A Survey of Statistical Machine Learning Elements in Genetic Programming

Alexandros Agapitos, Roisin Loughran, Miguel Nicolau, Simon Lucas, Michael O'Neill, and Anthony Brabazon

***Abstract*—Modern genetic programming (GP) operates within the statistical machine learning (SML) framework. In this framework, evolution needs to balance between *approximation* of an unknown target function on the training data and *generalization*, which is the ability to predict well on new data. This paper provides a survey and critical discussion of SML methods that enable GP to generalize.**

***Index Terms*—Bias-variance tradeoff, classification, generalization, genetic programming (GP), model averaging, model selection, overfitting, regularization, statistical machine learning (SML), symbolic regression.**

I. Introduction

In supervised learning, a labeled example is a pair (x, y), where $x \in X$ is a random vector of explanatory variables and $y \in Y$ is a random response variable associated with x through a true but unknown mapping g. The vector of explanatory variables $x^T = (x_1, x_2, \dots, x_p)$ may contain quantitative, ordinal, and categorical variables, whereas the response variable y is quantitative in the case of regression, and categorical in the case of classification. A joint probability distribution $P(x, y)$ is assumed over the space of labeled examples. A training sample D of size N is a set of labeled examples $D = \{(x_i, y_i)\}_{i=1}^N$ that are assumed to be independent and identically distributed according to $P(x, y)$.

The goal is to use the training sample D to find a function $f : X \to Y$, such that over the joint distribution $P(x, y)$ the expected value $\mathbb{E}_{x,y}$ of some specified *loss function* $L(y, f(x))$ is minimized

$$f^*(x) = \underset{f \in F}{\arg\min}\ \mathbb{E}_{x,y}\big[L(y, f(x))|D\big] \tag{1}$$

where F is the hypothesis set (i.e., the set of candidate functions f). Examples of loss functions are the *squared loss* $L(y, f(x)) = (y - f(x))^2$ for regression, and the *zero-one loss* $L(y, f(x)) = I(y \neq f(x))$ for classification, where $I(\cdot)$ is an indicator function that returns 1 when its argument is true, and 0 otherwise.

The expected value $\mathbb{E}_{x,y}[L(y, f(x))|D]$ is referred to as *generalization error*, where both x and y are drawn from their joint distribution $P(x, y)$. Here, the training sample D is fixed, and we expressed the explicit dependence of f on D.

In practice, generalization error E_{test} cannot be computed because the distribution $P(x, y)$ is unknown. Instead, a learning algorithm outputs a function that minimizes the loss function over the finite training sample. Training error E_{train} minimization is formulated as

$$f^*(x) = \underset{f \in F}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2}$$

Minimizing training error leads to infinitely many solutions; any function passing through the training points is a solution. *Overfitting* is the process where fitting the training data well no longer indicates that a similar E_{test} will be attained, and may actually lead to the opposite effect. The main case of overfitting is when a function $f^{(1)}(x)$ is chosen over another function $f^{(2)}(x)$ because of $E_{\text{train}}(f^{(1)}(x)) < E_{\text{train}}(f^{(2)}(x))$, and this results in $E_{\text{test}}(f^{(1)}(x)) > E_{\text{test}}(f^{(2)}(x))$. There is a distinct difference between *bad generalization* and *overfitting*. Bad generalization implies that $E_{\text{test}}(f) \gg E_{\text{train}}(f)$, which is a likely outcome when overfitting has occurred. Overfitting is the process of picking a function with lower and lower E_{train} resulting in higher and higher E_{test}.

In problems with multiple training examples $(x_i, y_{il}), l = 1, \dots, M_i$ at each input x_i the risk of large generalization error is reduced. If training sample size N is sufficiently large such that multiple observations at each x_i or a small neighborhood around x_i are guaranteed and densely sampled, then solution f passes through the average values of y_{il} at each x_i, or the average values of y_{il} at a small neighborhood around x_i. This solution approximates the conditional expectation $f(x) = \mathbb{E}(y|x = x_i)$, when best is measured by average squared error [1]. In all other cases, in order to obtain a useful function that generalizes for finite N, we must restrict the eligible solutions in the error minimization problem of (2) to a smaller set of candidate models, and/or use some kind of averaging among a committee of models that are diverse in the way they are constructed. In general, the restrictions imposed by statistical machine learning (SML) methods can

Manuscript received September 12, 2017; revised March 31, 2018, August 7, 2018, and November 30, 2018; accepted January 28, 2019. Date of publication February 21, 2019; date of current version November 27, 2019. The work of A. Agapitos was supported by the Science Foundation Ireland under Grant 08/SRC/FM1389. The work of R. Loughran was supported by the Science Foundation Ireland under Grant 13/IA/1850. *(Corresponding author: Alexandros Agapitos.)*

A. Agapitos is with the School of Computer Science, University College Dublin, Dublin 4, D04 N2E5 Ireland (e-mail: alexagapitos@gmail.com).

R. Loughran, M. Nicolau, M. O'Neill, and A. Brabazon are with the School of Business, University College Dublin, Dublin 4, D04 N2E5 Ireland.

S. Lucas is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, London E1 4NS, U.K.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2019.2900916

be regarded as *model complexity* restrictions. This translates to some kind of regular behavior in small neighborhoods of the input space X. That is, for input points x with small distance from each other in some metric, a solution exhibits some kind of nearly constant, linear, or low-order polynomial behavior [1].

A. Scope of This Paper

We are interested in the problem of generalization when learning a model of a real-valued or categorical-valued target function from input–output examples using GP [2].[1] A previous survey article on GP generalization was published in 2002 [3]. This paper provided a description of a framework for the learning problem, introduced the concept of probably approximately correct learning, and briefly reviewed some applications of GP to supervised and reinforcement learning tasks. A method to dynamically sample training examples throughout evolution was shown to improve the generalization of programs evolved to control an artificial ant on the Santafe trail.

The survey of the early GP literature in [3] suggested that the vast majority of research papers have put all emphasis on the *consistency* of GP in terms of routinely evolving solutions across a number of independent evolutionary runs. The sole criterion of a consistent GP system has often been the low variance of the training error across runs. One of the main conclusions of [3] is the lack of independent test sets for assessing the ability of evolved programs to generalize.

Since 2002 research in GP for symbolic regression and classification [2], [4] has reached a considerable level of maturity, and has built on elements rooted in the SML literature. In addition to consistency, recent GP studies put an emphasis on generalization; they assess prediction ability on unseen data using an independent test set, and often benchmark GP against its sister SML methods. The aim of this paper is to survey and critically discuss SML methods that enable GP to generalize.

As a final remark, this survey focuses on GP that is realized through variation operators of recombination and mutation. A complementary active research thread aims at applying generative machine learning models to the construction of expression-tree like programs that represent composition of functions, known collectively as probabilistic model building GP (PMBGP) [5]. Examples of generative models that have been previously used are graphical models in the works of [6], and probabilistic context-free grammars in the works of [7]. The authors of a recent survey on the area have already pointed out [5] that the issues of learning and generalization in PMBGP are as of time of this writing still at their infancy, therefore we decided not to address this area of research in the present survey.

B. Taxonomy of SML Elements in GP

The Elements of Statistical Learning by Hastie *et al.* [1] was first published in 2001, and since that time it has become one of the most important references on the fundamental principles and methods of SML. The authors provide a comprehensive treatment of SML methods within the model selection framework of *bias-variance tradeoff*, which captures the fundamental tradeoff between generalization error and model complexity for a given prediction method and training sample. For regression problems, the bias-variance tradeoff decomposes the expected generalization error into an additive model of variance and bias terms. *Variance* refers to the amount by which model f would change if it is fitted using a different training sample. In general, more complex models have higher variance, in which case small changes in the training data can result in large changes in the produced f. *Bias* refers to the error that is introduced by approximating a target function, which may be very complicated, using a much simpler model. In general, less complex models will have higher bias. Bias and variance are two competing properties of SML models. In order to minimize the expected generalization error we need an amount of model complexity that simultaneously achieves *low variance* and *low bias*.

In providing a taxonomy of SML elements adopted in GP we follow a similar organization as in [1]. *Model selection* [1, Chs. 3 and 7] are a family of methods to assess the generalization performance of a model in order to decide on the optimal amount of model complexity for a given prediction method and training data. Model selection encompasses also aspects of feature subset selection and learning algorithm hyper-parameter optimization. *Regularization* methods [1, Chs. 3 and 5] control the fit by adding a model complexity penalty to the training error. The penalty is defined to be large for functions that vary too rapidly over small neighborhoods of the input space. Both feature subset selection and regularization are families of methods for restricting the eligible solutions in the minimization problem of (2), and therefore improve generalization error by reducing variance. *Model averaging* methods [1, Chs. 8, 10, and 15] use a committee of models for prediction rather than a single model. The majority of model averaging methods reduce the variance of high-variance, low-bias models [1, Ch. 8].

Deciding on which papers to include in this survey is a tall order because evolutionary learning is so distinct in its use of a broad range of heuristics that have demonstrated improvements in generalization when applied to GP. The above grouping of SML methods into the classes of model selection, regularization, and model averaging serves as a good base for organizing the literature review. Nonetheless this categorization is not exhaustive.

We identified three additional families of methods tailored to the inductive bias and heuristics of learning with GP. These are: 1) data-centric methods that revolve around training data sampling; 2) fitness functions for regression and classification that use objectives other than squared loss or zero-one loss; and 3) search operators and selection schemes. Since regularization methods use complexity-penalty-augmented fitness functions, and data resampling techniques are involved in fitness evaluation we decided to merge all three classes of regular, data-centric methods, and other fitness functions into a super class named *fitness evaluation*. Data resampling methods sit at the heart of validation-based model selection

[1]In GP, the terms model and program refer to the same entity and will be used interchangeably.

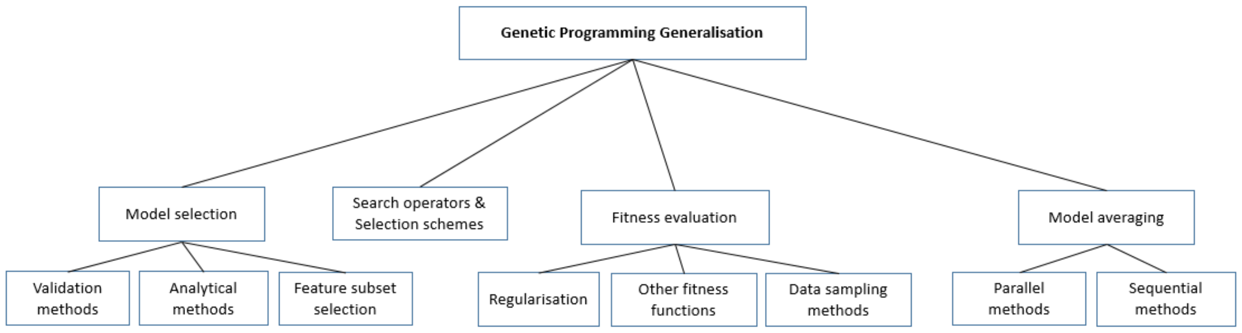


Fig. 1. High-level view of research on GP generalization.

and model averaging methods like bagging, stochastic gradient boosting, and stacking in SML [1]. Nonetheless, contrary to model selection where data resampling is used to estimate generalization error, and contrary to model averaging where data resampling is used to build different models of a committee, the research on data-centric methods considered in fitness evaluation class deals with estimating training error during the evolution of standalone predictors. Finally, the effect of search operators and selection heuristics on generalization has received very limited attention. We group these two research themes together under the same class.

In summary, the overall taxonomy that we propose is composed of four classes: 1) model selection; 2) fitness evaluation; 3) model averaging; and 4) search operators and selection schemes.

C. Structure of This Paper

We begin by presenting the decomposition of the expected generalization error into bias and variance terms for the case of regression, and discuss its implications for GP. We proceed with a survey of GP research organized according to the taxonomy that we devised previously. A separate section is devoted to every major class of the taxonomy, which is further structured into two sections. The first section reviews work in an active way, synthesizing similar papers together to present the main points of the idea. In the second section, we present a critical discussion of the methods, summarizing their strengths/weaknesses, and where applicable we draw attention to topics for future research. The final section concludes this paper, and provides recommendations for applying GP in a production environment.

Fig. 1 presents a high-level view of research on GP generalization, while Table I summarizes the surveyed papers under each class of the proposed taxonomy.

II. Decomposition of Generalization Error Into Bias and Variance: Implications for GP

The bias-variance decomposition of the expected generalization error for the case of regression captures the tradeoff between approximating a target function on the training data and generalizing on new data. The analysis essentially highlights the need for *model complexity control* when learning a prediction model using a finite training sample. This section starts by presenting the error decomposition into bias and variance terms, and then discusses its implications for evolving a model using GP. The derivations follow those of [8, Ch. 9, p. 333].

Let $f^{(D)}(x)$ be the output of the prediction model for input x, and $g(x) = y$ be the real-valued target evaluated at x. The generalization error $E_{\text{test}}(f^{(D)})$ using the squared loss is defined as

$$E_{\text{test}}(f^{(D)}) = \mathbb{E}_{x,y}\left[\left(f^{(D)}(x) - g(x)\right)^2\right] \tag{3}$$

where $\mathbb{E}_{x,y}$ denotes the expected value with respect to the joint probability distribution $P(x, y)$. We have made explicit the dependence of model $f^{(D)}$ on the training dataset D. We can rid (3) of the dependence on a particular dataset by taking the expectation with respect to all datasets. This produces the expected generalization error $\mathbb{E}_D[E_{\text{test}}(f^{(D)})]$ for the prediction model as follows:

$$\begin{aligned}\mathbb{E}_D\left[E_{\text{test}}\left(f^{(D)}\right)\right] &= \mathbb{E}_D\left[\mathbb{E}_{x,y}\left[\left(f^{(D)}(x) - g(x)\right)^2\right]\right]\\ &= \mathbb{E}_{x,y}\left[\mathbb{E}_D\left[\left(f^{(D)}(x) - g(x)\right)^2\right]\right].\end{aligned} \tag{4}$$

The term $\mathbb{E}_D[f^{(D)}(x)]$ is the "average model," which we denote by $\bar{f}(x)$. One can interpret $\bar{f}(x)$ as follows: we sample B number of datasets $D_1, D_2, \ldots, D_B$ using distribution $P(x, y)$, and apply the learning algorithm to each dataset to produce B models $f_1, f_2, \ldots, f_B$. We then estimate the average model for input x by $\bar{f}(x) \approx (1/b)\sum_{b=1}^{B} f_b(x)$. This implies that we are treating $f(x)$ as a random variable, with the randomness coming from the dataset used to train f. $\bar{f}(x)$ is the expected value of this random variable for a particular input x. We can now rewrite the expected generalization error in terms of the average model as follows:

$$\begin{aligned}&\mathbb{E}_{x,y}\left[\mathbb{E}_D\left[\left(f^{(D)}(x) - g(x)\right)^2\right]\right]\\ &\quad= \mathbb{E}_{x,y}\left[\mathbb{E}_D\left[\left(f^{(D)}(x) - \bar{f}(x) + \bar{f}(x) - g(x)\right)^2\right]\right]\\ &\quad= \mathbb{E}_{x,y}\left[\mathbb{E}_D\left[\left(f^{(D)}(x) - \bar{f}(x)\right)^2 + \left(\bar{f}(x) - g(x)\right)^2\right.\right.\\ &\qquad\left.\left. + 2\left(f^{(D)}(x) - \bar{f}(x)\right)\left(\bar{f}(x) - g(x)\right)\right]\right].\end{aligned} \tag{5}$$

TABLE I
TAXONOMY OF SML ELEMENTS AND OTHER GENERALIZATION-ENHANCING METHODS ADOPTED IN GP

Class of Methods	References
MODEL SELECTION	
Assessment of generalisation performance (Section III-A)	
Analytical methods (Section III-A1)	
Akaike Information Criterion, Final Prediction Error, Predicted Residual Error Sum of Squares	[11]
Mallows C_p statistic	[12]
Backward elimination	[13]
Validation methods (Section III-A2)	
Early stopping	[14], [15], [16], [17]
Ultimate selection of a model from a GP run	[18], [19], [20], [21], [22]
Evolutionary Algorithm hyper-parameter optimisation	[23]
Feature subset selection (Section III-B)	
Filter methods	[24], [25], [26], [27]
Wrapper methods	[27], [28], [29], [30]
Embedded methods	[31], [32], [33], [34], [35], [36], [37], [38] [39], [40], [41], [42], [43], [44]
FITNESS EVALUATION	
Regularisation (Section IV-A)	
Semantic complexity penalty (Section IV-A1)	
Tikhonov functional	[13], [45], [46], [47]
Curve Length of First Derivative	[48]
Order of non-linearity	[49]
Structural Risk Minimisation (empirical approximation of VC-dimension as model complexity)	[50]
Syntactic complexity penalty (Section IV-A2)	
Size of expression-tree	[19], [51], [52], [53], [54], [55], [56] [57], [58], [59]
Expressional complexity	[49]
Minimum Description Length	[13], [45], [59], [60]
Akaike Information Criterion (node count as model complexity)	[61]
Bayesian Information Criterion (node count as model complexity)	[61]
Structural Risk Minimisation (count of multiplication and division nodes as model complexity)	[61]
Fitness functions for regression (Section IV-B)	
Weighted average of errors (weight based on ruggedness of evolved function for input x)	[62]
Weighted average of errors (weight based on proximity, surrounding, remoteness, deviation from k-nearest points)	[63]
Fitness sharing (weight based on relative performance of population on training example)	[20]
Weighted average of (1) bootstrap standard deviation of squared error and (2) squared error on entire sample	[64]
Product of (1) bootstrap standard deviation of classification error rate by (2) error rate on entire sample	[65]
Pareto-based with objectives: (1) error, (2) error variance, (3) expression-tree size	[54]
Fitness functions for classification (Section IV-B)	
Product of (1) sensitivity, by (2) specificity, by (3) syntactic complexity	[52]
Multiple objectives for (1) Detection rate, (2) False alarm rate, (3) False alarm area, and (4) program size	[66]
Distance between decision boundary $f(x) = 0$ and training input points x	[67]
Measure of separability between program-output-value distributions for different classes	[68], [69], [70], [71]
Dynamic thresholds for binary classification	[72]
Approximations of Area Under Receiver Operating Characteristic Curve	[70], [73], [74]
Weighted average of (1) average daily return, and (2) p-value based on statistical hypothesis test of *superior predictive ability*	[18]
Stepwise Adaptation of Weights (weighted average of classification scores, with weight increasing for misclassified examples)	[75]
Vicinal Risk Minimisation	[76]
Data sampling methods (Section IV-C)	
Uniformly random down-sampling	[20], [55], [77], [78], [79], [80], [81]
Adaptive down-sampling based on difficulty	[78], [82]
Adaptive down-sampling based on competitive coevolution of programs vs. training examples	[20], [83], [84], [85]
Adaptive incremental sampling	[59], [86], [87], [88]
Layered complexification of training sample	[86]
MODEL AVERAGING	
Parallel training of component models (Section V-A)	
Averaging/Voting that does not require training of the model combination scheme:	
Bagging (unweighted majority vote for classification or unweighted averaging for regression)	[10], [89], [90], [91], [92], [93]
Majority voting, Maximum, Minimum, Average, Product, Naive Bayes, Behaviour-knowledge base, Decision templates	[92]
Weighted majority vote	[94]
Winner-takes-all, unweighted average, error-weighted average, weights coevolution, majority voting, weighted majority voting	[95]
Averaging/Voting that requires training of the model combination scheme:	
Bayesian model averaging and Least-squares linear combinations	[96]
Linear combination trained using Perceptron learning rule	[95]
Bayesian Network for model combination	[97]
Evolution of model combination expressions by means of GP	[74], [98], [99]
Sequential training of component models (Section V-B)	
Boosting	[89], [91], [100], [101]
SEARCH OPERATORS AND SELECTION SCHEMES	
Mutation	[102]
Size-fair crossover	[103]
Semantic similarity-based crossover	[104]
Geometric semantic crossover and Geometric semantic mutation	[104], [105]
Gradient descent for optimising constants	[106], [107]
Selection schemes	[108], [109]

By expanding the third term on the right-hand side of (5), we get

$$\mathbb{E}_D\Big[\Big(f^{(D)}(x) - \bar{f}(x)\Big)\big(\bar{f}(x) - g(x)\big)\Big] = \big(\bar{f}(x) - g(x)\big)\mathbb{E}_D\Big[\Big(f^{(D)}(x) - \bar{f}(x)\Big)\Big]. \quad (6)$$

Since $\bar{f}(x)$ and $g(x)$ are constants with respect to D, the term $(\bar{f}(x) - g(x))$ in (6) is a constant and factors out. The expected value of $\mathbb{E}_D[f^{(D)}(x) - \bar{f}(x))]$ is then

$$\mathbb{E}_D\Big[\Big(f^{(D)}(x) - \bar{f}(x)\Big)\Big] = \mathbb{E}_D\Big[f^{(D)}(x)\Big] - \mathbb{E}_D\big[\bar{f}(x)\big] = 0 \quad (7)$$

which is equal to zero since in our earlier definition of average model we have that $\mathbb{E}_D[f^{(D)}(x)] = \bar{f}(x)$. Substituting (6) and (7) into (5), we arrive at a decomposition of the expected generalization error into *bias* and *variance* terms as follows:

$$\mathbb{E}_{x,y}\left[\mathbb{E}_D\left[\left(f^{(D)}(x) - g(x)\right)^2\right]\right]$$
$$= \mathbb{E}_{x,y}\left[\underbrace{\mathbb{E}_D\left[\left(f^{(D)}(x) - \bar{f}(x)\right)^2\right]}_{\text{variance}} + \underbrace{\left(\bar{f}(x) - g(x)\right)^2}_{\text{bias}}\right]. \quad (8)$$

The *bias* term measures the extent to which the hypothesis set, represented at its best by $\bar{f}(x)$, is biased away (i.e., differs) from the target function g. High bias results in underfitting, which is discussed to some extent for the case of GP in [9]. The *variance* arises due to the fact that we have a finite dataset for learning, and measures the extent to which model $f^{(D)}$ changes when trained on different samples D drawn from $P(x, y)$. Overfitting is the result of a low-bias, high-variance model.

An investigation of bias-variance tradeoff in GP is presented in the work of [10]. Experiments based on univariate symbolic regression problems showed that in cases of data abundance, GP is able to approximate the target function to a high degree of accuracy, provided that the expression-tree size is unconstrained. During evolution the bias is asymptotically approaching the optimal level (the level of noise in the target function), while the variance decreases as well. In cases of limited training data sources, GP was empirically shown to be a low-bias, high-variance method. During evolution larger programs lead to a higher variance, yet the bias error term decreases. The use of model averaging by means of bagging [1, Sec. 8.7] was proposed for lowering variance. At the same time, component predictors need to be evolved with the goal of lowering bias. The maximum tree-size plays a major role into this, with an increasing maximum tree-size having the tendency to produce models with lower bias. The optimal maximum size is problem specific, and should depend upon the dimensionality of the input space and the size of the training sample. In addition, pathologies (i.e., undefined output or extreme output values for certain inputs) that may appear in the output of evolved models need to be excluded from model averaging. For that purpose a trimming process is suggested in [10], which removes 10% of the models with highest and lowest predictions given an input.

The following sections survey GP research based on the taxonomy we proposed earlier. We start with *model selection* methods for assessing the ability of a prediction method to generalize, and therefore estimating the optimal amount of model complexity for a given prediction method and training sample.

III. Model Selection

One aspect of the general problem of inference from data given a model is that of *model specification* [110]. It is partitioned into two components: 1) formulation of a set of candidate models and 2) selection of a model to be used for making inferences. GP models for regression and classification take the form of compositions of functions of one or several explanatory variables. In formulating the set of candidate models one needs to specify primitive functions and terminals, model size, and construction constraints (i.e., closure, strong typing, and context-free grammar). The candidates set consists of an enumeration of function compositions given function and terminal sets subject to model size and construction constraints.

The distinguishing characteristic of standard GP compared to its sister SML methods is that it does not assume any particular model form/structure. Given the building blocks of function compositions using function and terminal sets, evolutionary search samples models of variable structure and complexity. Selecting the appropriate model structure along with its parameterization (function composition with the associated explanatory variables and constants) is a challenging task due to the ambiguity caused by the multiplicity of solutions that can be evolved to fit a finite training sample. The decision on the right amount of model complexity, and therefore the selection of the output model from a GP run, needs to be guided by assessing and comparing the generalization performance of models sampled throughout evolution. SML provides two approaches for estimating generalization error. The first approach is to estimate the optimism inherent in the training error and adjust this error accordingly. This gives rise to analytical model selection methods defined for the class of models that are linear in their parameters, for example, Akaike information criterion (AIC), Bayesian information criterion (BIC), structural risk minimization (SRM), and Mallows C_p statistic, an introduction of which is given in [1, Ch. 7]. The second approach uses an independent validation dataset to directly estimate generalization error.

We organize our review of model selection methods in GP into two sections. The first section reviews analytical and validation methods for estimating generalization error and selecting a model as the output of a run. The second section is devoted to feature subset selection methods. By using only a subset of the original set of explanatory variables, is it possible to increase the bias in order to reduce the variance of the predicted values [111], and hence may improve generalization [1, Sec. 3.3].

A. *Assessment of Generalization Performance*

1) Analytical Methods: Analytical methods for estimating generalization error are defined as a function of training error, model complexity, and training sample size. GP researchers have used these methods to select a model out of a set of models sampled during evolution. AIC was used in the work of [11], and was compared against final prediction error (FPE) and predicted residual error sum of squares (PRESS). Models selected with AIC were shown to either outperform or result in similar generalization performance with models selected using FPE or PRESS. The work of [12] evolved polynomial models, and used the *Mallows statistic* to chose the number of polynomial terms from a nondominated set of solutions at the end of a run. The greedy heuristic of *backward elimination* was used to remove statistically insignificant terms from polynomial models in the work of [13]. Backward elimination, which iteratively removes insignificant terms based on student's t-test for statistical significance, was applied at every generation.

2) Validation Methods: Validation methods estimate the error on an independent dataset D_{val}, and use this estimate as a proxy for generalization error. Typically, a number of programs sampled throughout evolution are evaluated on the validation dataset at the end of a run (these are often the best-of-generation programs), with the best one designated as the output. Examples can be found in [18], [20], and [21]. The work in [19] presented several variants of validation-based model selection combined with Pareto-based sorting that takes into account the classification error rate and the program size.

Validation methods can be used to halt an evolutionary run when overfitting is detected, a process known as *early stopping* in the neural networks literature. A number of early stopping heuristics are investigated in [14], [16], and [17], which are based on functions of training and validation error estimates. A heuristic defined as a threshold on the Pearson correlation coefficient calculated on training and validation error-series was additionally used in financial modeling [15].

The work of [23] is a notable case that considered hyper-parameter fine-tuning as part of model selection. A distinction is made between fitting a model on training data and optimizing the model of the learning process. The use of disjoint training samples at every generation was recommended, where each sample is further divided into a dataset used to optimize the hyper-parameters of the evolutionary algorithm, and to a dataset used to evaluate fitness. Models are selected among a small set of best-of-generation candidates based on validation.

B. Feature Subset Selection

Feature subset selection [112] aims at retaining only a subset of the original set of explanatory variables in a model. The purpose is twofold. The first is prediction accuracy: by eliminating redundant or irrelevant variables from a model it is possible to sacrifice some bias in order to lower the variance of the prediction method [1]. Reducing the dimensionality of the input space may allow a model to generalize better in case of sparse datasets. The second is interpretation: we are often interested in determining the subset of variables that exert the strongest effects to the response variable. SML methods for feature subset selection divide into three categories: 1) filter; 2) wrapper; and 3) embedded methods.

Filter subset selection uses statistical, information-theoretic, and distance-based techniques to assign scores to either subsets of features or individual features according to their relevance to the response variable. Features are ranked based on score, and a subset is then selected. A distinguishing characteristic of filter methods is that they process the original feature set independently of a given predictor.

Wrapper methods [111], [113] formulate feature subset selection as a combinatorial optimization problem, and solve it using a search strategy. This can take the form of either an exact algorithm (i.e., leaps-and-bound in [1, Sec. 3.3.1]), or greedy heuristics (i.e., forward selection and backward elimination in [1, Sec. 3.3.2]), or meta heuristic search (a recent survey on the application of evolutionary computation to feature selection is given in [114]). During search, the accuracy of a trained predictor is used to assess the usefulness of a candidate subset, therefore the optimal subset is tightly coupled with a certain class of predictor. Wrapper methods are generally computationally intensive since they require the repetitive invocation of a learning algorithm each time a candidate solution is evaluated, and also need to account for estimation of prediction accuracy using cross-validation [111].

Embedded methods perform feature subset selection during the process of training a predictor. In GP, embedded feature subset selection is based on an evolved model to jointly select a subset and assess its usefulness, whereas wrapper subset selection is based on an evolved model to select a subset and a different predictor to assess its usefulness.

1) Embedded Feature Selection: Feature selection in GP is regarded as the by-product of its variable-length representation and evolutionary selection pressure as discussed in the works of [24] and [31]–[33]. In this sense, GP is inherently an embedded method of feature subset selection. In contrast to greedy forward selection or backward elimination heuristics, evolution is an alternative heuristic that is largely unconstrained in traversing the search space (i.e., add/remove multiple features rather than one), and can re-evaluate previous feature interactions as new interactions are tried [34].

The simplest method to rank individual features is to score them based on their frequency of occurrence in leaf-nodes of best-of-run programs. This practice was adopted in [33], [35], and [36].

In [37], GP is used to evolve expression-trees for binary classification, where features are selected from best-of-run individuals and ranked according to their signal-to-noise ratios. The work in [38] applied the filter methods of information gain and relief-F to preprocess the feature set, and subsequently used GP for embedded feature selection. A combination of embedded feature selection and construction was presented in [39].

Research reported in [40] and [41] applied GP to simultaneously select features and construct a classifier. Before initializing each individual classifier the terminal set is populated with a random feature subset. Crossover is then restricted between programs that were initialized with either the same or similar terminal sets. In [41], an additional crossover operator is applied, in which the best crossover point is identified by enumerating and evaluating all offspring expression-trees resulting from different crossover points given a subtree from the first parent. The fitness function in both [40] and [41] rewards GP classifiers that correctly classify more samples using fewer features.

Stage-wise embedded feature selection uses two stages of independent GP runs to progressively fine-tune feature subsets in high-dimensional problems [31], [34], [43]. Once features are selected in the first stage, GP runs are invoked with a subset of the original features. The work of [34] uses best-of-run programs to select features in the first stage. In [43], the first stage of runs are based on a Pareto-GP system that trades off accuracy for structural complexity, and variable contribution analysis is employed to select features from models that lie in the knee of the Pareto-front. The study of [31] employed the permutation-based feature importance measure of random forests to rank features collected from best-of-run individuals. Finally, the work of [42] divides a run into two stages: during the first stage, a subset of frequently occurring features are extracted from the good-performing portion of the current population, and new individuals are initialized using said subset to replace the average-performing portion of the population.

A number of works [32], [44] employed an adaptive mutation operator to create bias toward certain features. Mutation adaptation is based on normalized feature weights that yield a probability distribution over features of a terminal set; the distribution is used to draw terminal elements during random subtree creation. In [32], feature weights are based on the permutation-based feature ranking of random forests, whereas in [44] feature weights are defined as a function of frequency of occurrence in expression-trees and their respective fitness.

2) Wrapper Feature Selection: Wrapper methods rely on GP to search for feature subsets, but their performance is assessed using a different predictor. The work of [28] evolves programs for hierarchical subset construction. Expression-trees are composed of elementary set operators (union, intersection, and difference) as inner-nodes and randomly generated subsets of features as leaf-nodes. The performance of naive Bayes classifier is used as a wrapper. A similar program representation is presented in [29] that focused on problems with class imbalance. The difference is that leaf-nodes are now represented by feature subsets resulting from the application of filter methods of information gain, χ^2, odds-ratio, and correlation coefficient. A Gaussian naive Bayes classifier is used as a wrapper. Finally, GP was used to evolve a hyper-heuristic that controls the step-wise execution of greedy heuristics for adding and deleting features in the work of [30]. A J48 classifier is used as a wrapper.

3) Filter Feature Selection: Filter subset selection can be applied to individual feature evaluation as well as to feature subset evaluation [115]. For binary classification problems, GP is used to evolve a program that maximizes the square of Pearson correlation coefficient between the output of the program and a function of the categorical class variable [24]. Program output captures the nonlinear interactions between a number of features, therefore the correlation between the output value and a function of the class variable is treated as a proxy for the relevance of said feature subset. A similar framework for removing redundant features that often result in filter-based feature ranking methods is introduced in [25]. Given the complete set of features F, a single feature $x \in F$, and a subset of features $A \subseteq F\backslash\{x\}$, GP is required to evolve a program using subset A that maximizes the square of the Pearson correlation coefficient between x and the program output. In case where the coefficient exceeds a given threshold, feature x is discarded as redundant.

The study of [26] proposed the evolution of programs that were rewarded for partitioning the input space of binary classification problems in a way such that the probability of one class is greater than the probability of the other in a certain interval. This leads to a decrease in conditional entropy of the set of two classes, which signifies that the subset of features used in an evolved program are relevant for discriminating between classes.

The work of [27] proposed a hybrid filter/wrapper approach. The feature subset selection problem in binary classification is formulated as a bi-objective optimization problem of maximizing subset relevance and minimizing subset cardinality, and is tackled with a Pareto-based method. Subset relevance is defined similarly to [24]. The best subset from the resulting Pareto-front maintained in between independent runs is selected with wrapper evaluations using a variety of classifiers.

The research papers that were reviewed in this section studied the use of GP for feature subset selection in cases where a number features in the original set were irrelevant or redundant for predicting the response variable. The work in [116] took a different approach and investigated the scalability of GP as a function of the number of features in problems where all of the features are relevant in the prediction. Results demonstrated that in these cases GP does not scale well, with the number of fitness evaluations required to reach high classification accuracy in synthetic datasets increasing exponentially with the number of features.

C. Critical Discussion

The assessment of generalization performance can be involved into three main aspects of model selection in GP. These are: 1) feature subset selection; 2) selection of an ultimate model out of an evolutionary run (this includes the decision to halt a GP run); and 3) hyper-parameter optimization. Methods for estimating and comparing generalization errors of different GP models sampled throughout evolution is a way of deciding upon the optimal amount of model complexity given the training sample. First and foremost, one should distinguish the problem of model selection from that of assessing the prediction accuracy of the ultimately selected model on unseen data. For that last purpose, practitioners need to set aside an independent test set. The remaining data can be used for training and for performing model selection. For the purposes of model selection, a single validation set or various methods of sample reuse (i.e., cross-validation and bootstrapping) can be employed.

1) Analytical and Validation Methods for Model Selection: The use of analytical methods for assessing generalization performance promises to free us from the necessity of holding-out a validation dataset. These methods are therefore potentially useful in situations where there is insufficient data to use for validation. As an example, training and model selection of polynomials were both performed using all the available data organized in a single set in [12]. The application of analytical model selection criteria has not received much attention from the GP community. One aspect that warrants further investigation is how to best define model complexity, which is a term factored in the analytical formulas. Complexity was set to the expression-tree size in [11] and [61] for AIC and BIC, or to the number of multiplication and division nodes in [61] for SRM. The effectiveness of other complexity measures (examples of these can be found in the following section on regularization) is as of yet undetermined.

On the other hand, in a data-rich scenario, validation-based model selection should be preferred [1, p. 222]. It is easy to implement and use, applies in almost any setting without requiring specific metrics of model complexity, and results in good generalization error estimates in practice. Given a dataset of size N, a typical split is to use $K = N/4$ examples for validation and the remaining $N - K$ examples for training as suggested in [1, p. 222]. Nonetheless, as pointed out in [1] and [112], it is difficult to give a general rule on how much data is enough for each training and validation splits. This primarily depends on the signal-to-noise ratio of the target function, the complexity of the candidate models, and the size N relative to the dimensionality of the input space X.

Furthermore, choosing the data points to use for training and for validation may introduce sampling bias. N-fold or leave-one-out cross-validation [1, p. 241] avoid this issue, but their application to GP is not straightforward as discussed in [16]. Cross-validation relies on the modeling method producing models of similar form for each of the succession of N partitions, and is commonly used in methods where successive models for each fold differ relatively slightly. This is not the case of GP, where it is possible that the N models are very different from each other in structure, in constants, and in the independent variables used. One way to deal with this randomness is to average a number of GP runs for each successive partition, but at the expense of additional computational cost. Therefore, for GP, the division of the original dataset into disjoint, fixed, *train-validation-test* partitions is generally more practical. Data points for each partition should be drawn uniform-randomly.

An additional use of validation methods is to optimize the hyper-parameters of the evolutionary algorithm. Examples of GP system parameters include elements of the primitive function and terminal sets, maximum model size, probability of the application of variation operators, and constraints on syntax/semantics. This aspect of model selection in GP is largely neglected in practice, when at the same time hyper-parameter fine-tuning constitutes an integral part of model induction in SML. There is evidence that evolutionary systems are on average insusceptible to small changes of the probabilities of variation operator application, however, optimally deciding on maximum model size and constituent elements of the function set may turn to be advantageous in certain problems with limited data sources.

The main drawback of validation is the reduction in size of the training set, and as previously mentioned this method is most successful in data-rich domains. Another drawback concerns an improper use of validation. Practice has shown that if the validation set is used to decide on only one or a few parameters, the estimates based on a properly sized validation set will be reliable. The more choices made based on the same validation set, the more "contaminated" the validation set becomes and the less reliable its estimates will be. The number of choices here refer either to the number of parameters that are fine-tuned or the number of times the same validation set was used to affect the learning process during a run. When a large number of choices need to be made then multiple disjoint validation sets are required.

The final remark concerns early stopping heuristics [14], [16], [17]. These are useful in cases where fitness evaluation is very expensive and there is a need to reduce run time. In all other problems, allowing evolution to proceed until convergence is preferred, along with the use of validation at the end of a run for selecting the optimal level of model complexity given the training sample. The parameters that govern early stopping heuristics appear to be problem-dependent [17], and it is seldom known in advance which heuristic and parameterization will perform the best for any given problem. Run termination triggered by the value of Pearson correlation coefficient between training and validation error series appeared to be robust across a wide range of problems and GP system setups [14], [15], [17].

2) Feature Subset Selection: The vast majority of wrapper methods that use evolutionary search employ a direct encoding of solutions, which takes the form of a bit-string representation [114]. That is, for P features, solutions are represented using P bits, where each bit indicates whether a feature is present (1) or absent (0). This results in a combinatorial search space of size 2^P. GP research [28], [29] investigated the use of indirect encodings, where the solution is represented by a program that hierarchically constructs a subset of features using function compositions of elementary set operations. Another indirect encoding was presented in [30] where GP evolved a hyper-heuristic to construct a solution in a stage-wise manner. It may be the case that such indirect encodings offer a compressed representation for search, which is potentially more evolvable than the direct encoding of solutions for very high-dimensional problems. The use of GP indirect encoding in wrapper search and a comparison against direct encoding is a potential area for further research, especially in cases where the original feature set is of very high cardinality.

The decision between the use of filter, wrapper, or embedded subset selection methods needs to take into account the cardinality P of the original feature set relative to the number of examples N. Sophisticated wrapper and embedded methods obtain better predictive accuracy than filter methods in problems where $P \ll N$ [111], and GP used either for wrapper or embedded search should be no exception. On the other hand, for problems where $P \gg N$ or problems with small N, iterated overuse of cross-validated predictor accuracy estimates may be overly optimistic of the performance of certain subset of features on unseen data. In cases where the number of examples is small relative to the number of features then simpler filter selection methods (i.e., methods that consider simple types of relationships like Person correlation coefficient) may be necessary to prevent overfitting [112]. Another possible solution is to preprocess the feature set with a filter approach before passing it to GP as was done in the work of [38]. Alternatively, a greedy heuristic search like step-wise forward selection with a simple linear wrapper [112] can be used to preprocess the feature set before running GP. Step-wise forward selection consists in a far more constrained search than GP, and should result in lower variance. One way to apply embedded feature selection with GP in problems where $P \gg N$ is to adopt a stage-wise evolution of models as in [34]. In order to solve a binary classification problem with 7129 features and 60 examples [34], the first-stage GP runs enabled to trim the original feature set down to 404 features, and the second-stage single-generation runs limited the expression-tree depth to contain no functions or a single IF statement. Yet another way is to introduce evolutionary pressure toward the use of fewer features using a scalar multiobjective fitness function as in the work of [40] and [41], or a Pareto-based multiobjective fitness function as in [27].

The use of GP with embedded feature selection has received considerable attention. The work of [32] demonstrated that best-of-run GP programs often contain features that do not contribute to fitness even when bloat control measures are used. The permutation-based variable importance scheme of random forests was shown to be more useful in identifying relevant features in GP programs. Yet another study reported in [36] showed that the simple frequency-of-occurrence-based scoring performed better than permutation-based scoring in selecting a number of features for use with other classifiers. There is a notable difference in the maximum allowed tree depth of 17 in [32] versus 8 in [36], which suggests that frequency-of-occurrence-based scoring may be sensitive to this parameter. In any case, it was shown that generalization of GP benefits from adapting the probability of selecting features during the application of mutation operator using either the frequency-of-occurrence [44] or the permutation-based variable importance analysis [32].

The last remark concerns the inherent tendency of GP to use subsets with relatively low cardinality as shown in [27]. This is undoubtedly a desirable property of a feature selection algorithm in problems where there exists a subset of features that enables better generalization than the original set of features. By focusing on the smallest possible subsets evolution implicitly addresses the need to eliminate irrelevant and redundant features. Nevertheless, this constitutes a potential weakness in cases of high-dimensional problems with large subsets of relevant features. As pointed out in [27] one should ensure that the optimum size of the subset is not beyond the exploration capability of GP. Toward this end, a dynamic depth limit is used that allowed to increase only if a resulting offspring fulfills certain criteria on training and validation fitness compared to the current best solution. With the proposed method the evolution of depth was controlled, however for problems with very large subsets of useful features very deep expression-trees would be required.

The following section deals with the class of *fitness evaluation* of the proposed taxonomy. This primarily encompasses the use of regularization in GP, as well as additional heuristics in the form of fitness functions, and training data sampling strategies.

IV. Fitness Evaluation

One way in which SML methods address the ambiguity caused by the multiplicity of solutions to the minimization problem of (2) is to restrict the eligible solutions to a smaller set of functions. In general, the constraints imposed can be described as *complexity* restrictions of some form. For the case of GP, these restrictions can be built into the learning method itself either explicitly (i.e., setting the maximum model size, selecting certain elements for the function set, and use of syntax constraints by means of strong typing/grammar), or implicitly by means of a fitness function that exerts bias toward simpler solutions. The augmentation of fitness functions with complexity penalties is the focus of *regularization*, and forms the subject of the first section.

Furthermore, evolution is unique in orchestrating various heuristics for solving the problem of constructing a function composition that maximizes a measure of goodness-of-fit calculated on some finite training sample. The second section deals with such heuristics that are embedded in the fitness evaluation process of individuals. We focus on fitness functions (other than regularized fitness functions), and training data sampling strategies, which were empirically demonstrated to improve the ability of GP to generalize.

A. Regularization

Regularization [1, Chs. 3 and 5] are a family of methods for controlling variance. They constrain the space of eligible solutions by adding a model complexity penalty to the loss function. The penalty function expresses a prior belief that the input–output mapping we search for exhibits a *smooth* behavior in the sense that similar inputs produce similar outputs. The general formulation of regularization in GP takes the form of minimizing an augmented fitness function by adding a regularization term as follows:

$$L_R(f) = L_S(f) + \lambda\Omega(f) \quad (9)$$

where $L_R(\cdot)$ is the regularized fitness function, $L_S(\cdot)$ is the standard loss function (i.e., squared loss to be minimized for real-valued target functions), and $\Omega(\cdot)$ denotes the regularizing function (or regularizer) which is a measure of *complexity* of f. The regularizing term represents a *model complexity penalty function*, the influence of which on function f is controlled by the regularization parameter λ.

Regularizing functions Ω in GP are constructed for *semantic complexity* that characterizes the smoothness of a response surface, and *syntactic complexity* that characterizes the size of a program. The vast majority of regularized fitness functions based on semantic complexity were used to evolve real-valued target functions. Regularized fitness functions that incorporate some measure of syntactic complexity were applied to evolve real-valued target functions is the works of [13], [45], [49], [54], [56], [60], and [61], and categorical-valued target functions in the works of [19], [51]–[53], [55], [58], and [60].

1) Regularizers Based on Semantic Complexity: The first category of papers uses an augmented fitness function that takes the form of a scalar combination of a measure of goodness-of-fit and a regularizer. A regularizer based on curvature (second-order Tikhonov functional) of evolved polynomial models is applied in [13] and [45]. The work of [47] also used curvature as a regularizer; however, programs in this case are composed of protected arithmetic operators. In the work of [48], the first derivative of an evolved function along with the curve length of the first derivative were used as regularizers, while the work of [50] applied a regularizer based on an approximation of the VC-dimension.

The second category of papers uses Pareto-based multiobjective optimization between a goodness-of-fit objective and a regularizer objective. A number of different regularizers based on zeroth-order, first-order, and second-order derivatives of an evolved function with respect to the independent variable are studied in the work of [46]. A method of numerical differentiation based on Lagrange interpolation was used to compute derivatives. The analytic quotient operator replaced protected division to ensure that pathologies (i.e., undefined output or extreme output values for certain inputs) in an evolved function's output are eliminated. A regularizer based on the *order of nonlinearity* was introduced in the work of [49]. Order of nonlinearity is defined as the minimum degree of the best-fit *Chebyshev* polynomial approximating an evolved program with a certain precision.

2) Regularizers Based on Syntactic Complexity: Two of the earliest studies about the effect of program size on generalization are presented in [57] and [58]. Programs that took the form of decision-trees were evolved to control an agent for the game of Ms. Pacman. Experiments demonstrated a relationship between size and generalization, with smaller programs performing better than larger ones. The study in [58] similarly showed a positive correlation between the size of classification programs and the generalization error.

Regularized fitness functions that penalized program size measured in terms of tree-node count were reported in [52] and [55]. In those studies, the regularization parameter λ is kept constant throughout evolution. In the work of [56], this parameter is dynamically adapted using the method of *covariant parsimony pressure*. A penalty function of expression-tree size and depth is adopted in [59], while Bojarczuk *et al.* [52] used a regularizer as a function of expression-tree size and maximum-allowed size.

A number of studies have applied Pareto-based multiobjective optimization of goodness-of-fit and program size [19], [46], [51], [53], [54]. The work of [49] used *expressional complexity* (total number of tree-nodes in all constituent subtrees of a program) as syntactic complexity objective in Pareto-based optimization.

Fitness functions based on minimum description length (MDL) [1, Sec. 7.8] encourage syntactic simplicity of evolved programs. An MDL-based fitness function was used in [60] for evolving decision-trees, and in [59] for evolving neural trees. MDL was additionally combined with a curvature-based complexity penalty in the works of [13] and [45] for evolving polynomial models.

Finally, the previous research [61] studied symbolic regression using loss functions based on the methods of AIC, BIC, and SRM. For the cases of AIC and BIC, model complexity was measured in terms of expression-tree size, while for the case of SRM model complexity was set to the number of multiplication and division nodes of an expression-tree. It was empirically shown that SRM outperformed both AIC and BIC fitness functions.

B. Other Fitness Functions

A fundamental feature of evolutionary learning is that it is based on a gradient-free optimization process. This alleviates the need for fitness functions and evolved function compositions that are continuous and differentiable as is the requirement for models trained end-to-end with gradient-based optimization methods. There exist a number of alternative fitness functions to the regularized ones that implicitly restrict the size of the search space by exerting pressure toward certain solutions. These fitness functions have the potential to improve generalization.

A number of variations on an adaptive weighted average defined as $(1/N)\sum_{i=1}^{N} w_i L(f(x_i), y_i)$ were proposed as fitness functions in [20], [62], [63], [75], and [76]. Weight w_i reflects the influence of individual loss $L(f(x_i), y_i)$ to the weighted average calculated over the training sample $\{(x_i, y_i)\}_{i=1}^{N}$. In [62], a weight w is dependent on the level of ruggedness of evolved f around input x. In [63], a weight quantifies the importance of an input point x relative to rest of the input points in the training sample. Different measures of sampling density around individual training examples are used as weights, which assign higher influence to sparse areas of the input space. Given input x and its q nearest neighbors in the training sample (using a fractional or Euclidean distance metric), measures of density are defined based on proximity, surrounding, remoteness, and nonlinear deviation. The study of [75] uses weights to increase the influence of misclassified training examples, while the work of [20] proposed a sharing method that adaptively weighs the loss on individual examples according to the performance of the rest of the population on said examples. The fitness function in [76], defined for binary classification problems with a decision boundary $f(x) = 0$, weighs the contribution of misclassifications according to the hyper-volume of the portion of a kernel function that falls on the wrong side of the decision surface.

The studies of [64] and [65] applied bootstrapping [1, Sec. 7.11] to estimate the standard deviation of a statistic that measures goodness-of-fit, and factored this estimate into a fitness function. The fitness function in [64] takes the form of a weighted average between bootstrap standard deviation of Canberra distance between target and predicted values, and Canberra distance on the original training sample. The work of [65] uses the product of bootstrap standard deviation of classification error rate by classification error rate on the original training sample.

A number of fitness functions for classification were devised to address poor prediction performance on unseen data that often results when evolving discriminant functions with fixed thresholds on their output values to determine class boundaries [117]. A fitness function for multiclass classification problems that measures the separability of probability distributions of program-outputs for different classes is presented in [68], and is additionally applied to binary decomposition of the multiclass problem in [69], and to unbalanced classification problems in [70]. A similar fitness objective defined as the intersection area resulting by the projection of program-output distributions for different classes is introduced in [71]. Finally, a misclassification cost ratio defined as a function of false positives and false negatives is used as the basis of a method to dynamically determine the threshold representing class boundaries in the work of [72].

Additional fitness functions for classification problems are based on approximations of area under curve (AUC) measure [70], [73], [74], [98] and maximum-margin linear discriminants [67]. For regression problems, Pareto-based multiobjective optimization is applied in [54] with objectives defined for root mean squared error, variance of residuals $y - f(x)$, and program size.

The aforementioned fitness functions are general in the sense that they can be applied to arbitrary regression and classification problem domains. Examples of fitness functions developed for certain application areas are as follows. Medical classification was tackled with a fitness function based on sensitivity, specificity, and program size in [52], while the evolution of financial trading rules used a fitness function defined as a weighted sum of daily mean return and *p*-value generated from a statistical hypothesis test named superior predictive ability in the work of [18]. For object detection, a fitness function based on detection rate, false alarm rate, false alarm area, and program size is reported in [66].

C. Data Sampling Methods

Early work on GP generalization showed that repeatedly exposing a population to a fixed training sample for a large number of generations can result in solutions with poor generalization [3]. Dynamic sampling of training examples as opposed to a fixed static training sample can improve generalization by reducing variance.

Given a set of labeled examples, the training sample can be generated in different ways. The first approach is to down-sample the original set of examples uniform-randomly (with or without replacement) at predefined intervals throughout evolution. The second approach is to perform adaptive sampling. We distinguish between two major classes of methods of adaptive sampling: 1) down-sampling based on difficulty of a training example where the sample size is kept fixed throughout a run and 2) incremental or layered sampling where an initial subsample monotonically increases throughout a run until it accommodates every training example in the original set. Methods for adaptive sampling based on difficulty can be further divided into those based on single-population evolution, and those based on competitive coevolution of training examples versus programs.

The works of [55], [77], and [80] used a random sample of training examples $S < N$ to control overfitting in classification problems. Fitness was evaluated on a single randomly drawn example [79], or a version that periodically balances the random selection of a simple example with the use of the complete training sample [81]. Randomization of the training sample individually for each program in each generation was investigated in [20].

Dynamic subset selection introduced in [78] uses the performance of the current GP population to select a new subset of difficult examples (examples which are frequently misclassified) and infrequent examples (the age of an example relates to the number of generations since it was last selected) at every generation. Hierarchical dynamic subset selection, developed to improve training time in the case of large datasets, was applied to anomaly detection in [82]. Historical subset selection [78] uses a number GP runs to establish some measure of the difficulty of each training example. Over the course of several runs the cases misclassified by the best population member are recorded, and make up the subset used in subsequent GP runs (the subset remains fixed after its initial selection). In the work of [85], training examples are associated with a score that is incremented each time an example is misclassified and decremented otherwise (higher score means “fitter” example). Upon every individual fitness evaluation a two-member score-based tournament selection over the complete training sample is applied to generate a subsample of examples.

Competitive coevolution of a population of programs versus a population of training examples is presented in the works of [20], [83], and [84]. For regression problems, training sets of size 20 bred with one-point crossover and Gaussian mutation in [20], while the work in [83] used a single example that was perturbed using Gaussian mutation. The work of [84] uses a subsample of eight examples, which is encoded as an array of indexes to the full training set. Each index is allowed to repeatedly sample a point if necessary.

One form of incremental learning in GP uses a layered training strategy based on a sequence of nested training samples of increasing size for each layer [59], [87], [88] until the complete set of training examples is made of use. The training process in [87] starts with a basic set of training examples, and an increment in the number of examples is triggered when enough hits are scored for a particular training set configuration. The work of [59] proposed the method of incremental data inheritance, where each individual program in a population is assigned its own variable-length training sample that is initialized at a size of 20 and grows by an arbitrary increment of 6 additional examples per generation. Training samples are inherited and recombined using a variant of uniform crossover. The study in [88] addressed the problem of optimally choosing the size and composition of initial sample, which was based on the Kullback–Leibler divergence between a variable-sized sample and the complete set of training examples. Evolution proceeds, and a mechanism of overfitting detection triggers sample augmentation, which follows a geometric progression.

A different approach to layered learning is presented in [86] based on a method that enables a gradual complexification of training samples of fixed size as opposed to gradual sample-size augmentation used in research work discussed previously [59], [87], [88]. A measure of complexity of a training sample based on the variance of program outputs controls a transformation of training samples of increasing complexity. The population of evolved programs is exposed to a training sample until overfitting is detected or a generation limit is reached at which point a consecutive training sample is used.

D. Critical Discussion

1) Regularization: Regularized fitness functions are multiobjective. The scalar weighted average of (9) is easier to fine-tune if the values for the objectives of goodness-of-fit and regularizer are of same scale, therefore some form of normalization is necessary. The use of validation to optimize regularization parameter λ is recommended, which increases the number of runs according to the number of values for λ that need to be validated. On the contrary Pareto-based optimization uses the raw objectives' values, and model selection can be directly performed among models composing the Pareto-front.

For models evolved without any constraints on their functional form, computing the value of a semantic complexity penalty based on Tikhonov functional [46], [47], curve length of first derivative [48], order of nonlinearity [49], and VC-dimension [50] adds an additional cost to fitness evaluation. In the case of [48], only a subset of the population that performed well on the nonregularized version of the fitness function were chosen to be evaluated with the regularized fitness function due to high computational cost of computing the curve length of first derivative. For arbitrary model forms, the use of regularizers like Tikhonov functional [46], [47] or curve length of first derivative [48] require that the evolved programs represent differentiable function compositions, therefore the function sets need to be constrained to certain primitives. The use of Tikhonov regularization in [46] requires numerical differentiation, whereas Tikhonov regularization of polynomial models in [45] is based on partial derivatives that are computed analytically, and it is faster. Finally, syntactic regularizers defined as functions of program size and/or depth [51], [56], [58] yield virtually no extra computational cost, however their impact on generalization is not consistent as will be discussed later in this paper.

The semantic regularizer based on order of nonlinearity [49] does not always outperform a syntactic regularizer based on expressional complexity [49]. Regularization based on a combination of Tikhonov functional and program size consistently outperforms regularization based solely on program size in [46]. Furthermore, regularizers based on VC-dimension [50] and curve length of first derivative [48] are also shown to consistently improve generalization compared to nonregularized fitness functions.

The evolution of polynomial models [45] is amenable to a composite learning process that applies GP-based variation operators for evolving the hierarchical structure of the polynomial model, followed by regularized least-squares fitting of the coefficients of polynomial terms. In addition, polynomial models are amenable to analytical model selection using MDL, where the number of effective model parameters is set to the number of polynomial terms [45]. This waives the requirement of a hold-out set for validation, and is advantageous in cases of small datasets. On the other hand, polynomials are often limited by their global nature meaning that tuning the coefficients to achieve a functional form in one region of the input space can cause the function to vary wildly in remote regions. In addition, as shown in [1, Ch. 5], erratic polynomial fits often result near the input space boundaries, and extrapolation can be problematic.

The relationship between program size and generalization has received considerable attention. In GP representations of decision trees, smaller size is seen with better generalization [57], [59], [60]. In symbolic regression results are inconsistent. The works of [51], [56], and [58] reported better generalization with smaller expression-trees, the works of [54], [55], [103], [108], and [118]–[121] showed weak relationship between size and generalization, and finally the work of [122] reported that good-generalizing classifiers were not parsimonious. There is certainly a relationship between program representation, size, and generalization, albeit a complex, not well-understood one. The MDL principle is supporting the fact that smaller programs should generalize better.

2) Other Fitness Functions: Multiobjective fitness functions that are based on scalar weighted averages of different objectives (i.e., [18] and [64]) require fine-tuning of the mixing coefficients to yield the right tradeoff given a data sample, which should be performed by means of validation. The cost of model selection in terms of independent runs is proportionate to coefficients configurations that need to be validated. On the contrary, Pareto-based multiobjective methods (i.e., [54]) are free of mixing coefficients and can work with raw objectives' values.

Bootstrapping is applied to estimate the standard deviation of a goodness-of-fit statistic in [64] and [65], and to approximate the sampling distribution of a test statistic in [18]. This process results in a significant computation overhead, therefore these methods may be more suitable for cases of limited data sources.

Simple functions of the elements of a *confusion matrix* (i.e., sensitivity and specificity in [52] and true positive rate and true negative rate in [66]) are fast to compute and may form better alternatives to classification error rate (or classification accuracy) in cost-sensitive classification problems like medical diagnosis [52] or object detection [66]. The class of fitness functions based on approximations of the AUC are invariant to unbalanced class distributions, and outperform fitness functions based on standard and weighted variants of classification accuracy [70]. They are also suitable when GP is applied to fuse pretrained classifiers [74], [98]. Approximating the AUC requires that the true positive rate and false positive rate are estimated at multiple thresholds on the output value of an evolved discriminant function [70]. For that purpose, each classifier is evaluated at every threshold value, and there exists a tradeoff between the accuracy of approximation of the receiver operator characteristic (ROC) curve and training time. A faster alternative estimator of AUC that does not require construction of ROC curve is based on the Wilcoxon–Mann–Whitney statistic, and was applied in the works of [70] and [73].

The fitness function that measures separability between the program-output probability distributions for different classes [68] is among the state-of-the-art in the evolution of real-valued discriminant functions for multicategory classification. It is also shown to be a good alternative to AUC-based fitness function in class imbalance problems [70].

The fitness function developed for problems with sparse data in certain parts of the input space [63] requires that individual training example weights are computed at a preprocessing step before program evolution begins. A parameter that needs to be fine-tuned by means of validation is the number of neighbors for computing the different measures of sampling density around individual training examples (Vladislavleva *et al.* [63] proposed to set this to $P+1$ for a P-dimensional input space). Weights may also be used to compress the original training sample to a smaller sample of similar information content, and therefore reduce training times.

Stepwise adaptation of weights [75] (with weights reflecting the difficulty of misclassifying individual examples) requires two parameters to be fine-tuned using validation: the first is the weight update interval and the second is the magnitude of weight updates. The method incurs an additional computational cost having to re-evaluate the population after an update of weights. The fitness function based on vicinal risk minimization [76] requires fine-tuning of parameters for the widths of the kernel functions, and may be expensive for large datasets. This cost comes with an improvement in generalization compared to the discrete fitness function based on zero-one loss.

3) Data Sampling Methods: Strategies for uniform-random or adaptive downsampling of the original training sample reduce the computational cost of GP runs (i.e., [78] and [82]), and is also possible to yield generalization gains (i.e., [20], [55], [77], [79]–[81], and [84]). Gradual augmentation of training sample is positively affecting the speed of the training process [59], [88], but there is limited evidence for its impact on generalization [87], [88].

When deciding on the use of a downsampling method one needs to take into account the additional parameters introduced and the need to fine-tune these in the model selection sense. This will inevitably increase the number of independent GP runs required and necessitates the use of validation as discussed in Section III-C, which may be prohibitive in cases of limited data sources. In general, the more parameters that need to be fine-tuned the greater the computational overhead. Specifically, the use of dynamic subset selection [78] has three parameters (subsample size, difficulty exponent, and age exponent), while uniform-random downsampling has one parameter (subsample size). Reducing the sample size to a single example [79], [81] requires no parameters; however, this sampling technique results in fitness evaluations that are noisy. This may compromise selection, and therefore convergence of GP. In addition, the number of fitness evaluations required to generalize may be more difficult to determine as opposed to runs with uniform-random subsamples of larger size. The most recent study on variants of uniform-random downsampling in [81] demonstrated that the training strategy of alternating between the use of a single random example and the use of the complete training sample is consecutive generations resulted in better generalization than training with a single random example at every generation.

Methods for incremental learning are in general more difficult to fine-tune than uniform-random or adaptive downsampling methods. Common to the methods of [86] and [88] is an overfitting detection mechanism that triggers a transition to a consecutive training sample. It is based on a heuristic that uses validation data to compare the performance of programs sampled in successive generations. Care must be taken when the same validation set is used multiple times throughout a run, as this may bias the error estimates. In addition, the selection of values for the number of layers, and the transformation exponent is required in [86]. The method of [88] is easier to fine-tune, requiring only a stopping criterion for each layer.

Coevolution is a more complex system than single-population evolution, thus it is expected to require more effort to fine-tune its configuration. Nonetheless, for the case of symbolic regression there is some evidence that competitive coevolution [20], [83], [84] is a good alternative to the schemes of uniform-random downsampling and dynamic subset selection. More research is required to ascertain the conditions that make one class of methods preferable over the other.

As a final remark, methods of adaptive downsampling based solely on difficulty of individual examples may be susceptible to overfitting in cases where there is high level of noise or outliers in the training examples, and programs are evolved with fitness functions that are not robust to noise/outliers (i.e., mean squared error for regression). In cases of competitive coevolution, the work of [85] proposed a two-member tournament selection of training examples to counteract the tendency of fitness-proportional selection to quickly skew the distribution of selected training examples toward fitter examples. The use of the age heuristic in dynamic subset selection [78] plays a similar role in sampling more uniformly throughout the course of evolution. The issues of noise and overfitting in adaptive downsampling based on difficulty warrants more investigation.

V. Model Averaging

Model averaging are a class methods that build a composite predictor by combining multiple component predictors [1]. We distinguish between two dominant classes of methods for constructing component predictors. Central to both classes is the notion that component predictors must be diverse in their construction mechanism. The aim is to construct accurate component models, which are uncorrelated in their predictions.

A. Parallel Training of Component Models

In the first category, component models are built in parallel using a *perturb-and-combine* principle. Every component model may be trained on different bootstrap-sampled versions of the training data, or by means of perturbations in the construction method (i.e., available subsets of features and hyper-parameter configuration). This kind of perturbation in data or construction method causes different models to be built if the class of models are of high variance. The resulting models are then combined into a single predictor via some form of voting for classification, or some form of averaging for regression.

Bagging is a technique that works especially well for reducing the variance of low-bias high-variance models evolved by GP. By using an *average* model for making predictions, the error contribution due to variance is effectively eliminated, and the remainder error contribution is due to bias. Bagging uses bootstrap training samples to train different component models. Predictions are then combined using a simple unweighted average for regression, or a simple majority vote for classification (ties are broken arbitrarily). Examples of bagging GP predictors can be found in [10] and [89]–[93]. Despite bagging, a number of different model averaging/voting schemes that do not require training of the model combination scheme are compared in the works of [92], [94], and [95] (more details on these are given in Table I).

The model combination scheme can be itself trained. The perceptron learning rule was used to fit the weights in the linear combination of models in [95]. The work of [96] applied Bayesian model averaging, with weights proportional to the posterior probability of each component model. This was compared against least-squares fitting of weights in the linear combination of models. Evolution of Bayesian networks for selection and combination of GP models was proposed for classification problems in [97].

More complex combination schemes can be evolved in the form of expression-trees that use the outputs of pre-trained component models as leaf-nodes. Decision trees and multilayer perceptrons are fused by means of GP in [98], while Langdon and Buxton [74] evolved expressions to fuse linear discriminants or naive Bayes classifiers. In [99], GP is used to evolve a Pareto-front of component classifiers that trade-off classification accuracies between minority and majority classes. GP was subsequently applied to evolve a combination rule using classifiers from the Pareto-front.

Component models that are built in parallel do not solely rely on data perturbation by means of bootstrapping to de-correlate component predictions. Diversity can be promoted by means of negative correlation learning and pairwise failure crediting in a Pareto-based fitness function [123], island architectures [124], fitness sharing [96], cooperative coevolution [125], and edit-distance between expression-trees [92].

B. Sequential Training of Component Models

The second class of methods for constructing component predictors comes by the name of boosting. A sequence of component models $\{f_t\}_{t=1}^{T}$ evolves over discrete time t such that each successive model concentrates on those training examples that are "difficult" for the previous models in the sequence. A weight is associated with each training example. During training of each f_t in the sequence, weights are increased for those examples that are misclassified by f_{t-1} (in classification problems [100]) or have large absolute deviation from a target value using f_{t-1} (in regression problems [89], [101]). Difficult examples are assigned an ever-increasing influence with additional boosting iterations.

Boosting of GP models is realized in two ways. The first way is to normalize the weights to create a probability distribution over training examples. This distribution can be used to sample examples with replacement at each boosting iteration as in the work of [89]. The outputs of component models are combined using the geometric median weighted by confidence coefficients. The second way is to use the original training sample [101] or to perform uniform-random downsampling (with replacement) [91], [100], along with a fitness function defined as a weighted average of losses on individual examples, with weights reflecting difficulty as described above. The outputs of component models in [101] are combined using the geometric median as in [89], whereas a weighted majority vote that gives higher influence to more accurate component models in the sequence is used in [91] and [100]. A GP-based parallel implementation of AdaBoost.M1 that deals with binary classification problems is reported in [100], whereas the work of [91] implements a parallel version of AdaBoost.M2 to tackle multicategory classification problems.

C. Critical Discussion

Model averaging methods improve the generalization of low-bias high-variance GP models, but at the expense of additional computation required to train multiple component models. Bagging or boosting of GP models may take advantage of the inherent randomness in the construction of models to produce diverse models in the course of independent runs, or select diverse models from a Pareto-front of a single run.

Bagging impacts the performance of highly nonlinear models the most, while it does not affect linear models [1]. Therefore, one should refrain from placing stringent model complexity restrictions when using bagging; a view that is also supported by Keijzer and Babovic [10]. When comparing bagging to boosting GP models, generalization performance is often problem-specific with both methods producing competitive solutions. In terms of training time, bagging is faster since component models can be trained in parallel as opposed to boosting where models are trained in sequence.

Model selection deals with the optimization of additional parameters for the number of component models in bagging or iterations in boosting, and for the maximum size of expression-trees. Another aspect of model selection in committees of models is the optimal pruning by means of removing component models with a goal of improving speed of execution without sacrificing generalization, which was investigated in [126]. Bayesian model averaging, applied to GP in [96], is an alternative approach that can lift the ambiguity in regard to which component models to include in a committee.

Boosting is less robust to noise in the training examples as opposed to bagging. This is mainly because of the weighted voting mechanism of boosting placing higher influence to more accurate component models in the sequence, as opposed to simple majority voting used in bagging. One way to improve generalization performance and computational efficiency in these situations is to select a subsample of training examples uniform-randomly at each boosting iteration [1, Sec. 10.12.2]. In this way, it can be possible to reduce both bias and variance simultaneously. An example of this is found in the works of [91] and [100] that use downsampling of training examples at each boosting iteration. In addition, this practice combined with a parallel implementation of the evolutionary algorithm [91], [100] can reduce training time and memory footprint in cases of large datasets.

Methods for combining component model outputs based on a trainable combination scheme (i.e., the weights are obtained using the perceptron learning rule in [95], or using least-squares linear regression in [96]) need in principle an additional independent training sample for this purpose, or they may run the risk of severe overfitting in cases of limited data sources. Stacking [1, Sec. 8.8] proposes the use of leave-one-out cross-validation for this purpose, which is often not practical for GP due to computational cost. The use of trainable combination schemes such as those studied in [74], [95], [96], and [98] should be preferred in data-rich cases.

The final remark concerns the drawback of model averaging on model interpretability. Single models evolved by GP are highly interpretable, whereas weighted combinations of models lose this important feature. Simple majority voting in case of bagging has the potential of retaining part of the white-box property of GP.

VI. Search Operators and Selection Heuristics

In this section, we draw attention to the generalization effect of additional heuristics that drive evolutionary search; those of search operators and selection methods. Research on these topics is limited.

Increased mutation rate improved generalization in [102]. Semantic similarity-based crossover was studied in [104], geometric semantic mutation was investigated in [127], and geometric semantic crossover with angle-aware mating was proposed in [105]. All studies showed improved prediction ability on unseen data using standard GP as baseline performance. Size-fair crossover was applied to classification in [103], where it was shown to control overfitting to some extent.

A mixed search strategy that combines standard subtree crossover and mutation operators with the optimization method of gradient descent based on partial derivatives of a squared loss function with respect to constants (leaf-nodes) in an expression-tree was investigated in [106] and [107], and was shown to improve generalization performance for symbolic regression [107] and classification [106].

The effect of selection heuristics was studied in [108] and [109]. In [109], tournament participants are selected based on their structural dissimilarity compared to individuals with bad generalization stored in a *tabu* list that is updated throughout evolution. The work of [108] introduced a tournament selection scheme that considers both training error and variance of program-outputs as a measure of program complexity.

A. Critical Discussion

Fine-tuning the mutation rate in an evolutionary run is simpler to fine-tuning variants of semantic crossover and mutation. It is as of yet undetermined how standard mutation combined with regularization and model selection will perform against the more complex semantic operators. Size-fair crossover [103] requires very little fine-tuning, but increases the computational cost compared to standard mutation because of the need to exchange similar-sized subtrees. Semantic similarity-based crossover [104] is computationally inefficient compared to geometric semantic crossover [127] due to the trial-and-error process required to identify semantically similar subtrees. Geometric semantic crossover is shown to improve generalization in [127], however at the expense of interpretability of solutions since the evolved programs are difficult to visualize. The use of angle-aware mating scheme in [105] improves the exploration ability of geometric semantic crossover [127] but at an increased computational cost for choosing the parents.

The use of selection heuristics in the works of [108] and [109] come with additional data requirements since the selection process itself uses two disjoint sets for evaluating fitness. Their application may be more suitable in cases of large datasets.

The application of gradient descent [106], [107] for optimizing constant values in evolved programs is a useful complement to the search ability of GP. A potential drawback is that gradient descent requires a loss function that is differentiable, which precludes the use of various fitness functions that were discussed in Section IV-B. Squared loss is employed in both studies of [106] and [107]. In cases of regression or binary classification with $f(x) = 0$ as the class boundary, such loss function can be effective. In case of multicategory classification, generalization of real-valued discriminants has been shown to suffer when the number of classes is large, therefore one would have to opt for binary classification schemes such as one-versus-rest.

VII. Conclusion

We identified two broad families of methods of SML for addressing the fundamental tradeoff of approximation on training data versus generalization on unseen data. These are regularization and model averaging. Regularization are a class of methods for placing restrictions on the eligible solutions to the inverse ill-posed problem of finding a function that minimizes the error on a finite training sample. Regularization in GP takes the form of fitness functions that incorporate a model complexity penalty; this creates evolutionary pressure toward simpler solutions. Model averaging methods work by combining the predictions of a committee of evolved component models. Regularization methods sacrifice some bias in order to reduce variance. Model averaging in GP works primarily by reducing variance, leaving the bias effectively unaltered.

Furthermore, evolution is unique in orchestrating heuristics that were designed with the aim of improving generalization. These heuristics come in the form of fitness functions, training data sampling methods, and search operators and selection schemes.

All of the aforementioned methods work in synergy with model selection whose goal is to estimate the generalization performance of different models in order to chose the best one. The term model here may refer to different entities, for example, an ultimately selected program or a committee of programs for deployment, a subset of features, or evolutionary algorithm hyper-parameters.

A brief note on the application of GP in a production environment is as follows. There are two primary data-centric factors that govern the preference of one model over another: these are the signal-to-noise ratio of the target function and the size of the training sample N relative to the dimensionality of the input space P. In any case, one should ensure that a diverse set of models that tradeoff accuracy on training data for model complexity are generated through a number of independent GP runs. Restrictions in the form of regularization can help constrain GP search. A form of model selection can then be applied to chose a committee of models for making predictions. We recommend combining the predictions of multiple models as opposed to the use of a single model since this can help alleviate some of the ambiguity caused by the multiplicity of solutions given a training sample. In cases of small datasets or in cases where $P \gg N$, validation based on a single dataset may not be effective, and one should resort on some form of cross-validation. Several techniques for high-dimensional problems that follow the "less fitting is better" principle are presented in [1, Ch. 18]. An additional issue that needs to be addressed is that of undefined or excessively large output of evolved models caused by asymptotes in sparsely sampled areas of the input space. The use of interval arithmetic [128], or the replacement of division operator with analytic quotient [46] are candidate remedies to this problem.

We believe that this survey presented a representative sample of good practice of methods that enable GP to generalize. We hope that this encourages more focused research, and stimulates the field to define better benchmarking standards and problem suites for assessing generalization. The GP community needs to continue work on benchmarking that: 1) is based on realistic problems; 2) measures generalization performance on an independent test set; and 3) performs comparisons between GP and different SML algorithms. Initial work on benchmark problems is found in [129] for symbolic regression.

Finally, we hope that this survey paper can help stimulate interest in developing hybrid systems between GP and other SML algorithms. Examples of such hybrids are the evolution of kernel functions for support vector machines [130], kernel nearest neighbor classifier [131], and kernel regression [132]; the evolution of recurrent neural networks [133]; and the construction of features for symbolic classifiers [134], [135].

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer-Verlag, 2009.

[2] R. Poli, W. B. Langdon, and N. F. McPhee. (2008). *A Field Guide to Genetic Programming*. [Online]. Available: http://www.gp-field-guide.org.uk

[3] I. Kushchu, "Genetic programming and evolutionary generalization," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 431–442, Oct. 2002.

[4] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[5] K. Kim, Y. Shan, X. H. Nguyen, and R. I. McKay, "Probabilistic model building in genetic programming: A critical review," *Genet. Program. Evol. Mach.*, vol. 15, no. 2, pp. 115–167, Jun. 2014.

[6] Y. Hasegawa and H. Iba, "A Bayesian network approach to program generation," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 750–764, Dec. 2008.

[7] K. Kim, R. I. B. McKay, and N. X. Hoai, "Recursion-based biases in stochastic grammar model genetic programming," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 81–95, Feb. 2016.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford Univ. Press, 1996.

[9] S.-H. Chen and T.-W. Kuo, "Overfitting or poor learning: A critique of current financial applications of GP," in *Proc. Genet. Program. (EuroGP)*, vol. 2610. Essex, U.K., Apr. 2003, pp. 34–46.

[10] M. Keijzer and V. Babovic, "Genetic programming, ensemble methods and the bias/variance tradeoff—Introductory investigations," in *Proc. Genet. Program. (EuroGP)*, vol. 1802. Edinburgh, U.K., Apr. 2000, pp. 76–90.

[11] A. Garg, S. Sriram, and K. Tai, "Empirical analysis of model selection criteria for genetic programming in modeling of time series system," in *Proc. IEEE Conf. Comput. Intell. Financ. Eng. Econ.*, Singapore, 2003, pp. 90–94.

[12] J. W. Davidson, D. A. Savic, and G. A. Walters, "Rainfall runoff modeling using a new polynomial regression method," in *Proc. 4th Int. Conf. Hydroinformat.*, Iowa City, IA, USA, Jul. 2000.

[13] K. Y. Chan, C. K. Kwong, T. S. Dillon, and Y. C. Tsim, "Reducing overfitting in manufacturing process modeling using a backward elimination based genetic programming," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1648–1656, 2011.

[14] J. Fitzgerald and C. Ryan, "Validation sets for evolutionary curtailment with improved generalisation," in *Proc. 5th Int. Conf. Converg. Hybrid Inf. Technol.*, vol. 6935. Daejeon, South Korea, Sep. 2011, pp. 282–289.

[15] C. Neely, P. Weller, and R. Dittmar, "Is technical analysis in the foreign exchange market profitable? A genetic programming approach," *J. Financ. Quant. Anal.*, vol. 32, no. 4, pp. 405–426, 1997.

[16] J. J. Rowland, "Generalisation and model selection in supervised learning with evolutionary computation," in *Applications of Evolutionary Computing* (LNCS 2611). Heidelberg, Germany: Springer-Verlag, Apr. 2003, pp. 119–130.

[17] C. Tuite, A. Agapitos, M. O'Neill, and A. Brabazon, "Tackling overfitting in evolutionary-driven financial model induction," in *Natural Computing in Computational Finance (Volume 4)* (Studies in Computational Intelligence), vol. 380. Heidelberg, Germany: Springer, 2012, ch. 8, pp. 141–161.

[18] A. Agapitos, M. O'Neill, and A. Brabazon, "Evolutionary learning of technical trading rules without data-mining bias," in *Proc. 11th Int. Conf. Parallel Problem Solving Nat.*, vol. 6238. Kraków, Poland, Sep. 2010, pp. 294–303.

[19] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini, "Genetic programming, validation sets, and parsimony pressure," in *Proc. 9th Eur. Conf. Genet. Program.*, vol. 3905. Budapest, Hungary, Apr. 2006, pp. 109–120.

[20] L. Panait and S. Luke, "Methods for evolving robust programs," in *Proc. Genet. Evol. Comput. Conf.*, vol. 2724, Jul. 2003, pp. 1740–1751.

[21] D. Robilliard and C. Fonlupt, "Backwarding: An overfitting control for genetic programming in a remote sensing application," in *Proc. 5th Int. Conf. Artif. Evol. (EA)*, vol. 2310. Creusot, France, Oct. 2001, pp. 245–254.

[22] J. J. Rowland, "Model selection methodology in supervised learning with evolutionary computation," *Biosystems*, vol. 72, nos. 1–2, pp. 187–196, Nov. 2003.

[23] C. Igel, "A note on generalization loss when evolving adaptive pattern recognition systems," *IEEE Trans. Evol. Comput.*, vol. 17, no. 3, pp. 345–352, Jun. 2013.

[24] K. Neshatian and M. Zhang, "Genetic programming for feature subset ranking in binary classification problems," in *Proc. 12th Eur. Conf. Genet. Program. (EuroGP)*, vol. 5481. Tübingen, Germany, Apr. 2009, pp. 121–132.

[25] K. Neshatian and M. Zhang, "Unsupervised elimination of redundant features using genetic programming," in *Proc. 22nd Aust. Joint Conf. Artif. Intell. (AI)*, vol. 5866. Melbourne, VIC, Australia, Dec. 2009, pp. 432–442.

[26] K. Neshatian and M. Zhang, "Improving relevance measures using genetic programming," in *Proc. 15th Eur. Conf. Genet. Program. (EuroGP)*, vol. 7244. Málaga, Spain, Apr. 2012, pp. 97–108.

[27] K. Neshatian and M. Zhang, "Pareto front feature selection: Using Genetic programming to explore feature space," in *Proc. 11th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Montreal, QC, Canada, Jul. 2009, pp. 1027–1034.

[28] K. Neshatian and M. Zhang, "Dimensionality reduction in face detection: A genetic programming approach," in *Proc. 24th Int. Conf. Image Vis. Comput. New Zealand (IVCNZ)*, Wellington, New Zealand, Nov. 2009, pp. 391–396.

[29] I. Sandin *et al.*, "Aggressive and effective feature selection using genetic programming," in *Proc. IEEE Congr. Evol. Comput.*, Brisbane, QLD, Australia, Jun. 2012, pp. 2718–2725.

[30] R. Hunt, K. Neshatian, and M. Zhang, "A genetic programming approach to hyper-heuristic feature selection," in *Proc. 9th Int. Conf. Simulat. Evol. Learn. (SEAL)*, vol. 7673. Hanoi, Vietnam, Dec. 2012, pp. 320–330.

[31] Q. Chen, M. Zhang, and B. Xue, "Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression," *IEEE Trans. Evol. Comput.*, vol. 21, no. 5, pp. 792–806, Oct. 2017.

[32] G. Dick, "Sensitivity-like analysis for feature selection in genetic programming," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Berlin, Germany, Jul. 2017, pp. 401–408.

[33] K. Neshatian, M. Zhang, and P. Andreae, "Genetic programming for feature ranking in classification problems," in *Proc. 7th Int. Conf. Simulat. Evol. Learn. (SEAL)*, vol. 5361. Melbourne, VIC, Australia, Dec. 2008, pp. 544–554.

[34] W. B. Langdon and B. F. Buxton, "Genetic programming for mining DNA chip data from cancer patients," *Genet. Program. Evol. Mach.*, vol. 5, no. 3, pp. 251–257, Sep. 2004.

[35] K. Neshatian and M. Zhang, "Using genetic programming for context-sensitive feature scoring in classification problems," *Connection Sci.*, vol. 23, no. 3, pp. 183–207, Sep. 2011.

[36] R. Loughran, A. Agapitos, A. Kattan, A. Brabazon, and M. O'Neill, "Feature selection for speaker verification using genetic programming," *Evol. Intell.*, vol. 10, nos. 1–2, pp. 1–21, 2017.

[37] S. Ahmed, M. Zhang, and L. Peng, "Enhanced feature selection for biomarker discovery in LC-MS data using GP," in *Proc. IEEE Conf. Evol. Comput.*, vol. 1. Cancún, Mexico, Jun. 2013, pp. 584–591.

[38] S. Ahmed, M. Zhang, and L. Peng, "Feature selection and classification of high dimensional mass spectrometry data: A genetic programming approach," in *Proc. EvoBIO*, vol. 7833. Vienna, Austria, Apr. 2013, pp. 43–55.

[39] B. Tran, B. Xue, and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," *Memetic Comput.*, vol. 8, no. 1, pp. 3–15, Mar. 2016.

[40] D. P. Muni, N. R. Pal, and J. Das, "Genetic programming for simultaneous feature selection and classifier design," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 106–117, Feb. 2006.

[41] A. Purohit, N. S. Chaudhari, and A. Tiwari, "Construction of classifier with feature selection based on genetic programming," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Barcelona, Spain, Jul. 2010, pp. 1–5.

[42] Q. Chen, B. Xue, B. Niu, and M. Zhang, "Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Vancouver, BC, Canada, Jul. 2016, pp. 3793–3800.

[43] E. Vladislavleva, T. Friedrich, F. Neumann, and M. Wagner, "Predicting the energy output of wind farms based on weather data: Important variables and their correlation," *Renew. Energy*, vol. 50, pp. 236–243, Feb. 2013.

[44] A. Friedlander, K. Neshatian, and M. Zhang, "Meta-learning and feature ranking using genetic programming for classification: Variable terminal weighting," in *Proc. IEEE Congr. Evol. Comput.*, New Orleans, LA, USA, Jun. 2011, pp. 941–948.

[45] N. Y. Nikolaev and H. Iba, "Regularization approach to inductive genetic programming," *IEEE Trans. Evol. Comput.*, vol. 54, no. 4, pp. 359–375, Aug. 2001.

[46] J. Ni and P. Rockett, "Tikhonov regularization as a complexity measure in multiobjective genetic programming," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 157–166, Apr. 2015.

[47] Y. Wu, J. Lu, and Y. Sun, "Genetic programming based on an adaptive regularization method," in *Proc. Int. Conf. Comput. Intell. Security*, vol. 1. Guangzhou, China, Nov. 2006, pp. 324–327.

[48] Y. S. Yeun, K. H. Lee, S. M. Han, and Y. S. Yang, "Smooth fitting with a method for determining the regularization parameter under the genetic programming algorithm," *Inf. Sci.*, vol. 133, nos. 3–4, pp. 175–194, Apr. 2001.

[49] E. J. Vladislavleva, G. F. Smits, and D. den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 333–349, Apr. 2009.

[50] Q. Chen, B. Xue, L. Shang, and M. Zhang, "Improving generalisation of genetic programming for symbolic regression with structural risk minimisation," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Denver, CO, USA, Jul. 2016, pp. 709–716.

[51] Y. Bernstein, X. Li, V. Ciesielski, and A. Song, "Multiobjective parsimony enforcement for superior generalisation performance," in *Proc. IEEE Congr. Evol. Comput.*, Portland, OR, USA, Jun. 2004, pp. 83–89.

[52] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, "Data mining with constrained-syntax genetic programming: Applications to medical data sets," in *Proc. Intell. Data Anal. Med. Pharmacol. (IDAMAP)*, 2001.

[53] M. C. J. Bot, "Improving induction of linear classification trees with genetic programming," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Las Vegas, NV, USA, Jul. 2000, pp. 403–410.

[54] M. Castelli, L. Manzoni, S. Silva, and L. Vanneschi, "A comparison of the generalization ability of different genetic programming frameworks," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Barcelona, Spain, Jul. 2010, pp. 1–8.

[55] M. J. Cavaretta and K. Chellapilla, "Data mining using genetic programming: The implications of parsimony on generalization error," in *Proc. Congr. Evol. Comput.*, vol. 2. Washington, DC, USA, Jul. 1999, pp. 1330–1337.

[56] G. Kronberger, M. Kommenda, and M. Affenzeller, "Overfitting detection and adaptive covariant parsimony pressure for symbolic regression," in *Proc. 3rd Symbolic Regression Model. Workshop GECCO*, Dublin, Ireland, Jul. 2011, pp. 631–638.

[57] J. P. Rosca, "Generality versus size in genetic programming," in *Proc. 1st Annu. Conf. Genet. Program.*, Stanford, CA, USA, Jul. 1996, pp. 381–387.

[58] W. A. Tackett, "Genetic programming for feature discovery and image discrimination," in *Proc. 5th Int. Conf. Genet. Algorithms (ICGA)*, Jul. 1993, pp. 303–311.

[59] B.-T. Zhang, "Bayesian methods for efficient genetic programming," *Genet. Program. Evol. Mach.*, vol. 1, no. 3, pp. 217–242, Jul. 2000.

[60] H. Iba, H. de Garis, and T. Sato, "Genetic programming using a minimum description length principle," in *Advances in Genetic Programming*. Cambridge, MA, USA: MIT Press, 1994, ch. 12, pp. 265–284.

[61] C. E. Borges, C. L. Alonso, and J. L. Montaña, "Model selection in genetic programming," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Portland, OR, USA, Jul. 2010, pp. 985–986.

[62] M. Castelli, L. Manzoni, S. Silva, and L. Vanneschi, "A quantitative study of learning and generalization in genetic programming," in *Proc. 14th Eur. Conf. Genet. Program. (EuroGP)*, vol. 6621. Turin, Italy, Apr. 2011, pp. 25–36.

[63] E. Vladislavleva, G. Smits, and D. den Hertog, "On the importance of data balancing for symbolic regression," *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 252–277, Apr. 2010.

[64] A. Agapitos, A. Brabazon, and M. O'Neill, "Controlling overfitting in symbolic regression based on a bias/variance error decomposition," in *Proc. PPSN*, vol. 7491. Sicily, Italy, Sep. 2012, pp. 438–447.

[65] J. Fitzgerald, R. M. A. Azad, and C. Ryan, "A bootstrapping approach to reduce over-fitting in genetic programming," in *Proc. 15th Annu. Conf. GECCO*, Amsterdam, The Netherlands, Jul. 2013, pp. 1113–1120.

[66] M. Zhang and U. Bhowan, "Program size and pixel statistics in genetic programming for object detection," in *Applications of Evolutionary Computing* (LNCS 3005), Coimbra, Portugal, Apr. 2004, pp. 379–388.

[67] A. Agapitos, M. O'Neill, A. Brabazon, and T. Theodoridis, "Maximum margin decision surfaces for increased generalisation in evolutionary decision tree learning," in *Proc. 14th Eur. Conf. Genet. Program. (EuroGP)*, vol. 6621. Turin, Italy, Apr. 2011, pp. 61–72.

[68] M. Zhang and W. Smart, "Using Gaussian distribution to construct fitness functions in genetic programming for multiclass object classification," *Pattern Recognit. Lett.*, vol. 27, no. 11, pp. 1266–1274, Aug. 2006.

[69] W. Smart and M. Zhang, "Using genetic programming for multiclass classification by simultaneously solving component binary classification problems," in *Proc. 8th Eur. Conf. Genet. Program.*, vol. 3447. Lausanne, Switzerland, Mar./Apr. 2005, pp. 227–239.

[70] U. Bhowan, M. Johnston, and M. Zhang, "Developing new fitness functions in genetic programming for classification with unbalanced data," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 2, pp. 406–421, Apr. 2012.

[71] T. Theodoridis, A. Agapitos, and H. Hu, "A Gaussian groundplan projection area model for evolving probabilistic classifiers," in *Proc. ACM 13th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Dublin, Ireland, Jul. 2011, pp. 1339–1346.

[72] J. de Jong and K. Neshatian, "Binary classification using genetic programming: Evolving discriminant functions with dynamic thresholds," in *Trends and Applications in Knowledge Discovery and Data Mining* (LNCS 7867). Heidelberg, Germany: Springer, Apr. 2013, pp. 464–474.

[73] J. Doucette and M. I. Heywood, "GP classification under imbalanced data sets: Active sub-sampling and AUC approximation," in *Proc. 11th Eur. Conf. Genet. Program. (EuroGP)*, vol. 4971. Naples, Italy, Mar. 2008, pp. 266–277.

[74] W. B. Langdon and B. F. Buxton, "Genetic programming for combining classifiers," in *Proc. Genet. Evol. Comput. Conf.*, San Francisco, CA, USA, Jul. 2001, pp. 66–73.

[75] J. Eggermont, A. E. Eiben, and J. I. van Hemert, "Adapting the fitness function in GP for data mining," in *Proc. Genet. Program. EuroGP*, vol. 1598. Gothenburg, Sweden, May 1999, pp. 193–202.

[76] J. Ni and P. Rockett, "Training genetic programming classifiers by vicinal-risk minimization," *Genet. Program. Evol. Mach.*, vol. 16, no. 1, pp. 3–25, Mar. 2015.

[77] M. Conrads, P. Nordin, and W. Banzhaf, "Speech sound discrimination with genetic programming," in *Proc. 1st Eur. Workshop Genet. Program.*, vol. 1391. Paris, France, Apr. 1998, pp. 113–129.

[78] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in *Parallel Problem Solving From Nature—PPSN III* (Lecture Notes in Computer Science), vol. 866. Heidelberg, Germany: Springer-Verlag, Oct. 1994, pp. 312–321.

[79] I. Goncalves, S. Silva, J. B. Melo, and J. M. B. Carreiras, "Random sampling technique for overfitting control in genetic programming," in *Proc. 15th Eur. Conf. Genet. Program. (EuroGP)*, vol. 7244. Málaga, Spain, Apr. 2012, pp. 217–228.

[80] Y. Liu and T. Khoshgoftaar, "Reducing overfitting in genetic programming models for software quality classification," in *Proc. 8th IEEE Symp. Int. High Assurance Syst. Eng.*, Tampa, FL, USA, Mar. 2004, pp. 56–65.

[81] I. Gonçalves and S. Silva, "Balancing learning and overfitting in genetic programming with interleaved sampling of training data," in *Proc. 16th Eur. Conf. Genet. Program. (EuroGP)*, vol. 7831. Vienna, Austria, Apr. 2013, pp. 73–84.

[82] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training genetic programming on half a million patterns: An example from anomaly detection," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 225–239, Jun. 2005.

[83] B. Dolin, F. H. Bennett, III, and E. G. Rieffel, "Co-evolving an effective fitness sample: Experiments in symbolic regression and distributed robot control," in *Proc. ACM Symp. Appl. Comput.*, Madrid, Spain, 2002, pp. 553–559.

[84] M. D. Schmidt and H. Lipson, "Coevolution of fitness predictors," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 736–749, Dec. 2008.

[85] E. V. Siegel, "Competitively evolving decision trees against fixed training cases for natural language processing," in *Advances in Genetic Programming*. Cambridge, MA, USA: MIT Press, 1994, ch. 19, pp. 409–423.

[86] M. A. Haeri, M. M. Ebadzadeh, and G. Folino, "Improving GP generalization: A variance-based layered learning approach," *Genet. Program. Evol. Mach.*, vol. 16, no. 1, pp. 27–55, Mar. 2015.

[87] J. M. Daida, T. F. Bersano-Begey, S. J. Ross, and J. F. Vesecky, "Computer-assisted design of image classification algorithms: Dynamic and static fitness evaluations in a scaffolded genetic programming environment," in *Proc. 1st Annu. Conf. Genet. Program.*, Stanford, CA, USA, Jul. 1996, pp. 279–284.

[88] T. H. Nguyen and X. H. Nguyen, "Learning in stages: A layered learning approach for genetic programming," in *Proc. IEEE Conf. Comput. Commun. Technol. Res. Innov. Vis. Future (RIVF)*, Ho Chi Minh City, Vietnam, Feb./Mar. 2012, pp. 1–4.

[89] H. Iba, "Bagging, boosting, and bloating in genetic programming," in *Proc. Genet. Evol. Comput. Conf.*, vol. 2. Orlando, FL, USA, Jul. 1999, pp. 1053–1060.

[90] G. Folino, C. Pizzuti, and G. Spezzano, "Ensemble techniques for parallel genetic programming based classifiers," in *Proc. Genet. Program. EuroGP*, vol. 2610. Essex, U.K., Apr. 2003, pp. 59–69.

[91] G. Folino, C. Pizzuti, and G. Spezzano, "GP ensembles for large-scale data classification," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 604–616, Oct. 2006.

[92] J.-H. Hong and S.-B. Cho, "The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming," *Artif. Intell. Med.*, vol. 36, no. 1, pp. 43–58, Jan. 2006.

[93] Y. Zhang and S. Bhattacharyya, "Genetic programming in classifying large-scale data: An ensemble method," *Inf. Sci.*, vol. 163, nos. 1–3, pp. 85–101, Jun. 2004.

[94] K.-H. Liu and C.-G. Xu, "A genetic programming-based approach to the classification of multiclass microarray datasets," *Bioinformatics*, vol. 25, no. 3, pp. 331–337, Feb. 2009.

[95] M. Brameier and W. Banzhaf, "Evolving teams of predictors with linear genetic programming," *Genet. Program. Evol. Mach.*, vol. 2, no. 4, pp. 381–407, Dec. 2001.

[96] A. Agapitos, M. O'Neill, and A. Brabazon, "Ensemble Bayesian model averaging in genetic programming," in *Proc. IEEE Congr. Evol. Comput.*, Beijing, China, Jul. 2014, pp. 2451–2458.

[97] C. De Stefano, G. Folino, F. Fontanella, and A. S. di Freca, "Using Bayesian networks for selecting classifiers in GP ensembles," *Inf. Sci.*, vol. 258, pp. 200–216, Feb. 2014.

[98] W. B. Langdon, S. J. Barrett, and B. F. Buxton, "Combining decision trees and neural networks for drug discovery," in *Proc. 5th Eur. Conf. Genet. Program. (EuroGP)*, vol. 2278. Kinsale, Ireland, Apr. 2002, pp. 60–70.

[99] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Reusing genetic programming for ensemble selection in classification of unbalanced data," *IEEE Trans. Evol. Comput.*, vol. 18, no. 6, pp. 893–908, Dec. 2014.

[100] G. Folino, C. Pizzuti, and G. Spezzano, "Boosting technique for combining cellular GP classifiers," in *Proc. 7th Eur. Conf. Genet. Program. (EuroGP)*, vol. 3003. Coimbra, Portugal, Apr. 2004, pp. 47–56.

[101] G. Paris, D. Robilliard, and C. Fonlupt, "Applying boosting techniques to genetic programming," in *Proc. 5th Int. Conf. Evol. Artif. (EA)*, vol. 2310. Creusot, France, Oct. 2001, pp. 267–278.

[102] W. Banzhaf, F. D. Francone, and P. Nordin, "The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets," in *Parallel Problem Solving From Nature—PPSN IV* (LNCS 1141). Berlin, Germany: Springer-Verlag, Sep. 1996, pp. 300–309.

[103] G. Paris, D. Robilliard, and C. Fonlupt, "Exploring overfitting in genetic programming," in *Proc. 6th Int. Conf. Evol. Artif.*, vol. 2936. Marseille, France, Oct. 2003, pp. 267–277.

[104] U. N. Quang, T. H. Nguyen, X. H. Nguyen, and M. O'Neill, "Improving the generalisation ability of genetic programming with semantic similarity based crossover," in *Proc. EuroGP*, vol. 6021. Istanbul, Turkey, Apr. 2010, pp. 184–195.

[105] Q. Chen, B. Xue, Y. Mei, and M. Zhang, "Geometric semantic crossover with an angle-aware mating scheme in genetic programming for symbolic regression," in *Proc. 20th Eur. Conf. Genet. Program. (EuroGP)*, vol. 10196. Amsterdam, The Netherlands, Apr. 2017, pp. 229–245.

[106] M. Zhang and W. Smart, "Genetic programming with gradient descent search for multiclass object classification," in *Proc. 7th Eur. Conf. Genet. Program. (EuroGP)*, vol. 3003. Coimbra, Portugal, Apr. 2004, pp. 399–408.

[107] Q. Chen, B. Xue, and M. Zhang, "Generalisation and domain adaptation in GP with gradient descent for symbolic regression," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Sendai, Japan, May 2015, pp. 1137–1144.

[108] R. M. A. Azad and C. Ryan, "Variance based selection to improve test set performance in genetic programming," in *Proc. ACM 13th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Dublin, Ireland, Jul. 2011, pp. 1315–1322.

[109] L. Vanneschi and S. Gustafson, "Using crossover based similarity measure to improve genetic programming generalization ability," in *Proc. ACM 11th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Montreal, QC, Canada, Jul. 2009, pp. 1139–1146.

[110] K. Burnhum and D. Anderson, *Model Selection and Multimodel Inference*, 2nd ed. New York, NY, USA: Springer, 2010.

[111] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, nos. 1–2, pp. 273–324, Dec. 1997.

[112] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003.

[113] M. Dash and H. Liu, "Feature selection for classification," *Intell. Data Anal.*, vol. 1, nos. 1–4, pp. 131–156, 1997.

[114] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 606–626, Aug. 2016.

[115] B. Xue, M. Zhang, and W. N. Browne, "A comprehensive comparison on evolutionary feature selection approaches to classification," *Int. J. Comput. Intell. Appl.*, vol. 14, no. 2, 2015.

[116] R. Hunt, K. Neshatian, and M. Zhang, "Scalability analysis of genetic programming classifiers," in *Proc. IEEE Congr. Evol. Comput.*, Brisbane, QLD, Australia, Jun. 2012, pp. 509–516.

[117] M. Zhang and W. Smart, "Multiclass object classification using genetic programming," in *Applications of Evolutionary Computing* (LNCS 3005). Heidelberg, Germany: Springer-Verlag, Apr. 2004, pp. 369–378.

[118] S. Silva and L. Vanneschi, "Operator equalisation, bloat and overfitting: A study on human oral bioavailability prediction," in *Proc. ACM 11th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Montreal, QC, Canada, Jul. 2009, pp. 1115–1122.

[119] S. Mahler, D. Robilliard, and C. Fonlupt, "Tarpeian bloat control and generalization accuracy," in *Proc. 8th Eur. Conf. Genet. Program.*, vol. 3447. Lausanne, Switzerland, Mar./Apr. 2005, pp. 203–214.

[120] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *Proc. ACM 12th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Portland, OR, USA, Jul. 2010, pp. 877–884.

[121] G. Dick, "Bloat and generalisation in symbolic regression," in *Proc. 10th Int. Conf. Simulat. Evol. Learn. (SEAL)*, vol. 8886. Dunedin, New Zealand, 2014, pp. 491–502.

[122] J. Fitzgerald and C. Ryan, "On size, complexity and generalisation error in GP," in *Proc. ACM Conf. Genet. Evol. Comput. (GECCO)*, Vancouver, BC, Canada, Jul. 2014, pp. 903–910.

[123] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Trans. Evol. Comput.*, vol. 17, no. 3, pp. 368–386, Jun. 2013.

[124] K. Imamura, T. Soule, R. B. Heckendorn, and J. A. Foster, "Behavioral diversity and a probabilistically optimal GP ensemble," *Genet. Program. Evol. Mach.*, vol. 4, no. 3, pp. 235–253, Sep. 2003.

[125] R. Thomason and T. Soule, "Novel ways of improving cooperation and performance in ensemble classifiers," in *Proc. ACM 9th Annu. Conf. Genet. Evol. Comput. (GECCO)*, vol. 2. London, U.K., Jul. 2007, pp. 1708–1715.

[126] G. Folino, C. Pizzuti, and G. Spezzano, "Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification," *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 458–468, Aug. 2008.

[127] I. Gonçalves, S. Silva, and C. M. Fonseca, "On the generalization ability of geometric semantic genetic programming," in *Proc. 18th Eur. Conf. Genet. Program.*, vol. 9025. Copenhagen, Denmark, Apr. 2015, pp. 41–52.

[128] M. Keijzer, "Improving symbolic regression with interval arithmetic and linear scaling," in *Proc. Genet. Program. EuroGP*, vol. 2610. Essex, U.K., Apr. 2003, pp. 70–82.

[129] D. R. White *et al.*, "Better GP benchmarks: Community survey results and proposals," *Genet. Program. Evol. Mach.*, vol. 14, no. 1, pp. 3–29, Mar. 2013.

[130] K. M. Sullivan and S. Luke, "Evolving kernels for support vector machine classification," in *Proc. 9th Annu. Conf. Genet. Evol. Comput. (GECCO)*, vol. 2. London, U.K., Jul. 2007, pp. 1702–1707.

[131] C. Gagné, M. Schoenauer, M. Sebag, and M. Tomassini, "Genetic programming for kernel-based learning with co-evolving subsets selection," in *Parallel Problem Solving From Nature—PPSN IX* (LNCS 4193). Heidelberg, Germany: Springer-Verlag, Sep. 2006, pp. 1008–1017.

[132] A. Agapitos, J. McDermott, M. O'Neill, A. Kattan, and A. Brabazon, "Higher order functions for kernel regression," in *Proc. 17th Eur. Conf. Genet. Program.*, vol. 8599. Granada, Spain, Apr. 2014, pp. 1–12.

[133] A. J. Turner and J. F. Miller, "Recurrent cartesian genetic programming of artificial neural networks," *Genet. Program. Evol. Mach.*, vol. 18, no. 2, pp. 185–212, Jun. 2017.

[134] K. Neshatian, M. Zhang, and P. Andreae, "A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 16, no. 5, pp. 645–661, Oct. 2012.

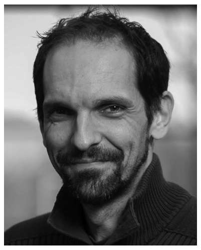
[135] K. Krawiec, "Genetic programming-based construction of features for machine learning and knowledge discovery tasks," *Genet. Program. Evol. Mach.*, vol. 3, no. 4, pp. 329–343, Dec. 2002.

Alexandros Agapitos received the B.Sc. degree in software engineering from the University of South Wales, Pontypridd, U.K., in 2002, and the M.Sc. degree in e-commerce technology and the Ph.D. degree in computer science (with a focus on genetic programming and its application to the evolution of recursive and memory-enabled algorithms) from the University of Essex, Colchester, U.K., in 2004 and 2009, respectively.

He moved to University College Dublin, Dublin, Ireland, in 2010, where he was a Post-Doctoral Research Fellow with the Complex and Adaptive Systems Laboratory until 2016. He is currently a Principal Researcher with the Ireland Research and Innovation Center, Huawei Technologies, Dublin, where he applies artificial intelligence to solve problems in telecommunication networks. His current research interests include evolutionary computation and artificial neural networks applied to reinforcement learning.

Roisin Loughran received the B.E. degree in electronic engineering from University College Dublin (UCD), Dublin, Ireland, in 2001, the M.Phil. degree in music and media technologies from Trinity College Dublin, Dublin, in 2004, and the Ph.D. degree in instrument recognition using evolutionary computation from the University of Limerick, Limerick, Ireland, in 2010.

She is currently a Senior Researcher of the Applications in Evolutionary Design Project (App'Ed) under SFI with UCD, where she is part of the Natural Computing Research and Applications Group, specializing in applying natural computing methods to computational creativity.



Miguel Nicolau received the B.Sc. degree from UCL, Ottignies-Louvain-la-Neuve, Belgium, and the B.Sc., M.Sc., and Ph.D. degrees from the University of Limerick, Limerick, Ireland.

He was an Expert Engineer with INRIA Institute, Paris, France. After moving back to Ireland, he was a Research Fellow and an Assistant Professor with University College Dublin, Dublin, Ireland. His research and teaching experience spans over 18 years, and includes positions at the University of Limerick, Fudan University, Shanghai, China, and University College Dublin.

Simon Lucas received the Ph.D. degree in artificial intelligence from the University of Southampton, Southampton, U.K., with a focus on neural networks based on grammars, an early form of deep neural network.

He is a Professor of Artificial Intelligence and the Head of the School of Electronic Engineering and Computer Science, Queen Mary University of London, London, U.K., where he also heads the Game AI Research Group. His currrent research interests include novel methods for better game AI, using AI to design better games, and artificial general intelligence.

Dr. Lucas is the Founding Editor-in-Chief of the IEEE TRANSACTIONS ON GAMES and co-founded the IEEE Conference on Computational Intelligence and Games.

Michael O'Neill received the Ph.D. degree in computer science from the University of Limerick, Limerick, Ireland.

He is the ICON Chair of Business Analytics with the School of Business, University College Dublin (UCD), Dublin, Ireland, where he is the Founding Director of the UCD Natural Computing Research and Applications Group, the Associate Dean and the Vice-Principal (Research) of the UCD Michael Smurfit Graduate Business School from 2015 to 2018, and the Director of the UCD's Interdisciplinary Research Institute from 2012 to 2015. One of the inventors of grammatical evolution, he is the lead author of the seminal book on this subject, and has published over 300 peer-reviewed publications including 4 monographs. His current research interests include automatic programming and genetic programming, with applications in areas such as communications networks, business analytics, sports analytics, and design and creativity.

Anthony Brabazon received the B.Com. degree from University College Dublin, Dublin, Ireland, in 1988, the first M.S. degree in statistics and the second M.S. degree in operations research from Stanford University, Stanford, CA, USA, in 1994, the M.B.A. degree from Heriot-Watt University, Edinburgh, U.K., in 1998, and the D.B.A. degree from Kingston University, London, U.K., in 2005.

He is currently the Dean of the School of Business, University College Dublin, Dublin, Ireland, where he is the Co-Founder and the Co-Director of the Natural Computing Research and Applications Group. He has published over 200 peer-reviewed studies and authored/edited 16 books. His current research interests include development of natural computing algorithms and their application to real-world problems.