an issue with the power of GP when compared to RS. One reason is wasteful exploration of repeated expressions. Notice that GP

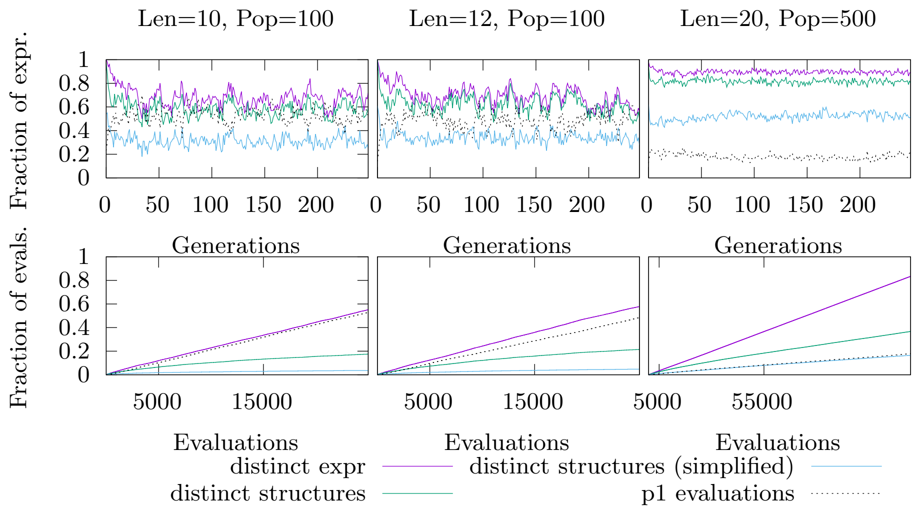


Fig. 7: Number of expressions, and distinct expression structures visited by a single GP run for the Nikuradse dataset relative to population size (top row) and number of total evaluations (bottom row).

can visit repeated expressions in two occasions: they may be regenerated through crossover and mutation, by chance; or GP can generate a new expression that is structurally different to all previously visited expressions but equivalent after simplification. The same simplification procedure based on eq-sat that we used for ESR to produce the set of semantically distinct expressions can be used to analyse how many of the expressions visited by GP are duplicates.

Fig. 7 shows the number of distinct expressions visited throughout the search for the three length limits. The top row of the plot shows the number of distinct expressions, distinct expression structures (ignoring parameter values), distinct expression structures after simplification, and expressions that can be simplified to the trivial expression p_1 (a constant model) for each generation. The frequencies are given as fractions of the population size. The plots in the bottom row show the aggregates over the whole run, where distinctness is determined over the whole run. These plots reveal a worrying behaviour of GP when restricted to such extreme length limits. Without applying simplification, GP generates between 60 to 80 % of distinct expressions (20 % of the expressions or expression structures are the same). When we simplify such expressions, this range goes down to only 20 to 40 %. This behaviour is similar for maximum lengths 10 and 12. For maximum length 20, the search is a little less wasteful, with distinct expressions after simplification in the range of 40% and 60%. Even more alarming is that around 50 % of the expressions can be simplified to a constant (p_1) for length=10 and length=12, and for length=20 this is still 10 to 20 %. The accumulated counts in the bottom row show that only 10 to 20 % of all expres-

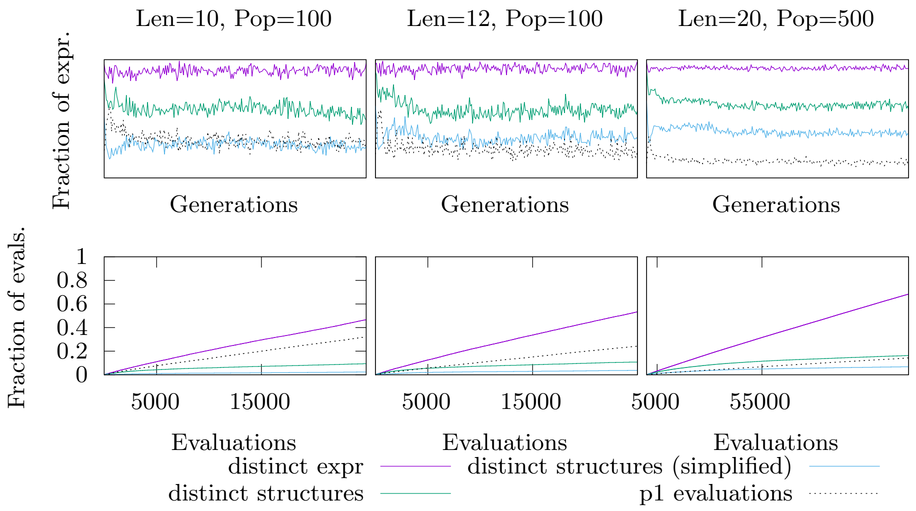


Fig. 8: Number of expressions, and distinct expression structures visited by Operon for Nikuradse relative to population size (top row) and number of total evaluations (bottom row).

sions that GP visits are unique. Notice that the expression p_1 can be generated in many forms, such as $p_1 + p_2$ or $p_1/p_2^{p_3}$. The high percentage of such trivial expression being sampled by GP is related to its position in the search space in which it has a reasonable MSE and many different expressions with MSE worse than this one.

To demonstrate that the issue not only affects our TinyGP implementation but persists even in more powerful GP implementations, we run the same experiment with Operon. However, as Operon does not support the MNR likelihood, we could only test the Nikuradse dataset and optimize the MSE. Figure 8 shows the results for Operon which are qualitatively similar to the TinyGP results.

We observe a similar behavior for RAR dataset as depicted in Fig. 9, with the main difference that the number of distinct expressions and expression structures visited by GP is higher (80 to 100 %), which could be a consequence of the fact that a large proportion of all expressions have a likelihood value close to the best one. The number of semantically distinct expressions after simplification is however again much lower and around 50 %, the main contributor to this are again the expressions that can be simplified to a constant (p_1), which account for ≈ 20 % of the expressions. The aggregated counts of distinct expressions are similarly low as for Nikuradse. Only around 10 % of the evaluated expressions were unique and 90 % were re-visits.

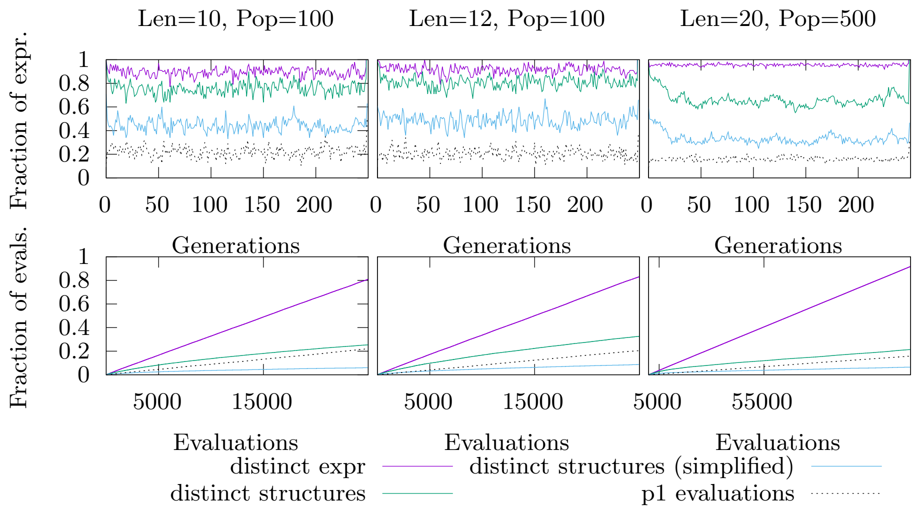


Fig. 9: Number of expressions and distinct expression structures visited by a single GP run for RAR relative to population size (top row) and number of total evaluations (bottom row).

5 Limitations

We should stress that these experiments are limited to the investigation of visited expressions without concerning about the lineage of the generated expressions (i.e., the exploration path traversed to reach the optimal expression). We focused on a broader view of the efficiency of GP as a search algorithm for SR comparing with the enumerated search space and an idealistic random search, if it could guarantee sampling only without repetition.

We used an adapted version of TinyGP which is much simpler than state-of-the-art GP systems such as Operon [6] or PySR [10]. To make sure that the results are not an artifact of TinyGP, we have executed the same experiments for Nikuradse dataset using Operon and have found similar results[8]. The number of unique expressions visited by Operon is similarly low.

We restricted the GP expressions to short length limits. This was necessary to have the same search space for ESR and GP. While we did quantify the efficiency of GP for a larger length limit of 20 nodes, we cannot calculate the success probability for finding the best solutions in the larger search spaces.

GP may require larger length limits to find the best solutions, which can then be simplified. We analysed the GP runs with length limit 20, and plotted the ECDF for the number of visited expressions required until finding solutions which have length smaller or equal to 12 nodes after simplification. As shown in Figure 10 both TinyGP and Operon are then capable of finding expressions

[8] Details can be found in the supplement

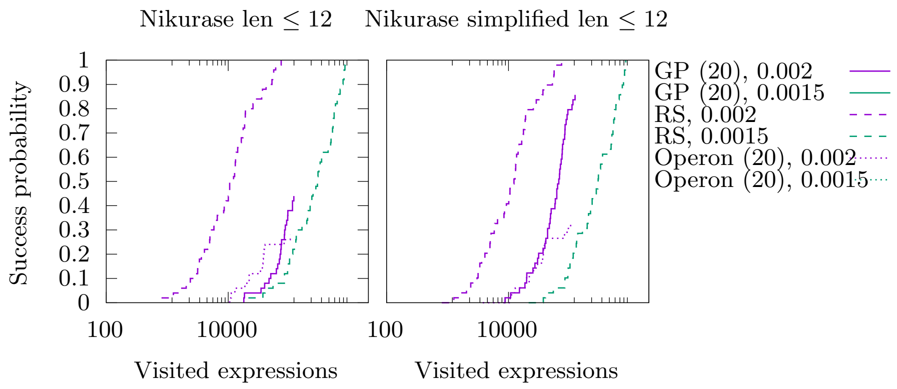


Fig. 10: Success probability over number of visited expressions for RS within the search space for length=12 compared to GP with larger length limit (=20). Here success is defined as finding a solution with MSE lower than the threshold *and* length less or equal 12.

with MSE less than 0.002 with a high success rate (compare to 4, but still fail to find the global optimum. Both GP systems produce similar results. We leave a more detailed study of this effect for future work.

6 Discussion and conclusions

We investigated the search behaviour of genetic programming for symbolic regression by applying a simple implementation called TinyGP to two real-world datasets from the physical sciences. We focused on symbolic regression of functions of a single variable with short length limits (10 – 20 nodes). For this search space we were able to enumerate the space of all semantically unique expressions using an improved implementation of the Exhaustive Symbolic Regression algorithm to verify how much of this space is explored by GP and the extent to which this exploration is efficient.

Having the enumerated solution space, we could first verify that in both datasets a large portion of the expressions are of low quality and only 0.01 to 1 % (depending on the dataset) are within a reasonable distance from the optimum in the fitness space. Using 50 independent restarts of GP we did not find the optimal solutions for any of the search spaces. Comparing the success rate of GP with an idealistic random search, we found that GP visits an order of magnitude more expressions to find solutions with the same success rate.

Finally, we highlighted that a significant portion of the expressions visited by GP are semantically equivalent to previously visited expressions. In our work we determine semantic equivalence of expressions by simplification to a canonical form using equality saturation. This allows us to detect isomorphic expressions even when parameter values are different. This is an important novelty compared to existing SR literature where semantic equivalence is detected only through shared fitness values. We used equality saturation for simplification of expressions. From our results, it remains to be seen whether the observed amount of re-evaluations is in any way helpful for GP, or if we can make the search more efficient by preventing these. Even though the redundancy of representations grows exponentially with larger search spaces, it is not known whether the same proportion of re-visited expressions also occurs when GP is used with more typical length limits of up to 100 nodes, or with more input variables. We leave this topic for future research.

Acknowledgements

We thank Pedro Ferreira for enabling this collaboration and his ideas for improving this work and Bogdan Burlacu for his help when making changes to Operon. Gabriel Kronberger is supported by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology, the Federal Ministry for Labour and Economy, and the regional government of Upper Austria within the COMET project ProMetHeus (904919) supported by the Austrian Research Promotion Agency (FFG). Fabricio Olivetti de Franca is supported by CNPq through the grant 301596/2022-0. Harry Desmond is supported by a Royal Society University Research Fellowship (grant no. 211046). Deaglan J. Bartlett is supported by the Simons Collaboration on "Learning the Universe."

References

1. Banzhaf, W., Hu, T., Ochoa, G.: How the combinatorics of neutral spaces leads genetic programming to discover simple solutions. In: Genetic Programming Theory and Practice XX, pp. 65–86. Springer (2024)
2. Bartlett, D.J., Desmond, H.: Marginalised Normal Regression: Unbiased curve fitting in the presence of x-errors. The Open Journal of Astrophysics **6**, 42 (Nov 2023). https://doi.org/10.21105/astro.2309.00948
3. Bartlett, D.J., Desmond, H., Ferreira, P.G.: Exhaustive symbolic regression. IEEE Transactions on Evolutionary Computation (2023)
4. Burlacu, B., Affenzeller, M., Kronberger, G., Kommenda, M.: Online diversity control in symbolic regression via a fast hash-based tree similarity measure. In: 2019 IEEE congress on evolutionary computation (CEC). pp. 2175–2182. IEEE (2019)
5. 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: 17th International Conference, Las Palmas de Gran Canaria, Spain, February 17–22, 2019, Revised Selected Papers, Part I 17. pp. 361–369. Springer (2020)
6. Burlacu, B., Kronberger, G., Kommenda, M.: Operon C++ an efficient genetic programming framework for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. pp. 1562–1570 (2020)

7. Cao, L., Zheng, Z., Ding, C., Cai, J., Jiang, M.: Genetic programming symbolic regression with simplification-pruning operator for solving differential equations. In: International Conference on Neural Information Processing. pp. 287–298. Springer (2023)
8. Chae, K.H., Bernardi, M., Domínguez Sánchez, H., Sheth, R.K.: On the Presence of a Universal Acceleration Scale in Elliptical Galaxies. Astrophysical Journal, Letters **903**(2), L31 (Nov 2020). https://doi.org/10.3847/2041-8213/abc2d3
9. Chae, K.H., Bernardi, M., Sheth, R.K., Gong, I.T.: Radial Acceleration Relation between Baryons and Dark or Phantom Matter in the Supercritical Acceleration Regime of Nearly Spherical Galaxies. Astrophysical Journal **877**(1), 18 (May 2019). https://doi.org/10.3847/1538-4357/ab18f8
10. Cranmer, M.: Interpretable machine learning for science with pysr and symbolicregression.jl (2023). https://doi.org/10.48550/ARXIV.2305.01582
11. Daida, J.M., Hilss, A.M.: Identifying structural mechanisms in standard genetic programming. In: Genetic and Evolutionary Computation Conference. pp. 1639–1651. Springer (2003)
12. Daida, J.M., Li, H., Tang, R., Hilss, A.M.: What makes a problem GP-hard? validating a hypothesis of structural causes. In: Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference Chicago, IL, USA, July 12–16, 2003 Proceedings, Part II. pp. 1665–1677. Springer (2003)
13. Desmond, H., Bartlett, D.J., Ferreira, P.G.: On the functional form of the radial acceleration relation. Monthly Notices of the RAS **521**(2), 1817–1831 (May 2023). https://doi.org/10.1093/mnras/stad597
14. Ebner, M.: On the search space of genetic programming and its relation to Nature's search space. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). vol. 2, pp. 1357–1361. IEEE (1999)
15. Eiben, A.E., Smith, J.E.: Introduction to evolutionary computing. Springer (2015)
16. Fletcher, R.: Practical methods of optimization. John Wiley & Sons (2000)
17. de França, F.O.: A greedy search tree heuristic for symbolic regression. Information Sciences **442**, 18–32 (2018)
18. de França, F.O.: Transformation-interaction-rational representation for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 920–928 (2022)
19. de Franca, F.O., Aldeia, G.S.I.: Interaction–transformation evolutionary algorithm for symbolic regression. Evolutionary computation **29**(3), 367–390 (2021)
20. de Franca, F.O., Kronberger, G.: Reducing overparameterization of symbolic regression models with equality saturation. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1064–1072 (2023)
21. Freundlich, J., Famaey, B., Oria, P.A., Bílek, M., Müller, O., Ibata, R.: Probing the radial acceleration relation and the strong equivalence principle with the Coma cluster ultra-diffuse galaxies. Astronomy and Astrophysics **658**, A26 (Feb 2022). https://doi.org/10.1051/0004-6361/202142060
22. Gopika, K., Desai, S.: A test of constancy of dark matter halo surface density and radial acceleration relation in relaxed galaxy groups. Physics of the Dark Universe **33**, 100874 (Sep 2021). https://doi.org/10.1016/j.dark.2021.100874
23. Guimerà, R., Reichardt, I., Aguilar-Mogas, A., Massucci, F.A., Miranda, M., Pallarès, J., Sales-Pardo, M.: A Bayesian machine scientist to aid in the solution of challenging scientific problems. Science Advances **6**(5) (Jan 2020). https://doi.org/10.1126/sciadv.aav6971

24. Gustafson, S., Burke, E.K., Krasnogor, N.: On improving genetic programming for symbolic regression. In: 2005 IEEE Congress on Evolutionary Computation. vol. 1, pp. 912–919. IEEE (2005)
25. Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: A platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software **36**, 114–144 (2021). https://doi.org/10.1080/10556788.2020.1808977
26. Hu, T., Banzhaf, W.: Neutrality, robustness, and evolvability in genetic programming. Genetic Programming Theory and Practice XIV pp. 101–117 (2018)
27. Hu, T., Ochoa, G., Banzhaf, W.: Phenotype search trajectory networks for linear genetic programming. In: European Conference on Genetic Programming (Part of EvoStar). pp. 52–67. Springer (2023)
28. Kammerer, L., Kronberger, G., Burlacu, B., Winkler, S.M., Kommenda, M., Affenzeller, M.: Symbolic regression by exhaustive search: Reducing the search space using syntactical constraints and efficient semantic structure deduplication. Genetic programming theory and practice XVII pp. 79–99 (2020)
29. Kartelj, A., Djukanović, M.: RILS-ROLS: robust symbolic regression via iterated local search and ordinary least squares. Journal of Big Data **10**(1), 71 (2023)
30. Kommenda, M., Burlacu, B., Kronberger, G., Affenzeller, M.: Parameter identification for symbolic regression using nonlinear least squares. Genetic Programming and Evolvable Machines **21**(3), 471–501 (2020)
31. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
32. Kronberger, G., Burlacu, B., Kommenda, M., Winkler, S.M., Affenzeller, M.: Symbolic Regression. Chapman & Hall / CRC Press (2024)
33. Kronberger, G., de Franca, F.O.: Effects of reducing redundant parameters in parameter optimization for symbolic regression using genetic programming. Journal for Symbolic Computation (2024)
34. Langdon, W.B.: Genetic programming convergence. Genetic Programming and Evolvable Machines **23**(1), 71–104 (Aug 2021). https://doi.org/10.1007/s10710-021-09405-9
35. Lelli, F., McGaugh, S.S., Schombert, J.M.: SPARC: Mass Models for 175 Disk Galaxies with Spitzer Photometry and Accurate Rotation Curves. Astronomical Journal **152**, 157 (Dec 2016). https://doi.org/10.3847/0004-6256/152/6/157
36. Lelli, F., McGaugh, S.S., Schombert, J.M., Pawlowski, M.S.: One Law to Rule Them All: The Radial Acceleration Relation of Galaxies. Astrophysical Journal **836**(2), 152 (Feb 2017). https://doi.org/10.3847/1538-4357/836/2/152
37. Margossian, C.C.: A review of automatic differentiation and its efficient implementation. Wiley interdisciplinary reviews: data mining and knowledge discovery **9**(4), e1305 (2019)
38. McGaugh, S., Milgrom, M.: Andromeda Dwarfs in Light of MOND. II. Testing Prior Predictions. Astrophysical Journal **775**(2), 139 (Oct 2013). https://doi.org/10.1088/0004-637X/775/2/139
39. McGaugh, S.S.: A tale of two paradigms: the mutual incommensurability of ΛCDM and MOND. Canadian Journal of Physics **93**(2), 250–259 (Feb 2015). https://doi.org/10.1139/cjp-2014-0203
40. McGaugh, S.S., Wolf, J.: Local Group Dwarf Spheroidals: Correlated Deviations from the Baryonic Tully-Fisher Relation. Astrophysical Journal **722**(1), 248–261 (Oct 2010). https://doi.org/10.1088/0004-637X/722/1/248

41. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings 11. pp. 134–145. Springer (2008)
42. Milgrom, M.: A Modification of the Newtonian Dynamics - Implications for Galaxy Systems. Astrophysical Journal **270**, 384 (Jul 1983). https://doi.org/10.1086/161132
43. Milgrom, M.: A modification of the Newtonian dynamics as a possible alternative to the hidden mass hypothesis. Astrophysical Journal **270**, 365–370 (Jul 1983). https://doi.org/10.1086/161130
44. Nelson, C.G.: Techniques for program verification. Stanford University (1980)
45. Niehaus, J., Igel, C., Banzhaf, W.: Reducing the number of fitness evaluations in graph genetic programming using a canonical graph indexed database. Evolutionary Computation **15**(2), 199–221 (2007)
46. Nikuradse, J.: Laws of flow in rough pipes. Tech. rep., National Advisory Committee for Aeronautics Washington, NACA TM 1292 - Translation of "Strömungsgesetze in rauhen Rohren" VDI-Forschungsheft 361. Beilage zu "Forschung auf dem Gebiete des Ingenieurwesens" Ausgabe B Band 4, July/August 1933. (1950)
47. Oman, K.A., Brouwer, M.M., Ludlow, A.D., Navarro, J.F.: Observational constraints on the slope of the radial acceleration relation at low accelerations. arXiv e-prints arXiv:2006.06700 (Jun 2020)
48. Randall, D.L., Townsend, T.S., Hochhalter, J.D., Bomarito, G.F.: Bingo: a customizable framework for symbolic regression with genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 2282–2288 (2022)
49. Reichardt, I., Pallarès, J., Sales-Pardo, M., Guimerà, R.: Bayesian machine scientist to compare data collapses for the Nikuradse dataset. Physical Review Letters **124**(8) (Feb 2020). https://doi.org/10.1103/physrevlett.124.084503
50. Rivero, D., Fernandez-Blanco, E., Pazos, A.: DoMe: A deterministic technique for equation development and symbolic regression. Expert Systems with Applications **198**, 116712 (2022)
51. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 3 edn. (2010)
52. Seidyo Imai Aldeia, G., Olivetti de Franca, F., La Cava, W.G.: Inexact simplification of symbolic regression expressions with locality-sensitive hashing. arXiv e-prints pp. arXiv–2404 (2024)
53. Sipper, M.: Tiny genetic programming in Python. `https://github.com/moshesipper/tiny_gp` (2019)
54. Virgolin, M., Alderliesten, T., Witteveen, C., Bosman, P.A.: Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1041–1048 (2017)
55. Vlček, J., Lukšan, L.: Shifted limited-memory variable metric methods for large-scale unconstrained optimization. Journal of Computational and Applied Mathematics **186**(2), 365–390 (2006). https://doi.org/https://doi.org/10.1016/j.cam.2005.02.010
56. Willsey, M., Nandi, C., Wang, Y.R., Flatt, O., Tatlock, Z., Panchekha, P.: Egg: Fast and extensible equality saturation. Proceedings of the ACM on Programming Languages **5**(POPL), 1–29 (2021)

57. Worm, T., Chiu, K.: Prioritized grammar enumeration: symbolic regression by dynamic programming. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 1021–1028 (2013)