Zoetrope Genetic Programming for Regression

Aurélie Boisbunon
MyDataModels
Sophia Antipolis, France
abb@mydatamodels.com

Carlo Fanara
MyDataModels
Sophia Antipolis, France
cf@mydatamodels.com

Ingrid Grenet
MyDataModels
Sophia Antipolis, France
ig@mydatamodels.com

Jonathan Daeden
MyDataModels
Sophia Antipolis, France
jd@mydatamodels.com

Alexis Vighi
MyDataModels
Sophia Antipolis, France
av@mydatamodels.com

Marc Schoenauer
INRIA TAU. CNRS & UPSaclay
LISN, Orsay, France
marc.schoenauer@inria.fr

ABSTRACT

The Zoetrope Genetic Programming (ZGP) algorithm is based on an original representation for mathematical expressions, targeting evolutionary symbolic regression. The zoetropic representation uses repeated fusion operations between partial expressions, starting from the terminal set. Repeated fusions within an individual gradually generate more complex expressions, ending up in what can be viewed as new features. These features are then linearly combined to best fit the training data. ZGP individuals then undergo specific crossover and mutation operators, and selection takes place between parents and offspring. ZGP is validated using a large number of public domain regression datasets, and compared to other symbolic regression algorithms, as well as to traditional machine learning algorithms. ZGP reaches state-of-the-art performance with respect to both types of algorithms, and demonstrates a low computational time compared to other symbolic regression approaches.

CCS CONCEPTS

• **Computing methodologies** → **Representation of mathematical functions**; **Supervised learning by regression**; **Learning linear models**; **Genetic programming**; **Feature selection**.

KEYWORDS

Symbolic regression, Genetic programming, regression

ACM Reference Format:
Aurélie Boisbunon, Carlo Fanara, Ingrid Grenet, Jonathan Daeden, Alexis Vighi, and Marc Schoenauer. 2021. Zoetrope Genetic Programming for Regression. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459349

1 INTRODUCTION

Symbolic Regression (SR) is a supervised learning approach that consists in searching through a vast space of predictive models. This space encompasses the rigid linear and polynomial models by enabling other transformations such as trigonometric and logarithmic, as well as the generalized additive models by allowing (linear and) nonlinear combinations of the transformed variables (see [39] and references therein). SR models are often represented via expression trees, and may include decision tree models if we consider equality and inequality operators instead of functions (see e.g. the Boolean multiplexer in [18]). Models can also be unravelled through mathematical formulae, which make them much more interpretable than other tree or network-based machine learning algorithms such as random forests [6] or neural networks where, with few recent exceptions [16], the relationships among the variables remain hidden. SR thus offers a good tradeoff between flexibility and interpretability. Moreover, it does not need large numbers of observations as in deep neural networks, and can be applied to smaller datasets (typically from several dozens to tens of thousands).

The history of SR is closely related to that of Genetic Programming (GP), starting with the early works of Koza [18, 22]. Indeed, most of the approaches for SR are GP algorithms (often denoted GPSR), as these offer a nice framework with expression trees representing the potential models. GPSR algorithms start with a pool of initial models, which are then iteratively and randomly perturbed to create new ones, until the one that fits the data best is finally selected. Variants to this scheme are discussed in [36], whereas newer approaches are enlisted in [37]. On the contrary, "standard" machine learning algorithms are not well suited for the complex task of optimizing both parameters and model shape at the same time [30, 38]. Several exceptions are worth noting, combining the idea of expression trees with classical ML [24] or with neural networks [16].

While SR is very present in the field of Evolutionary Algorithms (EA) [24], it is almost completely absent in Machine Learning (ML) reference books [5, 13] and toolboxes [29]. Possible reasons could be the following: first, there are many SR algorithms in the literature, each offering various advantages [38], and it might be difficult to know which one to use and how to tune their often large number of parameters; second, these algorithms are often slower than most ML algorithms, with performance that did not match those of e.g. random forests up until recently; finally, earlier works mostly tested symbolic regression on synthetic data with a known equation involving few input variables, with the aim of recovering exactly this equation [17], and not often on real datasets with more than 5

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07.
https://doi.org/10.1145/3449639.3459349

variables. For a critical view on SR benchmarks we refer the reader to [25, 27] and references therein.

These limitations have been overcome in the last decade, at least partially. The issue of computational time has been treated by Geometric Semantic Genetic Programming (GSGP) 2.0 [7] with the proposition of a very efficient algorithm. However, this efficiency comes at the cost of interpretability, as the use of geometric semantic variation operators results in exponentially growing trees. The performance of GPSR has been increased for instance by the combination of GP with more standard ML approaches [2]. Finally, novel benchmarks were established lately that also compare SR and classical ML algorithms on real datasets [1, 28, 40]. These benchmarks show that the performance of random forests can be matched by increasing the number of individuals and generations, considerably slowing down the computations. So the issue remains for GPSR to get good performance in a reasonable time without losing its characteristic interpretability. A recent and promising work has been proposed in that sense [20], as well as our own work that we present here.

In this article we present a new GP algorithm called Zoetrope Genetic Programming (ZGP), which brings the following main contributions: (1) a new and unseen representation of models, allowing fast computation and feature engineering, while keeping the interpretability advantage of most SR methods through the explicit model formula; (2) novel mutation and crossover processes, leading to improvement of models over the generations; (3) performance that is comparable to the best ML (Gradient Boosting) and SR algorithms. While ZGP can handle the three main supervised learning tasks, namely regression, binary classification and multiclass classification, we focus here only on the regression one. The paper is organized as follows. Section 2 briefly presents the general context and introduces GPSR state-of-the-art frameworks which are related to our work. Section 3 describes the entire ZGP algorithm, with its uncommon representation of individuals and variation operators, as well as the choices for fitness and cost. Section 4 presents and discusses the results of our benchmark against state-of-the-art SR frameworks and ML algorithms on 98 regression datasets. Finally, we conclude on our main contributions in Section 5, and suggest potential future work.

2 BACKGROUND

2.1 Context

Given a dataset $\mathcal{D}$ made of i.i.d. observations $(X_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ with $X_i = (X_{i1}, \ldots, X_{id})$, the goal of regression algorithms is to find a function $\mathcal{M} : \mathbb{R}^d \mapsto \mathbb{R}$ modelling the link between y and X, such that it generalizes well on unseen data from the same distribution.

As common in regression problems, as performance measure for $\mathcal{M}$ on dataset $\mathcal{D}$ we use the Mean Squared Error (MSE) defined by:

$$MSE(\mathcal{M}, \mathcal{D}) = \frac{1}{\#\mathcal{D}} \sum_{(X_i, y_i) \in \mathcal{D}} (y_i - \mathcal{M}(X_i))^2 , \qquad (1)$$

where $\#\mathcal{D}$ is the number of observations in $\mathcal{D}$. Any dataset $\mathcal{D}$ used in this work will be divided into training ($\mathcal{D}_T$), validation ($\mathcal{D}_V$) and test, or holdout ($\mathcal{D}_H$) sets, by default with a 40%-30%-30% ratio. The holdout set $\mathcal{D}_H$ is never to be seen during the learning procedure, and is only used to assess the final performance of the model. The use of the training and validation sets is detailed in Section 3.2.

2.2 Related work

As mentioned above, several SR frameworks have been recently proposed and already compared to classical ML algorithms. First, some SR techniques are not based on evolutionary process. For example, Fast Function Extraction (FFX) [24] only generates a large set of linear and non linear features and then fits a linear model on the features using elastic net [41]. While the deterministic part can be attractive to avoid getting different models from one run to another, it turns out that FFX often results in much larger models than conventional GP. Evolutionary Feature Synthesis (EFS) [3] uses a similar idea, but avoids building the basis entirely by randomly generating them. It is however not a GP algorithm. The idea of linearly combining branches of a tree is also very present in GP, as it allows the construction of new features. Multiple Regression Genetic Programming (MRGP) [2] combines all the possible subtrees of a tree through LASSO [35], thereby decoupling the linear regression from the construction of a tree. More recently, La Cava et al. developed Feature Engineering Automation Tool (FEAT) [20], which trades conciseness for accuracy. It is a stochastic optimization providing a succinct syntactic representation with variable dependencies explicitly shown (in contrast to the semantic approach [31]). Another related recent work is the Interaction-Transformation Evolutionary Algorithm (ITEA) [9], which builds generalized additive models including interactions between variables.

The efficiency of GP has been another direction of study. Geometric Semantic Genetic Programming (GSGP) [26] is a technique combining trees to get new individuals and adds semantic methods for crossover and mutation in order to introduce a degree of 'awareness'. However, in GSGP the generated individuals are larger than their parents, resulting in large bloat, and longer computing times. This is addressed by using a practical development environment, GSGP-C++ [7] with operators in native C++. Finally, other frameworks propose efficient selection techniques. Age-Fitness Pareto Optimization (AFP) [33] is meant to prevent premature convergence in evolutionary algorithms by including age as an optimization criterion using a Pareto front between age and fitness. This allows younger individuals to compete with older and fitter ones. Also, ϵ-lexicase selection (EPLEX) [21] performs parent selection according to their fitness on few random training examples, dropping all the population individuals with error higher than the best error. This selection technique is used in FEAT.

With respect to the above works, ZGP proposes two novelties. First, ZGP uses a parametric representation for its models. Second, within its complex genotype-to-phenotype mapping, ZGP borrows to Geometric Semantic Crossover [26], and thus compensates the could-be limitations of a fixed representation by creating a richer set of smoother trajectories in the space of all possible analytical expressions / programs. Furthermore, this process sets a strict bound on the complexity of the resulting expressions, and thus limits the bloat.

3 THE ZGP ALGORITHM

The Zoetrope Genetic Programming[1] (ZGP) algorithm is based on the original Zoetropic representation for programs, together with the corresponding variation operators (crossover and mutation). However, the "natural selection" components of all evolutionary algorithms are here directly incorporated into the variation operators (i.e., selection takes place between the parents and their offspring only). Furthermore, ZGP uses evolutionary components to build possible branches of a regression tree, and standard ML techniques to optimize the combination of those branches.

3.1 The Zoetropic Representation

This section describes both the genotype and the genotype-to-phenotype mapping of ZGP individuals. As in standard tree-based GP [4, 18], ZGP individuals are built from a set of unary or binary operators O and a set of terminals $\mathcal{T}$, variables of the problem and ephemeral constants. The genotype of a ZGP individual is built using elements (partial expressions built on $\mathcal{T}$ and O) and fusion operations (see below). Two parameters control the size of the genotype as well as the derivation of the corresponding phenotype (the final expression used to evaluate the fitness of the individual): the number of initial elements n_e and the number of maturation stages n_m. An individual is built as follows:

Overview and notations The elements used during the process can be seen as organized in n_m levels, one per maturation step. The n_e elements of level k, denoted $(E_1^k, \dots, E_{n_e}^k)$, are constructed by maturation step k from the elements of level $k-1$. However, due to boundary conditions depending on the parity of n_e, it is more convenient to visualize all the elements in a circle (reminding the original zoetrope mechanism[2]). Figure 1 illustrates the creation process, but for the sake of simplicity, the elements of levels 0, 1, 2, 3 are denoted E_i, E'_i, E''_i and Z_i respectively (explanations below).

Initialization The n_e elements $(E_1^0, \dots, E_{n_e}^0)$ of an individual are randomly drawn in $\mathcal{T}$, being a uniformly chosen variable with 90% probability, or an ephemeral constant with 10% probability. The latter are uniformly drawn in $[C_{min}, C_{max}]$, for some user-defined parameters C_{min} and C_{max}. An individual is also initialized with a set of n_f fusion operations $\mathcal{F}_1, \dots \mathcal{F}_{n_f}$ described next.

Fusion The fusion operation $\mathcal{F}$ transforms a pair (E_i, E_j) of elements into a new pair $(E'_i, E'_j) = \mathcal{F}(E_i, E_j)$. It starts by computing

$$f(E_i, E_j) = r \cdot \text{op}_1(E_i, E_j) + (1 - r) \cdot \text{op}_2(E_i, E_j), \qquad (2)$$

where op_i, $i = 1, 2$ are operators uniformly chosen in O, and $r = U[0, 1]$ (in case op_1 or op_2 is unary, only E_i is taken into account). Elements E'_i and E'_j are then defined by

$$E'_i = b \cdot E_i + (1 - b) \cdot f(E_i, E_j)$$
$$E'_j = (1 - b) \cdot E_j + b \cdot f(E_i, E_j),$$

where $b = U\{0, 1\}$, i.e., one new element is equal to one randomly chosen original element, while the other is defined by Eq. (2). The fusion $\mathcal{F}$ is defined by $(\text{op}_1, \text{op}_2, r, b)$.

[1] ZGP is a proprietary algorithm from MyDataModels with patent pending. An open source version is currently under development.

[2] The term zoetrope historically defines one of the first animation devices before the camera, consisting of a cylinder with images inside, that seem to be moving as the cylinder is turned.

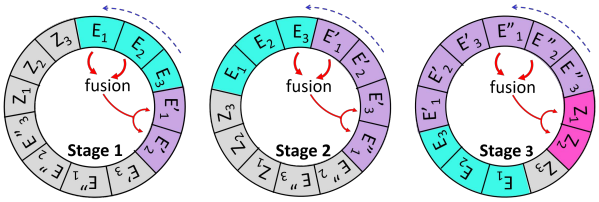

Figure 1: Illustration of Zoetropic representation building: for $n_e = n_m = 3$, there are $n_f = 4$ fusions in total, and for the sake of readability, the third one, generating $(E"_2, E"_3)$ from (E'_2, E'_3), taking place between center and right figures, is not represented. Note that $Z_3 = E"_3$ as no element is left for a fusion.

Note that these fusion operations, and in particular Equation (2), are similar in some ways to the Geometric Semantic Crossover [26]. But the linear combination with random weight is done here at the level of simple operators, not subtree, and during the genotype-to-phenotype mapping, not during crossover. In both situation, this results in a smoother landscape than only allowing blunt choices between one or the other argument, offering more transitional states to the evolutionary process.

Maturation The k^{th} maturation step, or stage, consists of the sequence of $\lfloor n_e/2 \rfloor$ fusions defining elements E_i^k from pairs of elements E_i^{k-1}. If n_e is even, then $(E_i^k, E_{i+1}^k) = \mathcal{F}(E_i^{k-1}, E_{i+1}^{k-1})$ for some fusion $\mathcal{F}$. If n_e is odd, the remaining element of maturation step k is used for the first fusion of the maturation step $k+1$, and fusions are further applied to the remaining $n_e - 1$ elements (see Figure 1–stage 2).

The Zoetrope model After n_m maturation steps, the n_e elements of level n_m, called "Zoetropes", are linearly combined to obtain the final model (see Section 3.2.1). The weights of this linear combination are obtained by minimizing the Mean Squared Error (MSE) with a sparsity-inducing regularization: the idea is to obtain the simplest possible expression, for obvious explainability reasons (see details in Section 3.2.1).

Complexity analysis There are $n_f = n_m \cdot \lfloor n_e/2 \rfloor + n_e\%2$ fusions in total. At each fusion, the size of the elements increases by the application of Eq. (2). In terms of standard GP indicators (though we never express the ZGP models as trees), the depth of $\mathcal{F}(E_i, E_j)$ is three more than the maximum depth of E_i and E_j. Hence the depth of the zoetropes is at most $3 * n_m + 1$. The linear combination applied to the zoetropes using the n_e-ary addition operator can be viewed as adding two more levels of depth. In particular, because all created individuals use the same template, the complexity of any ZGP model remains bounded. Therefore, ZGP individuals are not subject to uncontrolled bloat.

Discussion In summary, the **genotype** of a ZGP individual is made of the initial elements $(E_1^0, \dots, E_{n_e}^0)$ and the set of fusions $\mathcal{F}_1, \dots \mathcal{F}_{n_f}$ through their respective components $(\text{op}_1, \text{op}_2, r, b)$. Its **phenotype** is the model obtained after the n_m maturation steps, and the optimal linear combination of the n_e zoetropes explained in the sequel. Note that all individuals have the same number of "genes". And though all genes are not of the same type (i.e., initial

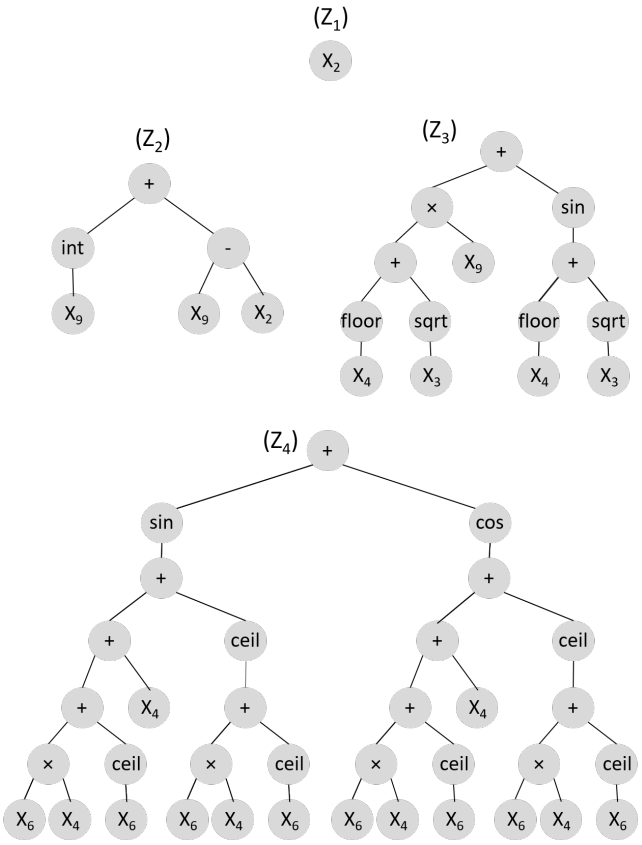

Figure 2: Examples of zoetropes that can be obtained from a set of input variables $(X_1, \ldots, X_9)$. As shown, the complexity of a zoetrope can vary from a single variable or constant to nested fusions. The r components of the fusions have been omitted for clarity.

elements or fusions defined by their 4 components), the genes at the same position in all individuals are always of the same type and have the same semantics in the genotype-to-phenotype mapping. Ultimately, the representation used in ZGP could also be viewed as a developmental representation: the initial elements are the embryo, and the fusions describe the development of the final expression from the embryo. Both the embryo and the development program are evolved here.

3.2 Fitness and Cost Functions

3.2.1 Combination of zoetropes. As said in previous Section, at the end of all fusions, the zoetropes are combined to obtain the full model as:

$$\mathcal{M}_{\boldsymbol{\alpha}}(X) = \sum_{j=1}^{n_e} \alpha_j Z_j(X), \tag{3}$$

for some weights $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{n_e}) \in \mathbb{R}^{n_e}$.

The optimal weights $\boldsymbol{\alpha}^*$ are computed by minimizing the MSE on the training set $\mathcal{D}_T$, using an Elastic net regularization [41]

$$\boldsymbol{\alpha}^* = \text{ArgMin}_{\boldsymbol{\alpha} \in \mathbb{R}^{n_e}} \left\{ MSE(\mathcal{M}_{\boldsymbol{\alpha}}, \mathcal{D}_T) + \lambda_1 \|\boldsymbol{\alpha}\|_1 + \lambda_2 \|\boldsymbol{\alpha}\|_2^2 \right\} \tag{4}$$

with $\lambda_1, \lambda_2 \in \mathbb{R}^+$. Such Elastic net regularization forces the solution to be both sparse and low biased, through the L_1 and the L_2 norms respectively. At the moment, we start by setting $\lambda_2 = 0.001$ (determined after some trial-and-errors on 6 datasets from UCI[3] and Kaggle[4]), because the value of λ_1 can then be obtained by minimizing the MSE on the validation set using the Least Angle Regression (LAR) algorithm [10], very fast for such small dimension (n_e). This is however done using the training set. Further work will consider first using Bayesian Optimization to find a possibly better value of λ_2, and using the validation set to set the value of λ_1.

Finally, note that this combination of zoetropes can be viewed as an ensemble-generalization of Maarten Keijzer' Scaled Symbolic Regression [15]: the minimum is taken here using all n_e zoetropes, though with some emphasis on sparsity, possibly adding some robustness to the final model.

3.2.2 The Fitness Function. The fitness function, used in Darwinian selection, is the MSE of the best linear combination of the zoetropes on the training set, $MSE(\mathcal{M}_{\boldsymbol{\alpha}^*}, \mathcal{D}_T)$, obtained from Equation (4). In ZGP, this fitness function is applied within the variation operators, between parents and offspring. Furthermore, the best individual in the population w.r.t. the MSE on the validation set $\mathcal{D}_V$ is stored at every generation, and after the algorithm has stopped, the overall best of these best-per-generation is returned (still according to the MSE on the validation set).

3.3 The variation operators

This section introduces the representation-specific variation operators, i.e., crossover and mutation (the initialization has been described in Section 3.1). As said, in ZGP, the selection is made within these variation operators, between parents and their offspring, using the MSE for comparisons. Furthermore, the way these operators are applied is also specific. This section will hence describe the operators as well as the choice of the individuals they are applied to.

3.3.1 The Crossover Operator. The crossover process of ZGP uses two parents, but works one-way: it only propagates components from the fittest parent to the other one, similarly in some ways to the InverOver operator for permutations [34]. It starts by selecting n_t individuals uniformly from the population (as in standard tournament selection). Then, it randomly replaces some of the 'genes' of the weakest parent by the corresponding genes of the fittest parent. The genes to replace are randomly chosen from the initial elements (terminals in $\mathcal{T}$) and the fusions, each fusion being considered as a single gene here.

3.3.2 Applying the Crossover. Our GP strategy for applying the crossover amounts to repeat $\rho_X \cdot P$ times the above procedure (tournament of size n_t, one-way gift of genes from best to worst), for some hyperparameter $\rho_X \in [0, 1]$. Its actual implementation runs some tournaments in parallel (i.e., without replacement between the tournaments), in order to decrease the overall computational time.

[3] https://archive.ics.uci.edu/ml/index.php
[4] https://www.kaggle.com/datasets

3.3.3 Point Mutation. The point mutation operator considers one parent, and works as expected: it replaces some 'genes' of the parents by random values. However, the fusions are here considered made of four 'genes' here, the four components $(\mathrm{op}_1, \mathrm{op}_2, r, b)$ (Section 3.1), that can be modified independently. For each point mutation, either one element or one fusion is randomly chosen from the "genes" and mutated.

When an element is to be mutated, it is replaced by a constant with probability ρ_{cst}, or with a variable uniformly chosen (and different from the current one if the element is a variable). When replacing a variable with a constant, this constant is simply chosen uniformly in $[C_{min}, C_{max}]$. When mutating a constant C to a new constant, an auxiliary constant $\hat{C}$ is uniformly drawn also in $[C_{min}, C_{max}]$, an operator o is uniformly drawn in $\{\times, /, +, power, nil\}$, and C is replaced by $C\ o\ \hat{C}$ (where $Cnil\hat{C} = \hat{C}$).

When a fusion is to be mutated, only one (uniformly drawn) of its four components $\mathrm{op}_1, \mathrm{op}_2, b, r$ is modified. In case of an operator, a new operator is chosen uniformly in O. r is modified by flipping one bit of its binary representation, and b is simply flipped.

Note that in ZGP, each individual designated for mutation is actually mutated twice. A first point mutation is applied to a component (element or fusion) randomly chosen from the "effective components", i.e., the components which are actually used by the model, thus ensuring that the mutation has an impact on the model. A second point mutation is applied to a component randomly chosen from all the components (effective or not), allowing components free from fitness pressure to drift and preserve diversity once they become effective [19].

3.3.4 Applying the Point Mutation. In most evolutionary algorithms, the mutation step consists in applying a mutation operator (e.g. point mutation) to a given individual, regardless of all other individuals. However, the mutation process of ZGP considers couples of individuals, and its application incorporates the Darwinian replacement selection, and explicitly handles the exploration/exploitation trade-off. It proceeds as follows.

A subset of size P/m_{mut} of the population is randomly drawn, with $m_{mut} \in [1, P/2]$ an integer hyperparameter. The individuals in this subset are randomly paired. All these pairs undergo the 2-individuals mutation described below. This loop is repeated m_{mut} times. In particular, this means that a total number of $P/2$ mutations are applied at each generation.

Within each pair obtained as described above, the *best* and *worst* individuals are identified based on their respective fitnesses. Then, depending on the number m_b of mutations that have been already been applied to *best* in previous iterations, one of the following two regimes applies:

- if $m_b < \tau_{lim}$, then *worst* is replaced by a mutant of *best*
- if $m_b \geq \tau_{lim}$, both *best* and *worst* undergo point mutation, but each offspring replaces its own parent only if its fitness improves that of this parent.

The threshold is computed as follows:

$$\tau_{lim} = P \cdot r_{lim} \cdot \frac{g}{G},$$

where P is the population size, G is the maximum number of generations, g is the current generation, and r_{lim} is a user-defined bounding parameter. This threshold is hence a linear function of the generation g. It aims at limiting the number of times a given individual can be used as a mutation's progenitor.

The first regime clearly favors exploration around good individuals, and as the algorithm advances and g increases, this regime becomes prominent, as usual in many EAs. However, limiting the number of offspring of good individuals, prevents super-individuals from invading too fast the whole population, thus favoring exploration, and allowing the algorithm to more easily escape local optima. At the same time, using some deterministic replacement selection in the second regime enforces a strong elitism, thus pushing toward convergence, still preserving some exploration, as mutation is then applied to both parents.

4 EXPERIMENTAL VALIDATION

This section describes and analyzes the performance of ZGP on regression tasks with tabular data, and compares them with those of state-of-the-art symbolic regression and classical machine learning algorithms.

4.1 Experimental Setting

The experiment closely follows the benchmark in [28], where the algorithms were run on the Penn Machine Learning Benchmarks (PMLB) database [27], a collection of real-world, synthetic and toy datasets, with a restriction to datasets with less than 3000 observations (small data regime). We compare ZGP with the same SR algorithms as in [28], namely MRGP [2], GSGP [7, 26], EPLEX [21], AFP [33], used with the best parameters their authors found by 5-fold cross-validation. To this list, we also added the more recent FEAT [20] and the deterministic FFX [24]. We also chose those algorithms because of the availability of a Python interface (*ellyn*[5] library for EPLEX and AFP, *feat*[6] package for FEAT, and the interface provided by the benchmark's authors in the case of GSGP and MRGP[7]). Indeed, while many state-of-the-art SR algorithms are open source, their source code comes in different languages (C++, Java, Matlab), hence quite some work is needed to re-implement and run those under the same conditions, which is the case of EFS [3] for instance. As for classical ML approaches, we chose the following algorithms from scikit-learn [29]: gradient boosting, random forests (RF), decision trees, elastic net, kernel ridge and linear SVR, the latter three being optimized by 5-fold cross validation. Finally, we added a multi-layer perceptron with keras [8] as in our experience, the one from scikit-learn does not perform well in general. The parameters for each algorithm are provided in supplementary material.

The experiment consisted in 20 runs of each algorithm, based on the same splits of training and test sets (70-30%) for all algorithms[8]. All datasets were standardized with scikit-learn' *StandardScaler*. We computed the Normalized Root Mean Squared Error (NRMSE), i.e., the square-root of the MSE (Eq. 4) divided by the range of target values, the R2-score (computed with scikit-learn), and the computational time for each algorithm and each run. Note that in

[5] https://github.com/EpistasisLab/ellyn
[6] https://github.com/lacava/feat
[7] https://github.com/EpistasisLab/regression-benchmark
[8] Note that some of the algorithms, including ZGP, further split the training set into training and validation, the rate of which was let to each algorithm's default parameters.

[28], only 10 independent runs were run for each algorithm, with random train-test splits. However, given the variability of symbolic regression approaches, we believe it is more robust to increase the number of runs, and fairer to compare them on exactly the same data.

All experiments were performed on a HP Z8 server with 40 cores[9]. All runs end when the maximum number of generation G is reached, or when the standard deviation of the best fitness over a window of size L reaches some user-defined threshold τ_σ, whichever comes first. The code to replicate the comparison experiments is provided in a public Gitlab repository[10] along with a csv file containing the results for all the algorithms, runs and datasets.

4.2 Hyperparameters

ZGP has quite a large number of hyperparameters. On the one hand, this allows a great flexibility when tuning the algorithm. But on the other hand, it makes its use time-consuming. Hence some default values have been fixed by intensive trial-and-error experiments performed on a few datasets from UCI/Kaggle as well as data from clients. These values are reported in Table 1. The optimization of these hyperparameters by some automatic Hyper Parameter Optimization (HPO) procedure, like SMAC [14], AutoSkLearn [11] or HyperBand [23] will be the subject of further work.

Table 1: Hyperparameters and default values used in ZGP

Hyperparameter name	Symbol	Value
Operator set	O	{+,-,*,/,abs, sqrt, sin, cos, $\lfloor\rfloor$, $\lceil\rceil$, int, mod}
# elements	n_e	7
# maturation stages	n_m	3
Interval for constants	$[C_{min}, C_{max}]$	[-3, 3]
Proba. of constants	ρ_{cst}	0.1
Xover tournament size	n_t	12
Xover param.	ρ_X	0.1
Mutation param.	m_{mut}	4
Threshold mut. regime	r_{lim}	0.1
Population size	P	500
Max. # of generations	G	100
Stopping criterion*	L, τ_σ	30, $1e^{-3}$

*The algorithm is stopped if either one of the two following criterion is reached: $g = G$ (the number of generations reaches the maximum) or the standard deviation of the best fitness over L generations goes below a threshold τ_σ (inspired by [32]).

4.3 Results and Discussion

As in [28], we report the median values for R2 and NRMSE over the 20 runs, for each algorithm and each dataset.

Figure 3 compares the distribution of these median R2 scores (3a) and NRMSE (3b) over all datasets, while the red dots show the average of the median R2/NRMSE scores over all datasets ("average R2/NRMSE" in the sequel) . The algorithms are ordered by decreasing average R2 and increasing average NRMSE, the best one being on the left. Table 2 gives the average rank for each algorithm, based on the median R2 scores (middle column) and median NRMSE (right column) for each dataset. Standard deviations of the ranks are given in parenthesis.

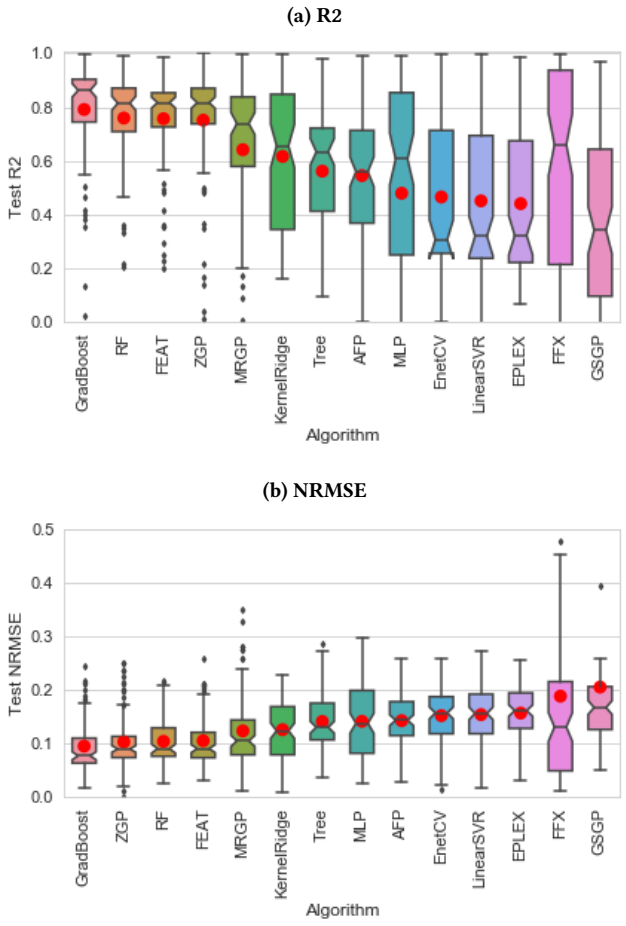

Figure 3: Distribution of the median performance (top: R2, bottom: NRMSE) on test set for each dataset. Red points show the average of median R2 over all datasets, and the algorithms are ordered by this measure (best is left, with value closest to 1).

Figure 3 and Table 2 show that gradient boosting attains the best performance, closely followed by random forests, FEAT and ZGP, and less closely by MRGP. Note that ZGP has lower average R2 than FEAT and better average rank in R2 and NRMSE. This fact comes from a few worse estimations for ZGP on some datasets (lower outliers in R2, Figure 3a), and better ones for other datasets (higher first and third quartiles in R2). The remaining algorithms display lower performance with a much higher variance, especially on the R2 scores. To complete the results, Wilcoxon signed-rank tests have been performed and are given in Supplementary Material (Section 3). They show that ZGP is statistically equivalent to random forests

[9]CPU Intel(R) Xeon(R) Silver 4114, 2.20GHz, 64 GigaBytes of RAM.
[10]https://gitlab.devenv.mydatamodels.com/publications/bench-zgp-symbolic-regression

Table 2: Average (and standard deviation) of the ranks in median R2-scores and NRMSE on test set.

Algorithm	R2 avg rank (std)	NRMSE avg rank (std)
GradBoost	3.7 (2.9)	3.7 (2.9)
ZGP	**4.9 (3.0)**	**5.0 (3.0)**
RF	5.0 (2.7)	5.1 (2.7)
FEAT	5.6 (2.7)	5.4 (2.8)
KernelRidge	6.0 (3.5)	6.1 (3.5)
MRGP	7.2 (3.4)	7.2 (3.3)
FFX	7.5 (5.1)	7.5 (5.2)
MLP	7.9 (4.3)	7.9 (4.2)
AFP	8.4 (2.0)	8.4 (1.9)
EnetCV	8.4 (4.0)	8.4 (4.0)
LinearSVR	9.2 (3.9)	9.1 (4.1)
Tree	9.6 (2.9)	9.6 (3.0)
EPLEX	10.3 (3.4)	10.2 (3.4)
GSGP	11.5 (2.8)	11.5 (2.8)

for both the R2 and NRMSE metrics, and that it is equivalent to FEAT for R2 only.

To further the comparison for symbolic regression, Figure 4 displays the performance in R2 against the computational time for all SR algorithms (top), and for the top three SR algorithms (bottom), namely ZGP, FEAT and MRGP. A 'good' algorithm should be in the upper left corner of these graphs (high performance and low computational time). Note that this comparison of runtime is somewhat qualitative because the algorithms rely on different programming languages (ZGP and FEAT are in C++ with a Python interface, while MRGP is in Java). In order to make the comparison as fair as possible, we provided FEAT and MRGP with the maximum execution time as run by ZGP, because both FEAT and MRGP require an upper limit in their input time parameter.

These figures show that ZGP and FEAT share the best performance. Moreover, while all SR algorithms have scattered computational time, GSGP is the fastest SR algorithm, complying with its claim. However, it has a large variance in performance, as does FFX. Among the best three, ZGP thus shows the highest performance and the shortest computational time. Also, we note that the algorithms with the lowest performance, GSGP, AFP and EPLEX, are those relying on the smallest operator set $O = \{+, -, *, /\}$. On the contrary, the best performance is obtained by algorithms with a wider operator set, including trigonometric functions and square roots among others (the full list of operators for each algorithm is provided in supplementary material). The choice of the operator set appears to be an important one: we investigated it further by expanding it for EPLEX and AFP to $\{+, -, *, /, \sin, \cos, \text{sqrt}\}$, and their performance was indeed greatly improved, but still far from those of ZGP and FEAT; we therefore decided to keep the default set in the results presented here to be consistent with the benchmark in [28], and to report the corresponding metrics in supplementary material. Note also that EPLEX and AFP are selection methods, and not full GPSR algorithm, and that EPLEX is used as a selection mechanism for FEAT. As for the deterministic FFX, it turns out that

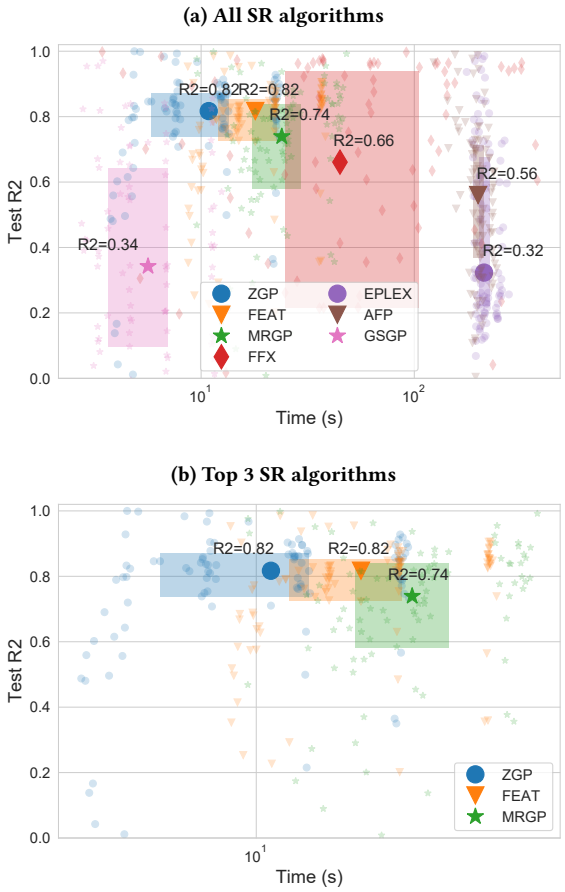

Figure 4: Performance (R2 on test) vs CPU time for SR algorithms (top:all, bottom: best ones, ZGP, FEAT, MRGP). Transparent markers are the medians for each dataset, large plain markers are the medians over all datasets, and rectangle transparent patches are the 25%-75% percentiles. The closest from the upper left corner (high R2, low CPU time) the better.

both performance and computational time are widely scattered, ranging from very good R2 and low computational time for some datasets, to the worst R2 or computational time on others. It is also worth noting that FFX is the only algorithm that cannot be parametrized directly to run on one thread only and it actually spans all 40 cores of our server, while all the other algorithms were limited to one core per run.

We are aware of the limitations of the datasets utilized. Despite their number, as mentioned in this section and in the introduction, a good portion (62 of them) consists of simulated data from the Friedman collection of artificial datasets, that follow a known non-linear function with only 3 relevant variables, as described in [12]. Hence, the chosen database may be a favorable setting for symbolic regression approaches that tend to select few variables in the final model (among which ZGP).

5 CONCLUSION AND FUTURE WORK

ZGP, a novel GPSR algorithm, is presented and its inner working explained in detail. The algorithm has been validated on regression tasks, in comparison with several-state-of-the art algorithms, both from classic ML tools and from existing GP-based SR frameworks.

The performance of ZGP is comparable or better than the state-of-the-art SR algorithms, in terms of accuracy of the resulting model (measured both by the RMSE or the R2), and of computational time. It is also comparable to state-of-the-art classic ML algorithms, but performs somewhat worse than the most advanced ML algorithms like gradient boosting. However, the comparison of the computational time is semi-quantitative as it is difficult to guarantee conditions that are fully equivalent for all.

Similarly to the majority of the SR algorithms, ZGP's interpretability is attained through a tight selection of variables, and the output of an analytical formula, linking the selected variables to the target. However, and different from most other SR algorithms, ZGP is "bloat-adverse by design": the zoetropic representation, and the genotype-to-phenotype mapping give an upper bound for the complexity of all ZGP models. Last but not least, ZGP performs feature construction and selection: the "zoetropes", the elements obtained on the last layer of the development process, are simply combined by linear regression. Therefore, they do represent useful features, being a by-product of the algorithm rather than a separate pre-processing step. These features offer yet another insight on the interpretation of the model.

One of the specifics of ZGP is the use of the fusion operation during the genotype-to-phenotype mapping (see Section 3.1 and in particular Equation 2). No operator is applied alone, and smooth transitions from one operator to the other are possible through modifications by mutation of the random weight r, in a way similar to that of Geometric Semantic Crossover [26]. However, the linear combination is limited here to simple operators, and is only performed n_m times, thus does not result in uncontrolled bloat: instead of increasing the search space by augmenting the complexity of the trees, as in traditional GP, the search space is extended in ZGP by replacing the discrete set of operators by the continuous family obtained by their linear combinations. On-going ablation studies are investigating this hypothesis.

In contrast to algorithms designed for big data, ZGP, like all GP-based SR algorithms, can attain its results by handling datasets with less than a few thousands of observations (less than 3000 in the present experiments), a context often loosely referred to today as "small data". Whereas emphasis has been put on Big Data in the recent years due to impressive results in image recognition and Natural Language Processing, to name a few, for many more companies out there the available data does not qualify as "Big".

As mentioned in the introduction, ZGP can also be applied to classification and benchmarking in both binary, and multi-class tasks is the subject of on-going work. Furthermore, an extension of the benchmark to a database including more real-world datasets for all three tasks will provide a fuller assessment for those algorithms.

Even though the inner mechanism of the ZGP algorithm does limit the bloat, a major effort for future work is the quantitative assessment of the model complexity. Several measures may capture the complexity of symbolic regression models, and we plan to assess it as an additional level for model selection. Finally, hyperparameter tuning is often performed ad-hoc, whereas a systematic treatment may help, in particular to select the number of elements and stages, which can be constraining at present.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Patrick Oneill who was at the origin of ZGP, as well as the technical teams of MyDataModels for their involvement in the development of ZGP, in particular Jean-Robert Polrot, Marcello Mansueto, Alina Tuholukova, and Norbert Leon. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported by the GPITISS project funded by the i-Lab innovation program.

REFERENCES

[1] Michael Affenzeller, Bogdan Burlacu, Viktoria Dorfer, Sebastian Dorl, Gerhard Halmerbauer, Tilman Königswieser, Michael Kommenda, Julia Vetter, and Stephan Winkler. 2019. White Box vs. Black Box Modeling: On the Performance of Deep Learning, Random Forests, and Symbolic Regression in Solving Regression Problems. In International Conference on Computer Aided Systems Theory. Springer, 288–295.

[2] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. 2014. Multiple regression genetic programming. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. 879–886.

[3] Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. 2015. Building predictive models via feature synthesis. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. 983–990.

[4] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. 1998. Genetic programming. Springer.

[5] Christopher M Bishop. 2006. Pattern recognition and machine learning. Springer, New York, NY. https://cds.cern.ch/record/998831 Softcover published in 2016.

[6] Leo Breiman. 2001. Random forests. Machine learning 45, 1 (2001), 5–32.

[7] Mauro Castelli and Luca Manzoni. 2019. GSGP-C++ 2.0: A geometric semantic genetic programming framework. SoftwareX 10 (2019), 100313.

[8] François Chollet et al. 2015. Keras. https://keras.io.

[9] Fabrício Olivetti de França and Guilherme Seidyo Imai Aldeia. 2020. Interaction-Transformation Evolutionary Algorithm for Symbolic Regression. Evolutionary Computation (2020), 1–25.

[10] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. The Annals of statistics 32, 2 (2004), 407–499.

[11] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In Advances in Neural Information Processing Systems, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc., 2962–2970. https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf

[12] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. Annals of statistics (2001), 1189–1232.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.

[14] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'05). Springer-Verlag, Berlin, Heidelberg, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40 event-place: Rome, Italy.

[15] Maarten Keijzer. 2004. Scaled Symbolic Regression. Genetic Programming and Evolvable Machines 5 (09 2004), 259–269. https://doi.org/10.1023/B:GENP.0000030195.77571.f9

[16] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. 2020. Integration of neural network-based symbolic regression in deep learning for scientific discovery. IEEE Transactions on Neural Networks and Learning Systems (2020).

[17] Michael F Korns. 2011. Accuracy in symbolic regression. In Genetic Programming Theory and Practice IX. Springer, 129–151.

[18] John R Koza. 1992. Genetic programming: on the programming of computers by means of natural selection. Vol. 1. MIT press.

[19] William La Cava, Thomas Helmuth, Lee Spector, and Kourosh Danai. 2015. Genetic programming with epigenetic local search. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. 1055–1062.

[20] William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason H. Moore. 2019. Learning concise representations for regression by evolving networks of trees. In International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1807.00981

[21] William La Cava, Lee Spector, and Kourosh Danai. 2016. Epsilon-lexicase selection for regression. In Proceedings of the Genetic and Evolutionary Computation Conference 2016. 741–748.

[22] William B Langdon and Riccardo Poli. 2013. Foundations of genetic programming. Springer Science & Business Media.

[23] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. Journal of Machine Learning Research 18, 185 (2018), 1–52. http://jmlr.org/papers/v18/16-558.html

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

[25] James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. 2012. Genetic programming needs better benchmarks. In Proceedings of the 14th annual conference on Genetic and evolutionary computation. 791–798.

[26] Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. 2012. Geometric semantic genetic programming. In International Conference on Parallel Problem Solving from Nature. Springer, 21–31.

[27] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData Mining 10, 1 (11 Dec 2017), 36. https://doi.org/10.1186/s13040-017-0154-4

[28] Patryk Orzechowski, William La Cava, and Jason H Moore. 2018. Where are we now? A large benchmark study of recent symbolic regression methods. In Proceedings of the Genetic and Evolutionary Computation Conference. 1183–1190.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12 (2011), 2825–2830.

[30] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. 2008. A field guide to genetic programming. Lulu. com.

[31] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should I trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 1135–1144.

[32] Olga Rudenko and Marc Schoenauer. 2004. A steady performance stopping criterion for Pareto-based evolutionary algorithms. In 6th International Multi-Objective Programming and Goal Programming Conference.

[33] Michael Schmidt and Hod Lipson. 2011. Age-fitness Pareto optimization. In Genetic programming theory and practice VIII. Springer, 129–146.

[34] Guo Tao and Zbigniew Michalewicz. 1998. Inver-over operator for the TSP. In International Conference on Parallel Problem Solving from Nature. Springer, 803–812.

[35] Robert Tibshirani. 2011. Regression shrinkage and selection via the lasso: a retrospective. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 73, 3 (2011), 273–282.

[36] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. 2014. A survey of semantic methods in genetic programming. Genetic Programming and Evolvable Machines 15, 2 (2014), 195–214.

[37] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter A. N. Bosman. 2020. Improving Model-Based Genetic Programming for Symbolic Regression of Small Expressions. Evolutionary Computation (Jun 2020), 1–27. https://doi.org/10.1162/evco_a_00278

[38] Sjoerd de Vries. 2018. Sensitivity Analysis Based Feature-Guided Evolution for Symbolic Regression. Master's thesis.

[39] Jan Žegklitz and Petr Pošík. 2017. Symbolic regression algorithms with built-in linear regression. arXiv preprint arXiv:1701.03641 (2017).

[40] Jan Žegklitz and Petr Pošík. 2020. Benchmarking state-of-the-art symbolic regression algorithms. Genetic Programming and Evolvable Machines (2020), 1–29.

[41] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. Journal of the royal statistical society: series B (statistical methodology) 67, 2 (2005), 301–320.