Model-Driven Regularization Approach to Straight Line Program Genetic Programming

José L. Montaña[a,*], César L. Alonso[b], Cruz E. Borges[c], Cristina Tîrnăucă[a]

[a] *Universidad de Cantabria, Av. de los Castros s/n, 39005 Santander, Spain*
[b] *Universidad de Oviedo, Calle San Francisco 1, 33003 Oviedo, Spain*
[c] *Universidad de Deusto, Av. de las Universidades 24, 48007 Bilbao, Spain*

Abstract

This paper presents a regularization method for program complexity control of linear genetic programming tuned for transcendental elementary functions. Our goal is to improve the performance of evolutionary methods when solving symbolic regression tasks involving Pfaffian functions such as polynomials, analytic algebraic and transcendental operations like sigmoid, inverse trigonometric and radial basis functions. We propose the use of straight line programs as the underlying structure for representing symbolic expressions. Our main result is a sharp upper bound for the Vapnik Chervonenkis dimension of families of straight line programs containing transcendental elementary functions. This bound leads to a penalization criterion for the mean square error based fitness function often used in genetic programming for solving inductive learning problems. Our experiments show that the new fitness function gives very good results when compared with classical statistical regularization methods (such as Akaike and Bayesian Information Criteria) in almost all studied situations, including some benchmark real-world regression problems.

Keywords: genetic programming, straight line program, Pfaffian operator, symbolic regression

*Corresponding author. Tel.: +34942201529
Email addresses: joseluis.montana@unican.es (José L. Montaña), calonso@uniovi.es (César L. Alonso), cruz.borges@deusto.es (Cruz E. Borges), cristina.tirnauca@unican.es (Cristina Tîrnăucă)

1. Introduction

Inductive inference (Tenebaum et al. (2006); Angluin and Smith (1983); Gori et al. (1998); Shaoning and Kasabov (2004)) is one of the main fields in Machine Learning, and it can be defined as the process of hypothesizing a general rule from examples. The methods used in inductive inference span a variety of very different Machine Learning tools including neural networks, regression and decision trees, support vector machines, Bayesian networks, probabilistic finite state machines and many other statistical techniques. Given a sample set of an unknown process, the problem of finding a model capable to predict correct values for new examples has applications in a diversity of fields such as economics, electronic design, game playing, physical processes, etc. When dealing with real-world problems, the sample data are generally obtained through measures that are often corrupted by noise. In addition, the distribution according to which the examples are generated is usually unknown. This situation is very common when trying to solve inductive learning problems in the context of symbolic regression and must be considered and modeled in its whole complexity to produce reasonable models. Symbolic regression can be understood as the problem of finding a mathematical formula that fits a set of data. For a long time symbolic regression was a human task, but the advent of computers facilitated the exploration of the huge search space in which the regression process usually takes place. Genetic Programming (GP) can be seen as a symbolic regression strategy by means of evolutionary algorithms. This idea was proposed by J. Koza to whom both the term and the concept are due (see Koza (1992)).

In the last years, GP has been applied to a wide range of situations to solve both unsupervised and supervised learning problems, and it has become a powerful tool in the Knowledge Discovery and Data Mining domain (e.g., Freitas (2002)), including the emergent field of Big Data analysis (see Castelli et al. (2015)). Main subjects in unsupervised learning like clustering have been approached using GP (see Bezdek et al. (1994); Jie et al. (2003); Falco et al. (2004); Folino et al. (2008)). Supervised classification by evolving selection

rules is another avenue in which GP obtains a remarkable success as shown, for example, in Carreño et al. (2007); Cano et al. (2007); Chien et al. (2003); Freitas (1997); Hennessy et al. (2005) and Kuo et al. (2007). Singular applications to medicine and biology problems (Bojarczuk et al. (2000, 2004); Castelli et al. (2014); Aslam et al. (2013)), feature extraction methods (Krawiec (2002); Smith and Bull (2005)), database clustering and rule extraction (Wedashwara et al. (2015)), generation of hybrid multi-level predictors for function approximation and regression analysis (Tsakonas and Gabrys (2012)) are other examples in which GP is applied. Specific applications to inductive learning problems solved via GP can be found in relatively old papers (Okley (1994); Poli and Cagnoni (1997)).

The general procedure of GP consists in the evolution of a population of computer programs, each of them computing a certain function, with the aim of finding one that best describes the target function. These computer programs are build out from a set of functions and a set of terminals consisting of variables and constants. In the evolutionary process, the fitness function evaluates the goodness of each member of the population by measuring the empirical error over the sample set. In the presence of noise and to prevent other causes of overfitting (like, for example, the use of a very complex model), this fitness function must be regularized with some term that usually depends on the complexity of the model. Regularization is then a central problem related to generalization. By generalization we mean that the empirical error must converge to the expected true error when the number of examples increases. This notion of generalization defined here roughly agrees with the informal use of the term in GP (good performance over unseen examples) but captures the nature of the problem in the learning theory setting. The problem of generalization is focused for example in Tackett and Carmi (1994) and also in Cavaretta and Chellapilla (1999). In Keijzer and Babovic (2000), ensembles methods that can improve the generalization error in GP are discussed. The specific problem of regularization is extensively treated for polynomials in Nikolaev and Iba (2001); Nikolaev et al. (2002). Recent work regarding regularization used in GP can be found in Ni

and Rockett (2015b), where a *vicinal risk* regularization technique is explored. A detailed description of other GP-regularization strategies can be found in Ni and Rockett (2015a), where Tikhonov regularization, in conjunction with node count as a general complexity measure in multiobjective GP, is proposed.

Most work describing GP strategies for solving symbolic regression problems employs trees as data structure for representing programs. There are other methodologies having an instruction-based approach, like that of matrix representation given in Li et al. (2008) or the more classical Cartesian Genetic Programming (CGP) of Miller and Thomson (2000). In this paper, we propose the use of straight line programs (SLPs) as data structure to evolve in GP. The SLP structure has a good performance when solving symbolic regression problem instances as shown in Alonso et al. (2008). The main difference between SLP GP and CGP consists in the implementation of the crossover operator, which in the case of SLPs is designed to interchange subgraphs, as described in Section 3. While GP-trees are expression trees of formulas, the SLPs correspond to expression dags (direct acyclic graphs) in which precomputed results can be reused. This makes the SLP data structure more compact and, usually, smaller than a tree, and consequently, easier to evaluate. Moreover, there is a canonical transformation mapping GP-trees onto SLPs (see Alonso et al. (2009) for a detailed explanation of the relationships between SLPs and GP-trees). This transformation preserves the size and other complexity measures. For this reason, conclusions related to SLP performance can be extended to GP-tree performance at least in the context of generalization and regularization.

This paper is primarily concerned with statistical regularization methods which can be applied to a wide family of models. In particular we study a complexity measure for families of data structures representing programs named Vapnik Chervonenkis dimension (VCD). This measure, which belongs to the field of Information Theory, usually appears in conjunction with the Probably Approximately Correct approach to supervised learning, commonly referred to in the literature as the PAC model (see Vapnik (1998) and Vapnik and Chervonenkis (1974)). We consider families of SLPs constructed from a set of Pfaffian

functions (solutions of triangular systems of first order partial differential equations with polynomial coefficients - the formal definition is given in Section 6). As important examples, polynomials, exponential functions, trigonometric functions on some particular intervals and, in general, analytic algebraic functions are all Pfaffian. The main result of this paper is summarized in Theorem 1, where an upper bound for the VCD of a family of SLPs using Pfaffian functions is found. This upper bound is polynomial on some parameters, and in particular, on the non-scalar length of the SLPs. This implies that the length of the SLPs can be arbitrary if the excess is caused by addition and/or subtraction operations. Based on this result, we propose a method for selecting models in SLP GP for the case in which the admitted operations are all Pfaffian. In general, analyticity of the operators is a very desirable property in numerical computations, leading to more robust computer programs and providing a higher classification capacity. There are even well-founded proposals to replace the traditional protected division in GP by analytic operators, providing much better experimental results as shown in Ni et al. (2013). This is another strong motivation for using Pfaffian operations.

The paper is organized as follows. Section 2 contains some basic concepts concerning statistical regression, the model selection criteria used in this paper, along with the general definition of the VCD of a family of sets. In Section 3 we describe the SLP data structure, and we introduce, via a technical lemma, the concept of an universal SLP. Section 4 includes the main traits of the SLP GP paradigm. In Section 5 we provide basic definitions for the Pfaffian operators, and we present a summary of previous technical results that are used in the proof of our main theorem, which can be found in Section 6. There, we give an upper bound for the VCD of families of SLPs that use as operators only Pfaffian functions. Section 7 shows the results of an extensive experimentation phase on a variety of symbolic regression problems. We also provide a comparative analysis between our method and two other well-known statistical regularization strategies, in which the complexity of the models is estimated by the number of free parameters. Finally, Section 8 contains some conclusions and future work.

2. Supervised Learning and Regression

Genetic Programming can be seen as a direct evolution method of computer programs for inductive learning. Inductive GP can be considered as a specialization of GP, in that it uses the framework of the last one in order to solve inductive learning problems. These problems are, in general, searching problems, where the aim is to find the best model from a finite set of observed data. Note that the best model might not be the one that perfectly fits the data, since this could lead to overfitting and, consequently, to a poor performance on unseen instances. The idea is to look for a model that fits well the data set, being at the same time as simple as possible. This immediately raises the question of how to measure the complexity of a model. There are many ways to do this. For example, we could prefer models with a small number of free parameters, which corresponds to simple mathematical formulas. Or, if the model is represented by a program, we could consider the length of the program as a complexity measure. Usually, for tree structures, a measure of the complexity is the height or the width of the tree. There is no universal way of measuring the complexity of the model and the choice of a specific measure inherently depends on the problem at hand. We adopt the approach to view the symbolic regression problem under the general setting for predictive learning provided by the PAC setting of Vapnik (see Vapnik (1995), Cherkassky and Mulier (1998), Hastie et al. (2001)). This point of view has the advantage of providing a toolbox of theory and methods both for problem statement and complexity measure of the predictive models. Under this framework, the goal is to estimate an unknown real-valued function f that satisfies the relation

$$y = f(\vec{x}) + \varepsilon, \tag{1}$$

where ε is independent identically distributed (i.i.d.) zero-mean random error (noise), $\vec{x}$ is a vector in $X \subseteq \mathbb{R}^n$ and y is a scalar in the output set $Y \subseteq \mathbb{R}$. The estimation is based on a sample of m pairs $\vec{z} = (\vec{x}_i, y_i)_{1 \leq i \leq m}$ (also called training data). These examples are drawn according to an unknown probability

measure ρ on the product space $Z = X \times Y$, and they are generated according to an i.i.d. process. As usual, the probability measure ρ factorizes through its marginal distribution in X, $\rho(\vec{x})$, and its conditional distribution in Y, $\rho(y|\vec{x})$, that is:

$$\rho(\vec{x}, y) = \rho(\vec{x})\rho(y|\vec{x})$$

Thus, we would like to find the function $f : X \to Y$ that predicts the value $y \in Y$ for any given $\vec{x} \in X$ with a low probability of error. The best estimation of the function is the mean of the output conditional probability:

$$f(\vec{x}) = \int y \; d\rho(y|\vec{x})$$

A learning method selects the best model $f \in \mathcal{H}$, where $\mathcal{H}$ is some class of functions. In general, the error $\varepsilon(f)$ of the estimator f is written as

$$\varepsilon(f) = \int Q(\vec{x}, f, y) \; d\rho, \tag{2}$$

where Q measures some notion of loss between $f(\vec{x})$ and the target value y. For regression tasks, one usually takes $Q(\vec{x}, f, y) = (f(\vec{x}) - y)^2$.

For a class of functions $\mathcal{H}$ of finite complexity (for instance trees with bounded size or height), the model can be chosen to minimize the empirical error, also known as empirical risk:

$$\varepsilon_m(f) = \frac{1}{m} \sum_{i=1}^{m} Q(\vec{x}_i, f, y_i) \tag{3}$$

Obviously, this method will have a good performance when the optimal model belongs to the class of functions considered. Unfortunately, in most practical applications, one cannot make this assumption, and a larger class must be considered. In this case, the problem of symbolic regression estimation requires optimal selection of model complexity in addition to model estimation via minimization of the empirical risk.

Analytical model selection criteria estimate the true error ($\varepsilon(f)$ in Eq. (2)) as a function of the empirical error ($\varepsilon_m(f)$ in Eq. (3)), corrected by a penalty term related to some measure of the model complexity:

$$\varepsilon(f) = \varepsilon_m(f) \odot pen(c, m), \quad \odot \in \{+, *\}$$

where c is the model's complexity and m is the size of the sample set. In the above equation, there exists a degree of freedom, namely the selection of the measure c for the model's complexity. This measure always depends on the considered class $\mathcal{H}$, and more precisely, on the representation structure for the models in $\mathcal{H}$. For example, if the functions are described by multivariate polynomials, c is usually the number of monomials or a function involving the degree and the number of variables of the polynomial. If functions in $\mathcal{H}$ are represented by programs or trees, some typical complexity measures are the length of the programs or the size of the trees.

As main representative examples of analytical model selection criteria we use two well-known statistical methods with different penalization terms, the Akaike Information Criterion (Akaike (1970)) and the Bayesian Information Criterion (Bernardo and Smith (1994)):

- Akaike Information Criterion (AIC):

$$\varepsilon(f) = \varepsilon_m(f) + \frac{2n}{m}\sigma^2 \tag{4}$$

- Bayesian Information Criterion (BIC):

$$\varepsilon(f) = \varepsilon_m(f) + \frac{n \ln m}{m}\sigma^2 \tag{5}$$

where n is the number of free parameters (of a linear estimator) and σ is an estimate of the noise variance in Eq. (1) (see Cherkassky and Yunkian (2003) for details).

$$\sigma^2 = \frac{m}{m-n}\frac{1}{m}\sum_{1 \le i \le m} (f(\vec{x}_i) - y_i)^2 \tag{6}$$

Then one can use Eq. (6) in conjunction with AIC or BIC for each (fixed) model complexity.

The third model selection criteria used in this paper is based on the *structural risk minimization* (SRM), which provides a powerful framework for model complexity control, without having to estimate noise variance. Under this approach, a set of possible models form a nested structure $C_1 \subset C_2 \subset \cdots \subset C_L \subset \cdots$, in which each C_L contains only models of bounded complexity depending on L. Model selection in this case amounts to choosing an optimal element of a structure using Vapnik-Chervonenkis (VC) generalization bounds. For regression problems, we use the following VC bound (see Cherkassky et al. (1999) for detailed derivation of Eq. (7) from the general bounds developed in VC-theory):

$$\varepsilon(f) \leq \varepsilon_m(f) \left(1 - \sqrt{\frac{h}{m}\left(1 - \ln\frac{h}{m}\right) + \frac{\ln m}{2m}}\right)^{-1} \tag{7}$$

where h is the VCD of the hypothesis class $\mathcal{H}$ that contains the model f, defined below for a family $\mathcal{F}$ of arbitrary subsets of a given set.

Definition 1. *Let $\mathcal{F}$ be a class of subsets of a set X. We say that $\mathcal{F}$ shatters a set $A \subset X$ if for every subset $E \subset A$ there exists $S \in \mathcal{F}$ such that $E = S \cap A$. Then VCD($\mathcal{F}$) is the cardinality of the largest set that is shattered by $\mathcal{F}$.*

In this paper, we use the VCD to measure the "classification capacity" of a class $\mathcal{H}$ of functions, each of these functions being computed by a certain SLP (see Section 5 for more details). Note that in our case, the search space of models does indeed form a nested structure $C_1 \subset C_2 \subset \cdots \subset C_L \subset \cdots$, each C_L being the class of SLPs that have at most L non-scalar instructions. Therefore, the algorithm will finally choose the model that minimizes the right side of Eq. (7).

3. Straight Line Programs

Straight line programs have a large history in the field of Computational Algebra. A particular class of SLPs, known in the literature as arithmetic cir-

cuits, constitutes the underlying computation model in Algebraic Complexity Theory (Burguisser et al. (1997)). Arithmetic circuits with the standard arithmetic operations $\{+, -, *, /\}$ are the natural model of computation for studying the computational complexity of algorithms solving problems with an algebraic flavor. They have been used in linear algebra problems (Berkowitz (1984)), in quantifier elimination (Heintz et al. (1990)) and in algebraic geometry (Giusti and Heinz (1993) and Giusti et al. (1998)). Also, the SLPs constitute a promising alternative to the trees in the field of Genetic Programming (see Alonso et al. (2008)). The formal definition of the SLP structure is the following:

Definition 2. *Let $F = \{f_1, \ldots, f_q\}$ be a set of functions, where f_i has arity a_i, for $1 \leq i \leq q$, and let $T = \{t_1, \ldots, t_m\}$ be a set of terminals. A straight line program over F and T is a finite sequence of computational instructions $\Gamma = \{I_1, \ldots, I_l\}$ where*

$$I_k \equiv u_k := f_{j_k}(\alpha_1, \ldots, \alpha_{a_{j_k}});$$

with $f_{j_k} \in F$, $\alpha_i \in T$ for all i if $k = 1$ and $\alpha_i \in T \cup \{u_1, \ldots, u_{k-1}\}$ for $1 < k \leq l$.

Terminal set T is of the form $T = V \cup C$, where $V = \{x_1, \ldots, x_n\}$ is a finite set of variables and $C = \{c_1, \ldots, c_p\}$ is a finite set of constants. The number of instructions l is the length of Γ.

Note that if we consider the SLP Γ as the code of a program, then a new variable u_i is introduced at each instruction I_i. We can therefore abuse the notation and denote by $\Gamma = \{u_1, \ldots, u_l\}$ an SLP. Each of the non-terminal variables u_i can be considered as an expression over the set of terminals T constructed by a sequence of recursive compositions from the set of functions F. The set of all SLPs over F and T is denoted by $SLP(F, T)$.

The output set of an SLP $\Gamma = \{u_1, \ldots, u_l\}$ is any set of non-terminal variables of Γ, that is, $O(\Gamma) = \{u_{i_1}, \ldots, u_{i_t}\}$, $i_1 < \cdots < i_t$. Provided that $V = \{x_1, \ldots, x_n\} \subset T$ is the set of terminal variables, the function computed by Γ, denoted by $\Phi_\Gamma : \mathbb{R}^n \to \mathbb{R}^t$, is defined recursively in the natural way

and satisfies $\Phi_\Gamma(x_1, \ldots, x_n) = (b_1, \ldots, b_t)$, where b_j stands for the value of the expression over V of the non-terminal variable u_{i_j}.

We can assume that the last assignment u_l belongs to $O(\Gamma)$ (i.e., $u_l = u_{i_t}$), because otherwise the assignments $u_{i_t+1}, \ldots, u_l$ could be removed from Γ, yielding a new SLP Γ' that satisfies $\Phi_\Gamma = \Phi_{\Gamma'}$.

Example 1. *Let F be the set given by the three binary standard arithmetic operations, $F = \{+, -, *\}$ and let $T = \{c_1, c_2, x_1, x_2\}$ be the set of terminals (we have $V = \{x_1, x_2\}$ and $C = \{c_1, c_2\}$ with $c_i := i$). Any SLP over F and T is a finite sequence of instructions where each instruction represents a polynomial in two variables with integer coefficients. If we consider the following SLP Γ of length 5 with output set $O(\Gamma) = \{u_5\}$:*

$$\Gamma \equiv \begin{cases} u_1 := x_1 + c_1 \\ u_2 := u_1 * u_1 \\ u_3 := x_2 + x_2 \\ u_4 := u_2 * u_3 \\ u_5 := u_4 - u_3 \end{cases} \tag{8}$$

then the function computed by Γ is the polynomial

$$\Phi_\Gamma = 2x_2(x_1 + 1)^2 - 2x_2$$

Every SLP $\Gamma = \{u_1, \ldots, u_l\}$ over F and T can be represented by a directed acyclic graph $G_\Gamma = (V, E)$. The set V of vertices is defined by $V = T' \cup \{u_1, \ldots, u_l\}$, where T' is a subset of T containing only those terminals involved in the computation. The set E of edges is constructed as follows: for every instruction $u_k := f_{j_k}(\alpha_1, \ldots, \alpha_{a_{j_k}})$ $(1 \leq k \leq l)$, we draw a_{j_k} edges (u_k, α_i), one for each $i \in \{1, \ldots, a_{j_k}\}$. The graph corresponding to the SLP described in Eq. (8) is represented in Fig. 1.

A good survey of the properties of the SLP structure and its relation with GP-trees can be found in Alonso et al. (2009). In what follows, we show how we can introduce parameters to represent families of SLPs of controlled complexity.

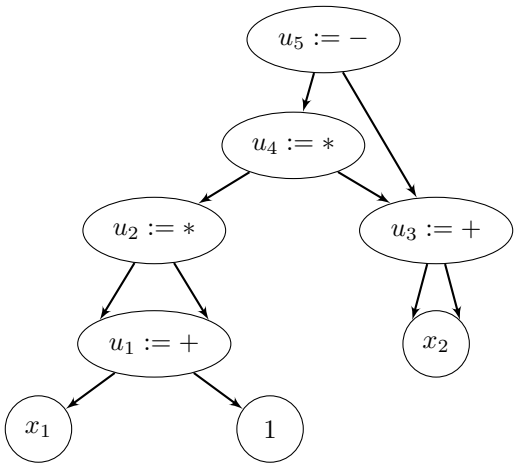

Figure 1: Directed graph representing the SLP in Eq. (8)

Lemma 1. *Let* $T = \{t_1, \ldots, t_m\}$ *be a set of terminals and* $F = \{+, -, *, /, sign\} \cup \{f_1, \ldots, f_q\}$ *be a set of functions of arity at most* A*. Let* $SLP_L(F, T)$ *be the collection of SLPs over the set* F *of functions and the set* T *of terminal symbols that use at most* L *non-scalar instructions (the number of scalar instructions is arbitrary) and whose output set consists of just one variable*[1]*. Then, one can build an universal* pseudo[2]*-SLP that parameterizes the elements of the family* $SLP_L(F, T)$ *using a vector of* k *elements, where:*

$$k = L\left(4 + q + A\left(m + \frac{L-1}{2}\right)\right) + m \tag{9}$$

Proof. We introduce three families of parameters $\vec{\alpha}, \vec{\beta}$ and $\vec{\gamma}$, which take values in $\{0,1\}^{3L}$, $\mathbb{Z}^{mAL+AL(L-1)/2+m+L}$ and $\{0,1\}^{qL}$, respectively, as follows.

For all $i \in \{1, \ldots, L\}$ define:

[1]This requirement can actually be lifted, but its absence unnecessarily complicates the proof

[2]As we shall shortly see, what we obtain is a slight modification of a standard SLP

$$U_i(\vec{\alpha},\vec{\beta},\vec{\gamma}) = (1-\alpha_1^i)\sum_{k=1}^{q}(\prod_{j=1}^{k-1}(1-\gamma_j^i))\gamma_k^i f_k(P_1^i,\ldots,P_A^i)+$$

$$+\alpha_1^i(\alpha_2^i P_1^i * P_2^i + (1-\alpha_2^i)(\alpha_3^i\frac{P_1^i}{P_2^i} + (1-\alpha_3^i)sign(P_1^i))),$$

where $P_k^i = \sum_{j=1}^{m}\beta_j{}^{(i,k)}t_j + \sum_{j=1}^{i-1}\beta_{m+j}^{(i,k)}u_j, \forall k \in \overline{1,A}$.

Also, $U_{L+1}(\vec{\alpha},\vec{\beta},\vec{\gamma}) = \sum_{j=1}^{m}\beta_j t_j + \sum_{j=1}^{L}\beta_{m+j}u_j$.

Note that for any values of $(\alpha_1^i,\alpha_2^i,\alpha_3^i)$ in $\{0,1\}^3$ and $(\gamma_1^i,\ldots,\gamma_q^i)$ in $\{0,1\}^q$, $U_i(\vec{\alpha},\vec{\beta},\vec{\gamma})$ is a non-scalar function over the set of terminals and a set $u_1,\ldots,u_{i-1}$ of non-terminal variables.

Now let Γ be an arbitrary SLP in $SLP_L(F,T)$. We can assume, without loss of generality, that Γ has exactly L non-scalar instructions. Any scalar assignment can be written as a linear combination (with integer coefficients) of terminal symbols and previously defined non-terminal variables whose values are assigned by non-scalar instructions. So, as a first step, we construct an equivalent pseudo-SLP Γ' in which we keep as non-terminal variables only those associated to non-scalar instructions, and replace the others with their equivalent rewriting (the renaming of the variables should be done carefully). For example,

$$\Gamma \equiv \begin{cases} u_1 := t_1 + t_2 \\ u_2 := u_1 * u_1 \\ u_3 := t_1 + u_2 \\ u_4 := u_3 - t_2 \\ u_5 := sign(u_4) \end{cases}$$

becomes

$$\Gamma' \equiv \begin{cases} u_1 := (t_1 + t_2) * (t_1 + t_2) \\ u_2 := sign(t_1 - t_2 + u_1) \end{cases}$$

If the last instruction of the original SLP is a scalar one, one should also maintain this last variable. In our previous example, if Γ had a sixth instructions $u_6 := u_1 + u_5$, then Γ' would be:

$$\Gamma' \equiv \begin{cases} u_1 := (t_1 + t_2) * (t_1 + t_2) \\ u_2 := sign(t_1 - t_2 + u_1) \\ u_3 := t_1 + t_2 + u_2 \end{cases}$$

The new pseudo-SLP will therefore have either $L + 1$ or L instructions, depending on whether the last one was scalar or not. If the last instruction is a non-scalar one, we can artificially add a new non-terminal variable and the corresponding instruction so as to always have $L + 1$ instructions with the first L of them being non-scalar.

Now, each new non-scalar instruction can be written as $u_i := U_i(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ for a suitable choice of parameters $\vec{\alpha}, \vec{\beta}$ and $\vec{\gamma}$. Finally, $\vec{\beta}$ can be chosen such that the output variable u_{L+1} can be represented by $U_{L+1}(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ (in our first example, $u_3 := u_2$, and in the second one, $u_3 := t_1 + t_2 + u_2$).

We defined $\Gamma_U = \{u_i := U_i(\vec{\alpha}, \vec{\beta}, \vec{\gamma})\}_{i=\overline{1,L+1}}$, an universal structure that parameterizes the family $SLP_L(F, T)$. Moreover, counting the number of parameters, we obtain Eq. (9). Actually, Γ_U is not exactly an SLP, because only non-scalar instructions are considered and the operators are applied to lineal combinations of previous non-terminal variables and terminal symbols. □

For each Γ in $SLP_L(F, T)$ we denote by $U(\Gamma)$ the vector $(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) \in \mathbb{R}^k$ obtained in the proof of Lemma 1, where the value of k is given by Eq. (9).

4. Straight Line Program Genetic Programming

The general definition of GP as an evolution method makes it independent of the data structure used for the representation of the evolved programs. Usually these programs are represented by directed trees with ordered branches (see Koza (1992)). Linear Genetic Programming (LGP) is a GP variant that evolves sequences of instructions from an imperative programming language or from a machine language. The term linear refers to the data structure used for the program representation, constituted by the sequence of instructions or the program itself. For a complete overview on LGP, see Brameier and Banzhaf

(2007). Inside the LGP paradigm we have proposed in Alonso et al. (2008) a new data structure: the straight line program (SLP). An SLP consists of a finite sequence of computational assignments, each assignment being obtained by applying some function to a set of arguments that can be variables, constants or precomputed results. The SLP structure can describe complex computable functions since it can reuse previously computed results during the evaluation process.

In what follows, we will reconsider the main steps of GP strategies using SLP to represent programs. To generate the initial population, the well-known methods for trees (see Koza (1992)) can be easily adapted to SLPs. To compute the fitness function in a GP process solving a particular problem, the computation of the function Φ_Γ is often necessary.

4.1. SLP Crossover

Taking into account that an SLP is a finite sequence of assignments, and considering each assignment as a gene, it is easy to define for our data structure the usual recombination operators for binary strings: one-point crossover, two-point crossover or uniform crossover.

Note that although the SLPs of a population may have different lengths, we can assume without loss of generality that all of them have the same length. Indeed, let L be the maximum length of the SLPs of a population, and consider any SLP $\Gamma = \{u_1, \ldots, u_l\}$ over F and T with output set $O(\Gamma) = \{u_{i_1}, \ldots, u_l\}$ such that $l < L$. Then, we can insert $L - l$ randomly constructed new assignments in Γ just before u_l to form a new SLP $\Gamma' = \{u_1, \ldots, u_{l-1}, v_1, \ldots v_{L-l}, u_l\}$ with $O(\Gamma') = O(\Gamma) = \{u_{i_1}, \ldots, u_l\}$ such that Γ and Γ' compute the same function and Γ' has length L.

For SLP-GP we have designed a specific crossover operation that produces another type of information exchange between the two selected parents. The objective is to carry subexpressions from one parent to the other. A subexpression is captured by an instruction u_i and all the instructions that are used to compute the expression over the set of terminals represented by u_i. The formal

procedure to perform this crossover is as follows.

Definition 3. *(SLP-crossover) Let* $\Gamma_1 = \{u_1, \ldots, u_L\}$ *and* $\Gamma_2 = \{u'_1, \ldots, u'_L\}$ *be two SLPs over* F *and* T*. First, a position* k *in* Γ_1 *is randomly selected,* $1 \leq k \leq L$*. Then we consider* $S_{u_k} = \{u_{j_1}, \ldots, u_{j_m}\}$ *with* $u_{j_m} = u_k$ *the set of variables in* Γ_1 *used to compute the subexpression of instruction* I_k*. Next, we randomly select a position* t *in* Γ_2*,* $m \leq t \leq L$ *and modify* Γ_2 *by making the substitution of the subset of instructions* $\{u'_{t-m+1}, \ldots, u'_t\}$*, by the instructions in* S_{u_k} *suitably renamed. The renaming function* $\mathcal{R}$ *over* S_{u_k} *is defined by* $\mathcal{R}(u_{j_i}) = u'_{t-m+i}$ *for all* $i \in \{1, \ldots, m\}$*. With this process we obtain the first offspring from* Γ_1 *and* Γ_2*. For the second offspring we symmetrically repeat the strategy, but now beginning by randomly selecting a position* k' *in* Γ_2*.*

Example 2. *(SLP-crossover) Consider* $F = \{*, +, -\}$*,* $L = 4$ *and* $T = \{x, y\}$*. Let* Γ_1 *and* Γ_2 *be the following two SLPs:*

$$\Gamma_1 \equiv \begin{cases} u_1 := x * y \\ u_2 := u_1 + y \\ u_3 := x - y \\ u_4 := u_2 * u_3 \end{cases} \qquad \Gamma_2 \equiv \begin{cases} u'_1 := x * x \\ u'_2 := u_1 + y \\ u'_3 := u_1 * y \\ u'_4 := u_3 - u_2 \end{cases}$$

If $k = 1$ *then* $S_{u_1} = \{u_1\}$*, and* t *must be selected in* $\{1, 2, 3, 4\}$*. For instance, for* $t = 2$*, the first offspring is*

$$\Gamma'_1 \equiv \begin{cases} u'_1 := x * x \\ \mathbf{u'_2} := \mathbf{x} * \mathbf{y} \\ u'_3 := u_1 * y \\ u'_4 := u_3 - u_2 \end{cases}$$

For the second offspring, if the selected position in Γ_2 *is* $k' = 3$*, then* $S_{u'_3} = \{u'_1, u'_3\}$ *and* t *must be selected in* $\{2, 3, 4\}$*. Now, if* $t = 2$*, the second offspring is*

$$\Gamma'_2 \equiv \begin{cases} \mathbf{u_1} := \mathbf{x} * \mathbf{x} \\ \mathbf{u_2} := \mathbf{u_1} + \mathbf{y} \\ u_3 := x - y \\ u_4 := u_2 * u_3 \end{cases}$$

4.2. SLP Mutation

The mutation operation in the SLP structure consists of a change in one of its instructions. This change can be either the substitution of a complete instruction by another one randomly generated, or just a small modification of one of the arguments of a certain instruction. Formally:

Definition 4. *(SLP-mutation) Let* $\Gamma = \{u_1, \ldots, u_L\}$ *be an SLP over* F *and* T*. Let* $u_i = f(\alpha_1, \ldots, \alpha_n)$ *be the selected mutation point, where* $f \in F$ *and* $\alpha_k \in T \cup \{u_1, \ldots, u_{i-1}\}$*. The mutation of* Γ *at point* i *yields*

$$\Gamma' = \{u_1, \ldots, u_{i-1}, u_i', u_{i+1}, \ldots, u_L\},$$

where u_i' *could be either* $f(\alpha_1, \ldots, \alpha_{j-1}, \alpha_j', \alpha_{j+1}, \ldots, \alpha_n)$*, with* $j \in \{1, \ldots, n\}$*,* $\alpha_j' \in T \cup \{u_1, \ldots, u_{i-1}\}$ *and* $\alpha_j \neq \alpha_j'$*, or a new generated instruction. Note that* j *and* α_j' *are both randomly selected.*

Example 3. *Consider* F, L *and* T *as above*

$$\Gamma \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ u_3 := x + u_2 \\ u_4 := u_3 - y \end{cases}$$

Assume that u_3 *is the selected mutation point, that the operator is conserved, and only one argument is randomly selected to mutate. For example, assume that* x *is substituted by* u_1*. The SLP generated by this mutation is the following*

$$\Gamma' \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ \mathbf{u_3 := u_1 + u_2} \\ u_4 := u_3 - y \end{cases}$$

5. Pfaffian Functions and VCD of Formulas

In this section we introduce some basic notions, tools and known results concerning the geometry of sets defined by boolean combinations of sign condi-

tions over Pfaffian functions, also called semi-Pfaffian sets in the mathematical literature (see Gabrielov and Vorobjov (2004) for a complete survey).

Definition 5. *Let $U \subset \mathbb{R}^n$ be an open domain. A Pfaffian chain of order $q \geq 1$ and degree $D \geq 1$ in U is a sequence of real analytic functions $f_1, \ldots, f_q$ on U satisfying a system of differential equations*

$$\frac{\partial f_i}{\partial x_j} = P_{i,j}(\vec{x}, f_1(\vec{x}), \ldots, f_i(\vec{x})), \forall i \in \overline{1,q}, \forall j \in \overline{1,n},$$

where $\vec{x} = (x_1, \ldots, x_n)$ and $P_{i,j} \in [\vec{x}, y_1, \ldots, y_i]$ are polynomials of degree at most D.

A function f on U is called a Pfaffian function of order q and degree (D, d) with $D, d \geq 1$ if

$$f(\vec{x}) = P(\vec{x}, f_1(\vec{x}), \ldots, f_q(\vec{x})),$$

where $P \in [\vec{x}, y_1, \ldots, y_q]$ is a polynomial of degree at most d and $f_1, \ldots, f_q$ is a Pfaffian chain of order q and degree D.

Example 4. *The following functions are Pfaffian.*

1. $\sin(x)$, *defined on* $(-\pi + 2\pi r, \pi + 2\pi r)$ *with* $r \in \mathbb{N}$,
2. $\tan(x)$, *defined on* $(-\pi/2 + \pi r, \pi/2 + \pi r)$ *with* $r \in \mathbb{N}$,
3. e^x *defined on* $\mathbb{R}$,
4. $\log(x)$ *defined on* $(0, \infty)$,
5. $1/x$ *defined on* $(-\infty, 0) \cup (0, \infty)$,
6. $\sqrt{x}$ *defined on* $[0, \infty)$.

Actually, all analytic algebraic functions are Pfaffian on suitable domains.

Given a class $\mathcal{H} = \{\Phi_i : i \in I\}$ with $\Phi_i : \mathbb{R}^n \to \mathbb{R}$, one can measure its classification capacity as the VCD of $\mathcal{C}_\mathcal{H}$, where $\mathcal{C}_\mathcal{H} = \{S_i : i \in I\}$ and $S_i = \{\vec{x} \in \mathbb{R}^n : \Phi_i(\vec{x}) > 0\}$ (i.e., each function Φ_i is seen as a decision boundary that divides the elements of the space $\mathbb{R}^n$ into S_i or $\mathbb{R}^n \backslash S_i$, depending on whether their image through Φ_i is positive or not). In many applications, functions

in $\mathcal{H}$ are indexed using vectors of real numbers, that is, $I = \mathbb{R}^N$ for some $N \geq 1$, and in this case, one may define a meta-function $\Phi : \mathbb{R}^{N+n} \to \mathbb{R}$ by $\Phi(\vec{y}, \vec{x}) = \Phi_{\vec{y}}(\vec{x})$ with $\vec{y} \in \mathbb{R}^N$ and $\vec{x} \in \mathbb{R}^n$. In this case, $\mathcal{C}_{\mathcal{H}} = \{S_{\vec{y}} : \vec{y} \in \mathbb{R}^N\}$ and $S_{\vec{y}} = \{\vec{x} \in \mathbb{R}^n : \Phi(\vec{y}, \vec{x}) > 0\}$.

Now, if condition $\Phi(\vec{y}, \vec{x}) > 0$ can be written as a linear combinations of s atomic formulas of the form $\tau_i(\vec{y}, \vec{x}) > 0$ or $\tau_i(\vec{y}, \vec{x}) = 0$, where each τ_i is a C^∞ (infinitely differentiable) function from $\mathbb{R}^{N+n}$ to $\mathbb{R}$, one can use a result published in Karpinski and Macintyre (1997) to estimate an upper bound for the VCD of the class $\mathcal{C}_{\mathcal{H}}$ under certain assumptions about the τ_i.

More precisely, let $\vec{v}_1, \ldots, \vec{v}_V$ be elements of $\mathbb{R}^n$. Form the ($\leq sV$) many infinitely differentiable functions $\tau_i(\vec{y}, \vec{v}_j)$ from $\mathbb{R}^N$ to $\mathbb{R}$ and choose $r \leq N$ of them, denoted $\Theta_1, \ldots, \Theta_r$. Moreover, define $\Theta : \mathbb{R}^N \to \mathbb{R}^r$ by

$$\Theta(\vec{y}) = \langle \Theta_1(\vec{y}), \ldots, \Theta_r(\vec{y}) \rangle .$$

By Sard's theorem (Milnor and Weaver (1965)), the set of non-regular values $\langle \epsilon_1, \ldots, \epsilon_r \rangle$ of Θ in $\mathbb{R}^r$ has Lebesgue measure zero (recall that $\langle \epsilon_1, \ldots, \epsilon_r \rangle$ is a regular value of Θ if $\Theta^{-1}(\langle \epsilon_1, \ldots, \epsilon_r \rangle)$ is either empty or an $(N-r)$-dimensional C^∞-submanifold of $\mathbb{R}^N$).

This motivates the following assumption on the τ_i. Assume that there is a bound B, independent on the $\vec{v}_j$, r and $\epsilon_1, \ldots, \epsilon_r$ such that if $\Theta^{-1}(\langle \epsilon_1, \ldots, \epsilon_r \rangle)$ is an $(N-r)$-dimensional submanifold of $\mathbb{R}^N$ then $\Theta^{-1}(\langle \epsilon_1, \ldots, \epsilon_r \rangle)$ has at most B connected components. Then,

$$VCD(\mathcal{C}_{\mathcal{H}}) \leq 2 \log_2 B + (16 + 2 \log_2 s)N$$

(see Theorem 2 on page 171 of Karpinski and Macintyre (1997)).

Let now $\tau_i(\vec{y}, \vec{x})$ be polynomials of degree at most d in the $\vec{y}, \vec{x}$ and in functions $f_1, \ldots, f_q$ which form a Pfaffian chain of length q and degree D. Karpinski and Macintyre (1997) show that if $\Theta : \mathbb{R}^N \to \mathbb{R}^r$ defines a manifold of dimension $N - r$, then such a bound exists, and it is given by

$$B \leq 2^{Nq(Nq-1)/2} \cdot d^N \cdot (N(d+D))^N \cdot (N^2(d+D))^{Nq}.$$

6. VCD Bounds of Straigth Line Programs with Pfaffian Instructions

In this section we show how we can extend the above mentioned results to the family of SLPs that use Pfaffian operators. As we shall shortly see, our upper bound does not depend on the length of the SLPs. In previous results about the VC dimension of programs or families of computation trees, a bound approximated by the number of steps of the program execution or by the height of the computation tree was needed. In our case, we only need a bound for the number of non-scalar SLP instructions (that is, instructions involving operations which are not in $\{+,-\}$, such as multiplication, division, or the *sign* function defined below).

$$sign(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otheriwse} \end{cases}$$

Next we present and prove our main theoretical result about the VCD of families of SLPs with Pfaffian instructions and bounded number of non-scalar operations.

Theorem 1. *Let $\mathcal{H} = \{\Phi_\Gamma : \Gamma \in SLP_L(F,T)\}$, with $T = \{x_1,\ldots,x_n\} \cup \{c_1,\ldots,c_p\}$ and $F = \{+,-,*,/,sign\} \cup \{f_1,\ldots,f_q\}$, where $f_1,\ldots,f_q$ is a Pfaffian chain of order q and degree D. Then the VCD of $\mathcal{C}_\mathcal{H}$ is of order*

$$\mathcal{O}(N^2q^2 + N(1+q)\log_2 N(2+D)) \tag{10}$$

where

$$N = \mathcal{O}(L^2A + LA(n+p) + Lq) \tag{11}$$

Proof. Let $\Gamma = \{u_1,\ldots,u_l\}$ be an arbitrary SLP in $SLP_L(F,T)$. Note that Φ_Γ is a function from $\mathbb{R}^n$ to $\mathbb{R}$ which depends on the values chosen for the vector $\vec{c} = (c_1\ldots,c_p)$ of constants. For instance, $\Phi_\Gamma = 2x_2(x_1+1)^2 - 2x_2$ in Example 1 because $C = \{1,2\}$. The same Γ would represent a different function, namely $\Phi_\Gamma = 2x_2x_1^2 - 2x_2$, if C was $\{0,1\}$. So, any parametrization of the SLP Γ should also include, apart from the vector $(\vec{\alpha},\vec{\beta},\vec{\gamma}) = U(\Gamma) \in \mathbb{R}^k$ (recall that $k = L\left(4+q+A\left(m+\frac{L-1}{2}\right)\right)+m$, where A is the maximum of the arities of the

functions in F and $m = n+p$), the set C of constants. We denote by $\vec{y}$ the vector $(\vec{\alpha}, \vec{\beta}, \vec{\gamma}, \vec{c}) \in \mathbb{R}^N$ of parameters ($N = L\left(4 + q + A\left(n + p + \frac{L-1}{2}\right)\right) + n + 2p$) and by Γ_U the universal pseudo-SLP from Lemma 1.

Let us rename our terminal symbols by $\{t_1, \ldots, t_m\}$. Each non-scalar instructions in Γ_U is of one of the following types:

1. $u_i := P_1^i * P_2^i$,
2. $u_i := P_1^i / P_2^i$,
3. $u_i := \sum_{k=1}^{q} (\prod_{j=1}^{k-1} (1 - \gamma_j^i)) \gamma_k^i f_k(P_1^i, \ldots, P_A^i)$
4. $u_i := sign(P_1^i)$,

where $P_k^i = \sum_{j=1}^{m} \beta_j{}^{(i,k)} t_j + \sum_{j=1}^{i-1} \beta_{m+j}^{(i,k)} u_j, \forall k \in \overline{1, A}$.

Γ_U includes at most L sign instructions. For each sign instruction u_i as above, let $U_i = \{u_{i_1}, \ldots, u_{i_{k_i}}\}$ be the set of sign instructions that are computed before u_i in an execution of Γ_U on an input $\vec{x} = (x_1, \ldots, x_n)$. Then in the computation of u_i, the variables in U_i have fixed values in $\{0, 1\}$. Hence the sign condition u_i can be considered as a function that depends on $\vec{x}, \vec{y}$ and $u_1, \ldots, u_{i-1}$, where the involved previous sign instructions verify $u_{i_j} = \delta_j \in \{0, 1\}$, $u_{i_j} \in U_i$. Thus, we have an atomic predicate $\tau_{\vec{\delta}_i}(\vec{y}, \vec{x})$ for each $\vec{\delta}_i = (\delta_1, \ldots, \delta_{k_i}) \in \{0, 1\}^{k_i}$.

We define for each sign instruction u_i in Γ_U the following set of atomic predicates:

$$W(u_i) = \{\tau_{\vec{\delta}_i}(\vec{y}, \vec{x}) / \vec{\delta}_i = (\delta_1, \ldots, \delta_{k_i}) \in \{0, 1\}^{k_i}\} \tag{12}$$

The elements of $W(u_i)$ are terms of the first order language L over the real numbers, with symbols $+, -, *, f_1, \ldots, f_q$ and $<$ for the order. As $k_i < i, W(u_i)$ contains at most $2 \cdot 2^{i-1} = 2^i$ atomic predicates.

After the above considerations, given Γ_U, the universal pseudo-SLP for the family $SLP_L(F, T)$, the condition $\Phi_\Gamma(\vec{x}) > 0$ can be written equivalently with a formula that consists of at most $2^{L+1} - 2$ atomic predicates. This is a bound for the number of atomic predicates in $W = \bigcup_{i=1}^{L} W(u_i)$.

We now apply a projection technique to get a bound on the VCD of the

class $\mathcal{C}_{\mathcal{H}}$. Recall that we need to know a bound on the number of connected components of a manifold of dimension $N-r$ defined by conditions of the form

$$\tau(\vec{y},\vec{v}_i)-\epsilon_i=0, 1\leq i\leq r\ (r\leq N)$$

where $\vec{v}_i$ are constants in $\mathbb{R}^n$ and $\tau(\vec{y},\vec{v}_i)$ is an L-term of W. Assume, for simplicity, that $\tau(\vec{y},\vec{v}_i)\in W(u_i)$ and let $(\delta_1,\ldots,\delta_{k_i})\in\{0,1\}^{k_i}$ be the associated fixed values according to Eq. (12). In the sequel, we shall use the estimates given in Theorem 2 of Karpinski and Macintyre (1997).

For each instruction $u_j\in\Gamma_U$ with $j\in\{1,\ldots,u_{i-1}\}$ we add the following equations:

1. if $u_j=P_1^i * P_2^i$, we add the equation

$$u_j-P_1^i * P_2^i=0 \tag{13}$$

2. if $u_j=P_1^i/P_2^i$, we add the equation

$$u_j * P_2^i-P_1^i=0 \tag{14}$$

3. if $u_j=\sum_{k=1}^{q}(\prod_{l=1}^{k-1}(1-\gamma_l^i))\gamma_k^j f_k(P_1^i,\ldots,P_A^i)$ is a Pfaffian instruction, we add the equation

$$u_j-\sum_{k=1}^{q}(\prod_{l=1}^{k-1}(1-\gamma_l^i))\gamma_k^j f_k(P_1^i,\ldots,P_A^i)=0 \tag{15}$$

4. if u_j is a sign instruction, we add the equation

$$u_j-\delta_j=0,\ \delta_j\in\{0,1\} \tag{16}$$

Finally, we add the equation

$$u_i-\epsilon_i=0 \tag{17}$$

Let $S(\vec{v}_1,\ldots,\vec{v}_r,\vec{y},\epsilon_1,\ldots,\epsilon_r,\vec{u})$ be the system obtained by Eqs. (13), (14), (15), (16) and (17) for all i, $1\leq i\leq r$, where we denoted by $\vec{u}$ the vector $(u_1,\ldots,u_{L+1})$ of variables. Then we can state the following:

1. $S(\vec{v}_1, \ldots, \vec{v}_r, \vec{y}, \epsilon_1, \ldots, \epsilon_r, \vec{u})$ implies $\tau(\vec{y}, \vec{v}_i) = \epsilon_i$ for all $i \in \{1, \ldots, r\}$
2. If $\tau(\vec{y}, \vec{v}_i) = \epsilon_i$ for all $i \in \{1, \ldots, r\}$, then there exists an unique $\vec{u}$ that verifies the equations of system $S(\vec{v}_1, \ldots, \vec{v}_r, \vec{y}, \epsilon_1, \ldots, \epsilon_r, \vec{u})$.
3. The set defined in the variables $\vec{y}$ by equations $\tau(\vec{y}, \vec{v}) = \epsilon_i$, $1 \leq i \leq r$ is homeomorphic to the set defined in the variables $\vec{y}, \vec{u}$ by the system $S(\vec{v}_1, \ldots, \vec{v}_r, \vec{y}, \epsilon_1, \ldots, \epsilon_r, \vec{u})$. So, either both of them or none of them are manifolds.

Thus, we have constructed a system S that contains at most $r(L+1) \leq N(L+1)$ equations. These equations are polynomials of degree at most 2 in the polynomial ring $\mathbb{R}[\vec{y}, \vec{u}, f_1, \ldots, f_q]$. Now we can use Khovanskii's estimates on system S, assuming that it defines an $N-r$ dimensional manifold. The number B of connected components defined by system S is bounded by

$$2^{Nq(Nq-1)/2} \cdot 2^N \cdot (N(2+D))^N \cdot (N^2(2+D))^{Nq}$$

where $N = L\left(4 + q + A\left(n + p + \frac{L-1}{2}\right)\right) + n + 2p)$.

The final result displayed in Eq. (10) follows from Theorem 2 of Karpinski and Macintyre (1997) taking $s = 2^{L+1} - 2$, that is an upper bound for the number of atomic predicates, and considering the above bound for the parameter B.

□

Remark 1. *Simplifying the Eq.* (10) *after making the substitution of* N *by the expression displayed in Eq.* (11)*, the following result is obtained:*

$$VCD(\mathcal{C}_{\mathcal{H}}) = \mathcal{O}(q^2L^2(A(L+n+p)+q)^2 +$$

$$L(A(L+n+p)+q)(1+2q)\log_2 L(A(L+n+p)+q)(2+D))$$

Note that for small values of D *the above expression is dominated by the first term. Hence, in these conditions, the VCD of the family* $\mathcal{C}_{\mathcal{H}}$ *is polynomial of degree four in the number of non-scalar operations.*

7. Experimentation

In this section we present the results obtained after an important experimental phase, in which symbolic regression problem instances were solved using SLPs over a fixed set of Pfaffian functions as the underlying structure of the model. We implemented a GP algorithm with the genetic operators described in Section 2 and the fitness regularization function from Eq. (7) (the model known as structural risk minimization). Although Theorem 1 presents an upper bound for the VCD of our families of SLPs that is of degree four in the number L of non-scalar operations, our experiments showed that the best option is to consider h as the square root of L. This is not a contradiction with the theorem. Note that in our theoretical result we give an upper bound for the VCD. In practice, L^4 is too big as a regularization term for our symbolic regression problem instances.

We introduced additive Gaussian noise in the sample set $z = (x_i, y_i)_{1 \leq i \leq m}$. Hence, for a target function g, the sample set verifies: $y_i = g(\vec{x}_i) + \epsilon$, where ϵ is i.i.d. zero mean random error.

For the first experiment we have considered a set of 500 multivariate polynomials with real coefficients whose degrees are bounded by 5. The number of variables varies from 1 to 5, with 100 polynomials for each case. In the second experiment, the target function was chosen from a pool of fixed specific multivariate functions (see Section 7.2 for details). In these two experiments, for every execution, the sample set is constituted by points with added noise. Finally, a third experiment was performed considering some well-known real benchmark problems. In all cases, when the GP process finishes, the best individual is selected as the proposed model for the corresponding target function.

Frequently, when different Genetic Programming strategies solving symbolic regression instances are compared, the quality of the selected model is evaluated by means of its corresponding fitness value over the sample set. But a fitness value close to zero may be due to a very complex model that overfits the data in the training set, and that may not be able to generalize well on some unseen

points of the target function. In fact, when the sample set is corrupted with noise, models with very low fitness values are most often a bad choice.

Therefore, in order to avoid overfitting, we have considered a new set of unseen points, generated without noise using the target function. This new set of examples is known as the test set or validation set. So, let $(\vec{x}_i, y_i)_{1 \leq i \leq n_{test}}$ be a validation set for the target function $g(\vec{x})$ (i.e., $y_i = g(\vec{x}_i)$) and let $f(\vec{x})$ be the model estimated from the training data. Then the prediction risk $\varepsilon_{n_{test}}$ is defined by the mean square error between the values of f and the true values of the target function g over the validation set:

$$\varepsilon_{n_{test}}(f) = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (f(\vec{x}_i) - y_i)^2$$

Regarding the problem of tuning parameters in our experiments, in the present work we have used a generalist approach (see Eiben and Smit (2011)), since our objective is to compare different regularization methods. For this purpose, it is reasonable to fix the same set of parameters for all problems under study and vary the regularization function. The chosen parameters are standard in the literature (for example, see Poli et al. (2008)).

7.1. First Experiment

We denote by $\mathrm{P}_n[X]$ the set of polynomials over $X = (x_1, \ldots, x_n)$ with $1 \leq n \leq 5$. The individuals are SLPs over $F = \{+, -, *, /, sqrt, \sin, \cos, \exp\}$. In order to avoid errors generated by divisions by zero, instead of the traditional division we used in our computation the operation usually named "protected division", that returns 1 if the denominator is zero. Besides the variables x_i which take values in $[-1, 1]$ for all $i \in \overline{1,5}$, the terminal set also includes five constants c_i, $1 \leq i \leq 5$, randomly generated in $[-1, 1]$. Note that although the target functions are polynomials, our set F not only contains the operators of sum, difference and product, but also some other Pfaffian functions. This situation increments considerably the search space. Nevertheless, this is quite realistic, since in practice we usually do not know if the target function is or not a polynomial.

The parameters for the GP process are the following: population size $M = 200$, probability of crossover $p_c = 0.9$, probability of mutation $p_m = 0.05$ and tournament selection of size 5. The real length of the SLPs in the population is bounded by 40. Elitism and a particular generational replacement are used, so the offsprings do not necessarily replace their parents. After a crossover we have four individuals: two parents and two offsprings. We select the two best individuals with the smallest fitness values. The motivation is to prevent premature convergence and to maintain diversity in the population.

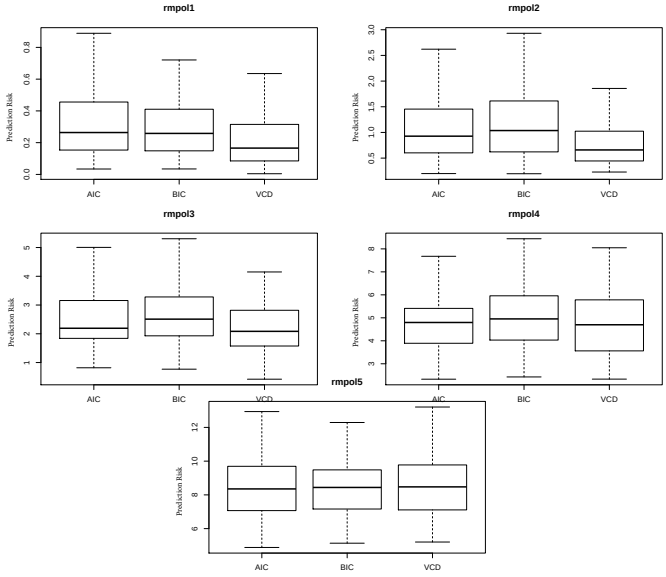

Figure 2: Empirical distribution of the executions: first experiment

Since we are considering multivariate polynomials as target functions, the difficulty of the problem instance increases with the number of variables. Hence, to vary the size of the sample set as a function of the number of variables is a reasonable decision. Note that an upper bound for the number of monomials in a polynomial with n variables and degree d is $4 \cdot d^{n+1}$ and this is also a quite good estimation for a lower bound of the size of the sample set. Thus, in our case we have considered sample sets of size $4 \cdot 5^{n+1}, 1 \leq n \leq 5$. In this experiment, one execution for each strategy (AIC, BIC, VCD) has been performed over the 500 generated target functions. In every execution the process finishes after 250

generations were completed. Finally, the validation set consists of a number of unseen points that is equal to two times the size of the sample set.

In Fig. 2 we present the empirical distribution of the executions of the three compared strategies over the sets of polynomials. We have separated the polynomial sets by the number of variables, from one to five. These empirical distributions are displayed using standard box plot notation with marks at the best execution, 25%, 50%, 75%, and the worst execution, always considering the prediction risk of the selected model.

We also include tables that show means and variances of the prediction risk values, as well as the prediction risk of the best obtained model for each method (see Tables 1 and 2).

Table 1: Values of means and variances: first experiment

Problem instance	AIC		BIC		VCD	
	μ	σ	μ	σ	μ	σ
$P_1[X]$	0.39	0.42	0.38	0.41	0.24	0.27
$P_2[X]$	1.17	0.86	1.25	0.85	0.82	0.54
$P_3[X]$	2.62	1.28	2.89	1.57	2.24	0.85
$P_4[X]$	4.79	1.49	5.08	1.74	4.76	1.43
$P_5[X]$	8.51	2.00	8.63	2.16	8.63	2.17

Table 2: Prediction risk of the best execution: first experiment

Problem instance	AIC	BIC	VCD
$P_1[X]$	0.0340	0.0344	0.0043
$P_2[X]$	0.20	0.19	0.22
$P_3[X]$	0.82	0.77	0.42
$P_4[X]$	2.33	2.43	2.33
$P_5[X]$	4.88	5.13	5.20

As we can see from the above figures and tables, it seems that VCD regularization performs better than the well-known regularization methods AIC and BIC. This is more obvious for the polynomials in up to three variables and not

so clear for the rest of the polynomial sets. This could be because for polynomials with four and five variables, as they constitute more complex problem instances, a larger number of generations would be necessary in the evolutive process.

In order to confirm the comparative results of the studied strategies, we have made crossed statistical hypothesis tests. The obtained results are showed in Table 3. Roughly speaking, the null-hypothesis in each test with associated pair (i, j) is that strategy i is not better than strategy j. Hence, if value a_{ij} in Table 3 is less than a significance value α, we can reject the corresponding null-hypothesis.

Table 3: Crossed statistical hypothesis tests: first experiment

$P_1[X]$	AIC	BIC	VCD
AIC		0.91	1
BIC	0.30		1
VCD	3.93e-4	1.1e-3	

$P_2[X]$	AIC	BIC	VCD
AIC		3.91e-2	1
BIC	0.96		1
VCD	1.86e-5	1.12e-7	

$P_3[X]$	AIC	BIC	VCD
AIC		7.7e-2	0.99
BIC	0.99		1
VCD	1.2e-2	3.93e-4	

$P_4[X]$	AIC	BIC	VCD
AIC		0.12	0.90
BIC	0.77		0.99
VCD	0.27	2.7e-3	

$P_5[X]$	AIC	BIC	VCD
AIC		0.44	0.52
BIC	0.77		0.78
VCD	0.61	0.37	

Taking into account the results of the crossed statistical hypothesis tests with a significance value $\alpha = 0.05$, we can confirm that our proposed regularization method based on the VC dimension of families of SLPs is the best of the studied strategies for the considered sets of multivariate polynomials.

7.2. Second Experiment

In this case we have run the algorithms on instances associated to three groups of target functions. The first group consists of the following three polynomial functions that were also used for experimentation in Topchy and Punch (2001) and Vanneschi et al. (2006). For all functions, variables take values in the range $[-100, 100]$ and constants are randomly generated in $[0, 1]$.

$F(x, y, z) = (x + y + z)^2 + 1$

$G(x, y, z) = x/2 + y/3 + 2z/3$

$K(x, y, z, w) = x/2 + y/4 + z/6 + w/8$

The second group includes three functions proposed in Cherkassky and Yunkian (2003) for a similar study on several regularization strategies, but considering trees as the structure that represents the models.

$$g_1(x) = \begin{cases} 4x^2(3 - 4x), & \text{if } x \in [0, 0.5] \\ 4x(4x^2 - 10x + 7)/3 - 3/2, & \text{if } x \in (0.5, 0.75] \\ 16x(x - 1)^2/3, & \text{if } x \in (0.75, 1] \end{cases}$$

$g_2(x) = \sin^2(2\pi x),\ x \in [0, 1]$

$g_3(x) = \left(\sin\sqrt{x_1^2 + x_2^2}\right)/(x_1^2 + x_2^2),\ x_1, x_2 \in [-5, 5]$

Finally, the third group of target functions is constituted by five unary functions of several classes: trigonometric functions, polynomial functions and one exponential function. These functions are the following

$f_1(x) = x^4 + x^3 + x^2 + x \qquad x \in [-15, 15]$

$f_2(x) = \mathrm{e}^{-\sin(3x)+2x} \qquad x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$

$f_3(x) = \mathrm{e}x^2 + \pi x \qquad x \in [-\pi, \pi]$

$f_4(x) = \cos(2x) \qquad x \in [-\pi, \pi]$

$f_5(x) = \min\{2/x, \sin(x) + 1\} \qquad x \in [0, 15]$

The parameters of the GP (population size, tournament selection, recombination operators, termination condition) are the same as those fixed for the first experiment. For the problem instances associated to the functions of this second experiment, the upper bound for the length of the SLPs is 16, the sample set has 30 points and the validation set is of size 200. The corresponding sets of functions and terminals for each target function are shown in Table 4.

Table 4: Function set and terminal set for the functions. $\mathcal{F} = \{+, -, *, /\}$

Function	Function set	Terminal set
F	$\mathcal{F}$	$\{x, y, z\}$
G	$\mathcal{F}$	$\{x, y, z, c_1, \ldots, c_6\}$
K	$\mathcal{F}$	$\{x, y, z, w, c_1, \ldots, c_6\}$
g_1	$\mathcal{F} \cup \{sign\}$	$\{x, c_1, \ldots, c_5\}$
g_2	$\mathcal{F} \cup \{\sin\}$	$\{x, c_1, \ldots, c_5\}$
g_3	$\mathcal{F} \cup \{sqrt, \sin\}$	$\{x_1, x_2, c_1, \ldots, c_5\}$
f_1	$\mathcal{F} \cup \{sqrt\}$	$\{x\}$
f_2	$\mathcal{F} \cup \{sqrt, \sin, \cos, \exp\}$	$\{x, c_1\}$
f_3	$\mathcal{F} \cup \{\sin, \cos\}$	$\{x, c_1\}$
f_4	$\mathcal{F} \cup \{sqrt, \sin\}$	$\{x, c_1\}$
f_5	$\mathcal{F} \cup \{\sin, \cos\}$	$\{x, c_1\}$

We have performed 100 executions of the three compared methods for each problem instance. The results are presented in Fig. 3 and 4.

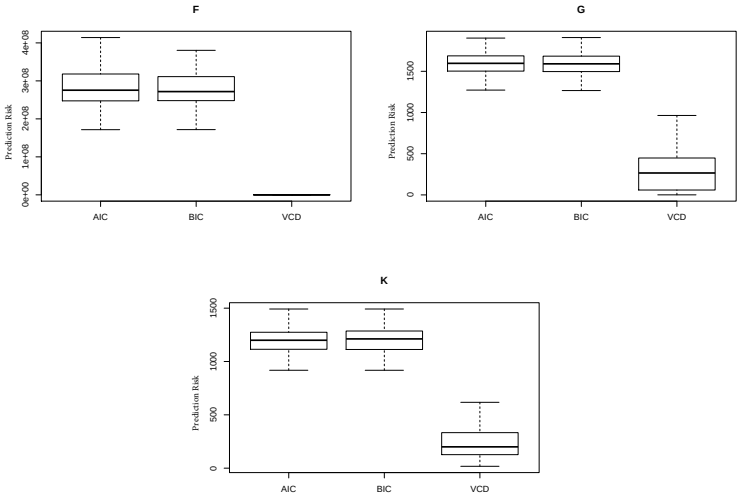

Figure 3: Empirical distribution of the executions: second experiment (first group)

Analyzing the empirical distribution of the executions, the VCD regularization method is clearly the best one for all target functions of the first and third group. There are significant differences in their performance, especially for the functions of the first group, where the AIC and BIC methods found no good solutions in any of the 100 executions.

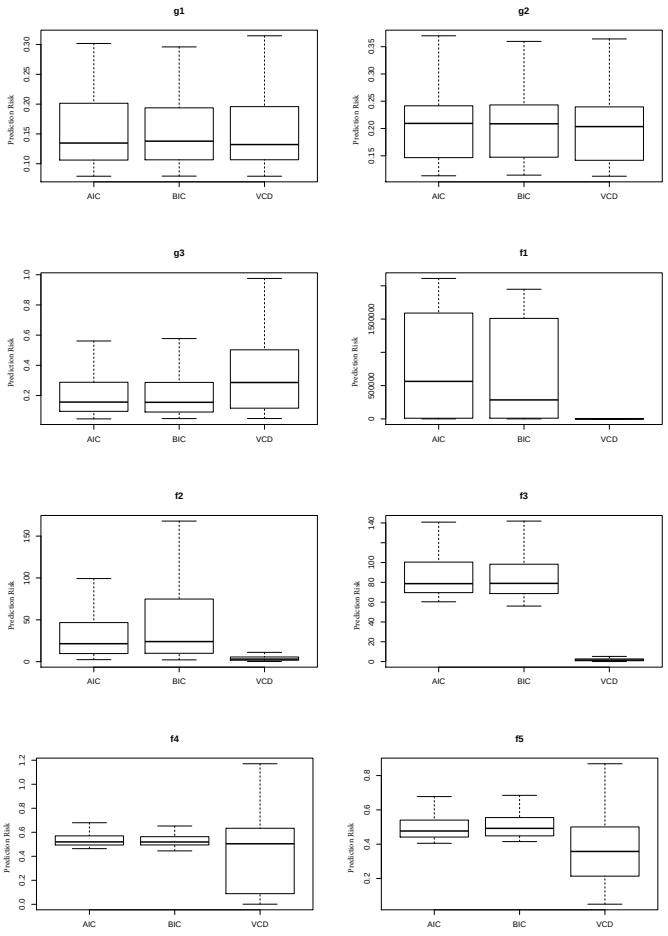

Figure 4: Empirical distribution of the executions: 2nd experiment (2nd and 3rd group)

However the three compared methods have a similar performance over the target functions of the second group. In fact, in the case of the bivariate function g_3, our VCD regularization seems to be worse than AIC and BIC. Considering that g_1 is a discontinuous piecewise polynomial function, g_2 is a sin-square function and g_3 is a two-dimensional sin function, we could conjecture that for noisy problem instances associated with trigonometric or discontinuous target functions the VCD method is not necessarily better than AIC or BIC. A similar situation occurs for the trigonometric functions of the third group f_4 and f_5. On the other hand, AIC and BIC methods always perform quite similar. This

could be because both associated fitness functions (Eqs. (4) and (5)) have the same structure with very similar additive penalization terms.

As we did in the case of the first experiment, for each of the three models that we compare, we present means and variances for the prediction risk values (see Table 5), as well as the prediction risk of the model obtained in the best execution (see Table 6).

Table 5: Values of means and variances: second experiment

Problem	AIC		BIC		VCD	
instance	μ	σ	μ	σ	μ	σ
F	$2.83 \cdot 10^8$	$5.3 \cdot 10^7$	$2.74 \cdot 10^8$	$4.12 \cdot 10^7$	0	0
G	1627.1	236.87	1612.58	217.41	314.15	293.55
K	1199.14	107.03	1203.45	110.30	237.78	170.13
g_1	0.15	0.06	0.15	0.06	0.13	0.19
g_2	0.21	0.06	0.21	0.06	0.20	0.06
g_3	0.20	0.15	0.20	0.15	0.37	0.33
f_1	$7.6 \cdot 10^5$	$7.8 \cdot 10^5$	$7.5 \cdot 10^5$	$7.6 \cdot 10^5$	11.23	27.01
f_2	34.3	34.4	44.76	44.92	4.99	5.49
f_3	84.33	20	81.35	18.87	1.53	1.90
f_4	0.54	0.11	0.54	0.09	0.43	0.32
f_5	0.49	0.08	0.50	0.07	0.36	0.19

Table 6: Prediction risk of the best execution: second experiment

Problem instance	AIC	BIC	VCD
F	$1.7 \cdot 10^8$	$1.7 \cdot 10^8$	0
G	1272.03	1266.42	0.69
K	917.83	917.67	17.46
g_1	0.0791	0.0793	0.0791
g_2	0.11	0.11	0.11
g_3	0.0453	0.0471	0.0475
f_1	3.02e-24	3.24e-24	2.26e-24
f_2	2.43	2.17	0.17
f_3	60.38	56.06	0.0277
f_4	5.5e-3	0.011	9.9e-4
f_5	0.15	0.42	0.0503

Results displayed in Table 5 and 6 agree with the above comments on the comparative study of the three methods. Note that VCD regularization obtains the lowest mean prediction risk for almost all the proposed target functions.

The crossed statistical hypothesis tests shown in Table 7 confirm VCD as the best method for the problem instances associated to the functions of the first and third group.

Table 7: Crossed statistical hypothesis tests: second experiment

F	AIC	BIC	VCD
AIC		0.96	1
BIC	0.2		0.96
VCD	3.7e-44	3.7e-44	
G	AIC	BIC	VCD
AIC		0.91	1
BIC	0.70		1
VCD	2.72e-43	2.72e-43	
K	AIC	BIC	VCD
AIC		0.44	1
BIC	0.85		1
VCD	2.72e-43	2.72e-43	
g_1	AIC	BIC	VCD
AIC		0.66	0.79
BIC	0.75		0.87
VCD	0.63	0.75	
g_2	AIC	BIC	VCD
AIC		0.81	0.90
BIC	0.75		0.96
VCD	0.40	0.69	
g_3	AIC	BIC	VCD
AIC		0.84	1.37e-3
BIC	0.76		8.37e-4
VCD	1	1	

f_1	AIC	BIC	VCD
AIC		0.64	1
BIC	0.52		1
VCD	1.24e-30	2.39e-28	
f_2	AIC	BIC	VCD
AIC		0.19	1
BIC	0.79		1
VCD	1.34e-15	9e-19	
f_3	AIC	BIC	VCD
AIC		0.98	1
BIC	0.37		1
VCD	1.08e-39	1.07e-38	
f_4	AIC	BIC	VCD
AIC		0.86	4.78e-2
BIC	0.43		2.35e-2
VCD	1.09e-5	5.7e-6	
f_5	AIC	BIC	VCD
AIC		0.39	0.77
BIC	0.99		0.77
VCD	4.72e-13	6.5e-15	

7.3. Third Experiment

Finally, we have executed the described evolutionary processes over some sample sets obtained from the KEEL-dataset repository (see Alcalá-Fdez et al. (2011)). KEEL-dataset includes sets of sample points for problems of different

categories. We have selected three data sets corresponding to the following regression problems: *Abalone*, that consists in the prediction of the age of these molluscs from physical measurements; *Ailerons*, which addresses a control problem, flying an F16 aircraft, and *AutoMPG8*, where the objective is to predict fuel consumption of a city cycle. The respective sizes of the data sets are 4177, 13750 and 392. We consider the same general configuration parameters, and SLPs of length bounded by 16.

The set of functions consists of all the functions previously considered in this work and the terminal set contains the variables and a number of constants that equals the number of variables for each problem. That is 8 for *Abalone*, 40 for *Ailerons* and 7 for *AutoMPG8*. In all cases, and for each execution, we have divided the data sets into two equal parts: one of them was considered as the set of examples and the other was used as the validation set. Note that in this experiment the examples are not corrupted by noise. As usually, we have performed 100 executions for each problem instance and method. Fig. 5 displays the empirical distributions of executions for this third experiment.

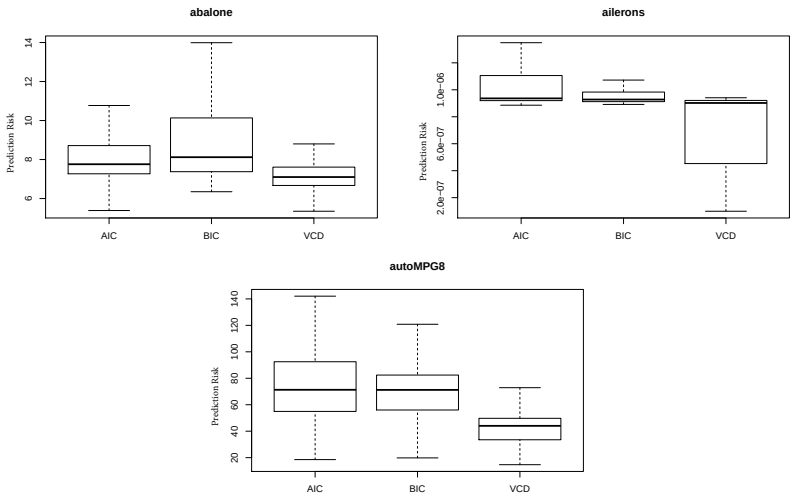

Figure 5: Empirical distribution of the executions: third experiment

Moreover, Tables 8, 9 and 10 depict the results obtained, with the same

structure as in the two previous experiments.

Table 8: Values of means and variances: third experiment

Problem	AIC		BIC		VCD	
instance	μ	σ	μ	σ	μ	σ
Abalone	8.1	1.31	8.78	1.77	7.15	0.67
Ailerons	9.37e-7	2.71e-7	8.83e-7	2.96e-7	6.93e-7	3.06e-7
$AutoMPG8$	73.07	24.31	71.32	23.22	41.76	12.25

Table 9: Prediction risk of the best execution: third experiment

Problem instance	AIC	BIC	VCD
Abalone	5.39	6.35	5.35
Ailerons	1.22e-7	1.01e-7	9.93e-8
$AutoMPG8$	18.53	15.38	14.67

With 40 variables, the *Ailerons* problem seems to be the most easy to learn, as optimal solutions were obtained for the three methods in almost all executions. For the other two problems, the investigated methods also perform quite well. Again, for these selected real problem instances, the VCD regularization method is clearly better than the other two, and the results of the crossed statistical hypothesis tests with a significance value $\alpha = 0.05$ confirm it.

Table 10: Crossed statistical hypothesis tests: third experiment

Abalone	AIC	BIC	VCD
AIC		5.37e-2	1
BIC	0.95		1
VCD	3.36e-6	4.78e-9	

Ailerons	AIC	BIC	VCD
AIC		1	1
BIC	0.12		1
VCD	9.26e-12	5.3e-8	

$AutoMPG8$	AIC	BIC	VCD
AIC		0.76	1
BIC	0.99		1
VCD	1.2e-2	3.93e-4	

7.4. Execution Times

All our experiments have been carried out on a Core 2 Duo E7400 CPU with 2GB RAM and a Gentoo Linux up to date. We report execution times (average values, μ, and variances, σ) in Table 11 below.

Table 11: Execution times for all three experiments (in seconds)

Problem	AIC		BIC		VCD	
instance	μ	σ	μ	σ	μ	σ
$P_1[X]$	5.90	0.44	5.95	0.51	6.63	0.81
$P_2[X]$	9.96	1.60	9.96	1.86	12.53	2.98
$P_3[X]$	32.58	9.90	31.17	9.10	41.74	13.60
$P_4[X]$	157.27	53.15	132.39	42.39	164.81	55.03
$P_5[X]$	307.71	15.70	316.35	15.20	304.17	16.06
F	6.34	0.53	6.18	0.68	4.89	0.60
G	4.06	0.71	4.02	0.67	4.34	0.61
K	3.96	0.71	4.07	0.81	4.16	0.66
g_1	8.20	0.85	8.09	0.78	8.02	0.72
g_2	8.34	0.91	8.45	0.90	8.09	0.73
g_3	8.02	0.73	8.24	0.81	8.50	0.99
f_1	4.72	0.86	4.81	0.94	4.72	0.47
f_2	10.19	1.81	10.34	1.90	11.48	1.60
f_3	3.82	0.43	3.77	0.43	4.14	0.39
f_4	3.53	0.33	3.53	0.27	3.91	0.47
f_5	3.55	0.29	3.60	0.31	3.99	0.40
Abalone	47.92	13.78	45.38	14.51	59.03	17.58
Ailerons	85.34	13.07	85.88	16.17	97.31	24.84
AutoMPG8	6.38	1.55	6.48	1.62	8.22	2.00

Note that running times values are very similar for all three regularization methods discussed in this paper.

8. Conclusions and Future Work

Straight line programs constitute a promising structure for representing models in the GP framework. In this paper we control the complexity of populations of SLPs while they evolve in order to find good models for solving symbolic regression problem instances. The evolving structure is constructed from a set of functions that contains Pfaffian operators. We have considered the VCD as a complexity measure. We have found a theoretical upper bound of the VCD of families of SLPs over Pfaffian operators, which is polynomial in the number of the non-scalar instructions of the family of the SLPs. This bound generalizes similar results obtained for more simple sets of operators such as rational functions. As a consequence of the main result, we propose a regularized fitness function for solving symbolic regression problem instances corrupted with noise using GP. We have compared our fitness function based on the VCD upper bound with two well-known statistical penalization criteria. The experimental results obtained on three different groups of target functions show that our proposed complexity measure and its corresponding penalization criterion is better than the others in almost all studied situations. Of special interest is the group of real problem instances where the VCD regularization method is clearly the best.

As future work, we plan to compare our approach with other non-statistical methods such as Tikhonov regularization. Finally, we want to point out that the bounds we derived based on the VCD are distribution independent. Distribution independence is a desirable property because it guarantees the bounds to hold for any data distribution. On the other hand, the bounds may not be tight for some specific distributions (but still better than the worst case). Rademacher complexity (Koltchinskii (2001); Bartlett et al. (2002); Mendelson (2002)) is a relatively newer notion of complexity, which is distribution dependent and it can be estimated from the sample. We believe that investigating this notion in the context of SLP GP could be a promising data-driven regularization approach.

Acknowledgment

This article significantly extends Alonso et al. (2013), a paper that was presented at the 5th International Joint Conference on Computational Intelligence (IJCCI 2013), where it received the Best Paper Award. This work was partially supported by project BASMATI (TIN2011-27479-C04-04) of Programa Nacional de Investigación and project PAC::LFO (MTM2014-55262-P) of Programa Estatal de Fomento de la Investigación Científica y Técnica de Excelencia, Ministerio de Ciencia e Innovación (MICINN), Spain.

References

Akaike, H., 1970. Statistical prediction information. Ann. Inst. Statistic. Math 22, 203–217.

Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., Garcia, S., Sánchez, L., Herrera, F., 2011. Keel data-mining sofware tool: Data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple–Valued Logic and Soft Computing 17 (2-3), 255–287.

Alonso, C. L., Montaña, J. L., Borges, C. E., 2013. Model complexity control in straight line program genetic programming. In: Rosa, A. C., Dourado, A., Correia, K. M., Filipe, J., Kacprzyk, J. (Eds.), IJCCI 2013 - Proceedings of the 5th International Joint Conference on Computational Intelligence, Vilamoura, Algarve, Portugal, 20-22 September, 2013. SciTePress, pp. 25–36.

Alonso, C. L., Montana, J. L., Puente, J., November 2008. Straight line programs: a new linear genetic programming approach. In: Proceedings of the International Conference on Tools with Artificial Intelligence. pp. 517–524.

Alonso, C. L., Montana, J. L., Puente, J., October 2009. A new linear genetic programming approach based on straight line programs: Some theoretical and experimental aspects. International Journal on Artificial Intelligence Tools 18 (5), 757–781.

Angluin, D., Smith, C., September 1983. Inductive inference: Theory and methods. ACM Computing Surveys 15 (3), 237–269.

Aslam, M. W., Zhu, Z., Nandi, A. K., 2013. Feature generation using genetic programming with comparative partner selection for diabetes classification. Expert Systems with Applications 40 (13), 5402 – 5412.

Bartlett, P. L., Boucheron, S., Lugosi, G., 2002. Model selection and error estimation. Machine Learning 48 (1-3), 85–113.

Berkowitz, S., March 1984. On computing the determinant in small parallel time using a small number of processors. Information Processing Letters 18 (3), 147–150.

Bernardo, Smith, A., 1994. Bayesian Theory. John Willey & Sons.

Bezdek, J. C., Boggavarapu, S., Hall, L. O., Bensaid, A., June 1994. Genetic algorithm guided clustering. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation. IEEE, Orlando, Florida, USA, pp. 34–39.

Bojarczuk, C. C., Lopes, H. S., Freitas, A. A., July/August 2000. Genetic programming for knowledge discovery in chest pain diagnosis. IEEE Engineering in Medicine and Biology Magazine 19 (4), 38–44.

Bojarczuk, C. C., Lopes, H. S., Freitas, A. A., Michalkiewicz, E. L., 2004. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artificial Intelligence in Medicine 30 (1), 27–48.

Brameier, M., Banzhaf, W., 2007. Linear Genetic Programming. Springer.

Burguisser, P., Clausen, M., Shokrollahi, M., 1997. Algebraic Complexity Theory. Springer.

Cano, J. R., Herrera, F., Lozano, M., January 2007. Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. Data & Knowledge Engineering 60 (1), 90–108.

Carreño, E., Leguizamón, G., Wagner, N., September 2007. Evolution of classification rules for comprehensible knowledge discovery. In: Srinivasan, D., Wang, L. (Eds.), Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, Singapore, pp. 1261–1268.

Castelli, M., Vanneschi, L., Manzoni, L., Popovič, A., 2015. Semantic genetic programming for fast and accurate data knowledge discovery. Swarm and Evolutionary Computation. In press, Corrected Proof. Available online at `http://www.sciencedirect.com/science/article/pii/S2210650215000516`.

Castelli, M., Vanneschi, L., Silva, S., 2014. Prediction of the unified Parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. Expert Systems with Applications 41 (10), 4608 – 4616.

Cavaretta, M. J., Chellapilla, K., July 1999. Data mining using genetic programming - the implications of parsimony on generalization error. In: Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A. (Eds.), Proceedings of the Congress of Evolutionary Computation. Vol. 2. IEEE, pp. 1330–1337.

Cherkassky, V., Shao, X., Mulier, F. M., Vapnik, V. N., Sep. 1999. Model complexity control for regression using VC generalization bounds. Trans. Neur. Netw. 10 (5), 1075–1089.

Cherkassky, V., Yunkian, M., 2003. Comparison of model selection fo regression. Neural Computation 15 (7), 1691–1714.

Cherkassky, V. S., Mulier, F., 1998. Learning from Data: Concepts, Theory, and Methods, 1st Edition. John Wiley & Sons, Inc., New York, NY, USA.

Chien, B.-C., Yang, J.-H., Lin, W.-Y., September 2003. Generating effective classifiers with supervised learning of genetic programming. In: Kambayashi, Y., Mohania, M. K., Wöß, W. (Eds.), Proceedings of the 5th International

Conference on Data Warehousing and Knowledge Discovery. Vol. 2737 of Lecture Notes in Computer Science. Springer, Prague, Czech Republic, pp. 192–201.

Eiben, A. E., Smit, S. K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation.

Falco, I. D., Tarantino, E., Cioppa, A. D., Fontanella, F., September-October 2004. An innovative approach to genetic programming-based clustering. In: Abraham, A., Baets, B. D., Köppen, M., Nickolay, B. (Eds.), Proceedings of the 9th Online World Conference on Soft Computing in Industrial Applications. Vol. 34 of Advances in Soft Computing. Springer, pp. 55–64.

Folino, G., Pizzuti, C., Spezzano, G., August 2008. Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification. IEEE Transactions on Evolutionary Computation 12 (4), 458–468.

Freitas, A. A., 1997. A genetic programming framework for two data mining tasks: classification and generalized rule induction. In: Proceedings of the 2nd Annual Conference on Genetic Programming. Morgan Kaufmann, pp. 96–101.

Freitas, A. A., 2002. Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag.

Gabrielov, A., Vorobjov, N., 2004. Complexity of computations with Pfaffian and Noetherian functions. In: et al., Y. I. (Ed.), Normal Forms, Bifurcations and Finiteness Problems in Differential Equations. Kluwer, pp. 211–250.

Giusti, M., Heintz, J., Morais, J., Morgentern, J., Pardo, L., February 1998. Straight line programs in geometric elimination theory. Journal of Pure and Applied Algebra 124, 121–146.

Giusti, M., Heinz, J., November 1993. La détermination des points isolés et la dimension dúne varieté agebrique peut se faire en temps polynomial. In:

Eisenbud, D., Robbiano, L. (Eds.), Computational Algebraic Geometry and Commutative Algebra, Symposia Matematica XXXIV. Cambridge University Press, pp. 216–256.

Gori, M., Maggini, M., Martinelli, E., Soda, G., 1998. Inductive inference from noisy examples using the hybrid finite state filter. IEEE Transactions on Neural Networks 9 (3), 571–575.

Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Heintz, J., Roy, M., Solerno, P., 1990. Sur la complexité du principe de tarski-seidenberg. Bulletin de la Societé Mathematique de France 118 (1), 101–126.

Hennessy, K., Madden, M. G., Conroy, J., Ryder, A. G., August 2005. An improved genetic programming technique for the classification of Raman spectra. Knowledge Based Systems 18 (4-5), 217–224.

Jie, L., Xinbo, G., Li-cheng, J., September 2003. A GA-based clustering algorithm for large data sets with mixed numeric and categorical values. In: Proceedings of the 5th International Conference on Computational Intelligence and Multimedia Applications. IEEE, Xi'an, China, pp. 102–107.

Karpinski, M., Macintyre, A., February 1997. Polynomial bounds for VC dimension of sigmoidal and general Pfaffian Neural Networks. J. Comput. Syst. Sci. 54 (1), 169–176.

Keijzer, M., Babovic, V., 2000. Genetic programming, ensemble methods and the bias/variance tradeoff–introductory investigations. In: Genetic Programming. Springer, pp. 76–90.

Koltchinskii, V., 2001. Rademacher penalties and structural risk minimization. IEEE Transactions on Information Theory 47 (5), 1902–1914.
URL http://dx.doi.org/10.1109/18.930926

Koza, J., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press.

Krawiec, K., December 2002. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. Genetic Programming and Evolvable Machines 3 (4), 329–343.

Kuo, C.-S., Hong, T.-P., Chen, C.-L., 2007. Applying genetic programming technique in classification trees. Soft Computing 11 (12), 1165–1172.

Li, G., Wang, J. F., Lee, K. H., Leung, K.-S., August 2008. Instruction-matrix-based genetic programming. IEEE Transactions on Systems, Man, and Cybernetics, Part B 38 (4), 1036–1049.

Mendelson, S., 2002. Rademacher averages and phase transitions in glivenko-cantelli classes. IEEE Transactions on Information Theory 48 (1), 251–263.

Miller, J. F., Thomson, P., 2000. Cartesian genetic programming. In: Genetic Programming. Springer, pp. 121–132.

Milnor, J. W., Weaver, D. W., 1965. Topology from the differentiable viewpoint: based on notes by David W. Weaver. University Press of Virginia.

Ni, J., Drieberg, R. H., Rockett, P., 2013. The use of an analytic quotient operator in genetic programming. IEEE Trans. Evolutionary Computation 17 (1), 146–152.

Ni, J., Rockett, P., 2015a. Tikhonov regularization as a complexity measure in multiobjective genetic programming. IEEE Trans. Evolutionary Computation 19 (2), 157–166.

Ni, J., Rockett, P., 2015b. Training genetic programming classifiers by vicinal-risk minimization. Genetic Programming and Evolvable Machines 16 (1), 3–25.

Nikolaev, N., de Menezes, L. M., Iba, H., 2002. Overfitting avoidance in genetic programming of polynomials. In: Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on. Vol. 2. IEEE, pp. 1209–1214.

Nikolaev, N., Iba, H., August 2001. Regularization approach to inductive genetic programming. IEEE Transactions on Evolutionary Computation 5 (4), 303–309.

Okley, H., 1994. Two scientific applications of genetic programming: Stack filters and nonlinear fitting to chaotic data. In: Kinnear, K. E. (Ed.), Advances in Genetic Programming. Cambridge, MA: MIT Press, pp. 369–389.

Poli, R., Cagnoni, S., 1997. Evolution of pseudo-colouring algorithms for image enhancement with interactive genetic programming. Cognitive Science research papers - Univeristy of Birmingham.

Poli, R., Langdon, W. B., McPhee, N. F., 2008. A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd.

Shaoning, P., Kasabov, N., July 2004. Inductive vs transductive inference, global vs local models: SVM, TSVM and SVMT for gene expression classification problems. In: Proceedings IEEE International Joint Conference on Neural Networks vol. 2. IEEE, IEEE, pp. 1297–1202.

Smith, M. G., Bull, L., September 2005. Genetic programming with a genetic algorithm for feature construction and selection. Genetic Programming and Evolvable Machines 6 (3), 265–281.

Tackett, W., Carmi, A., 1994. The donut problem: Scalability and generalization in genetic programming. In: Kinnear, K. E. (Ed.), Advances in Genetic Programming. Cambridge, MA: MIT Press, pp. 143–176.

Tenebaum, J., Griffiths, T., Kemp, C., 2006. Theory based Bayesian models of inductive learning and reasoning. Natural Computing 10 (7), 309–318.

Topchy, A., Punch, W., 2001. Faster genetic programming based on local gradient search of numeric leaf values. In: Proc. of Genetic and Evolutionary Computation Conference GECCO 2001. Morgan Kaufmann, pp. 155–162.

Tsakonas, A., Gabrys, B., 2012. Gradient: Grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems. Expert Systems with Applications 39 (18), 13253 – 13266.

Vanneschi, L., Mauri, G., Valsecchi, A., Cagnoni, S., July 2006. Heterogeneus cooperative coevolution: Strategies of integration between GP and GA. In: Proc. of the 8th annual conference on Genetic and Evolutionary Computation. ACM, ACM, pp. 361–368.

Vapnik, V., 1998. Statistical Learning Theory. John Willey & Sons.

Vapnik, V., Chervonenkis, A., 1974. Ordered risk minimization. Automation and Remote Control 34, 1226–1235.

Vapnik, V. N., 1995. The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA.

Wedashwara, W., Mabu, S., Obayashi, M., Kuremoto, T., 2015. Combination of genetic network programming and knapsack problem to support record clustering on distributed databases. Expert Systems with Applications, –.

Appendix

In the sequel we present the mathematical expressions of the best SLPs obtained for all three experiments and each of the three regularization methods described in this article.

Target function	Method	Mathematical expression
A random	AIC	$f(x_0) = c_1 + c_2 - k_0 - x_0$
polynomial	BIC	$f(x_0) = c_0 - c_2$
in $P_1[X]$	VCD	$f(x_0) = \log(k_1 + x_0)$
A random	AIC	$f(x_0, x_1) = \exp(x_1) + c_5$
polynomial	BIC	$f(x_0, x_1) = x_0 + x_1 + \exp(x_1) - c_1 + 2c_2$
in $P_2[X]$	VCD	$f(x_0, x_1) = \exp(c_1 x_0 - c_4 - x_1)$
A random	AIC	$f(x_0, x_1, x_2) = sign(x_2) - c_0 + c_7$
polynomial	BIC	$f(x_0, x_1, x_2) = \exp(x_2 - x_0)$
in $P_3[X]$	VCD	$f(x_0, x_1, x_2) = \exp(x_2)/\exp(c_1 x_2 - k_0)$
A random	AIC	$f(x_0, x_1, x_2, x_3) = x_2 - c_6$
polynomial	BIC	$f(x_0, x_1, x_2, x_3) = x_0 + x_2 - c_8$
in $P_4[X]$	VCD	$f(x_0, x_1, x_2, x_3) = \exp(x_0) * x_0 - x_3 + x_2$
A random	AIC	$f(x_0, x_1, x_2, x_3, x_4) = \exp(c_1 - x_0)$
polynomial	BIC	$f(x_0, x_1, x_2, x_3, x_4) = \exp(c_5 - x_0 + c_0)$
in $P_5[X]$	VCD	$f(x_0, x_1, x_2, x_3, x_4) = \exp(\exp(c_8 x_2))$

Table 12: Mathematical expressions for the best models: first experiment

Note that in this first experiment the target function is a random polynomial in $P_i[X]$ (with $1 \leq i \leq 5$) and each of the three regularization methods under study performed best for possibly distinct targets (therefore, each line of Table 12 represents a different function). Recall that in the first and second experiment, the examples are perturbed with noise, which explains why the algorithms fail to find the exact symbolic expression of the target function. In the third experiment the target function is unknown.

Target function	Method	Mathematical expression
F	AIC	$F(x,y,z) = 30x + 27y$
	BIC	$F(x,y,z) = 8x - 30y - 28z$
	VCD	$F(x,y,z) = (x+y+z)^2 + 1$
G	AIC	$G(x,y,z) = z + c_2 + 3c_3 + 2c_4$
	BIC	$G(x,y,z) = z + 3c_2 + 2c_3 + 5c_4 + c_6$
	VCD	$G(x,y,z) = (c_2 - c_6)x + z + c_2$
K	AIC	$K(x,y,z,w) = 2c_3 + 6c_4 + 7c_5$
	BIC	$K(x,y,z,w) = 4c_1 + 16c_6$
	VCD	$K(x,y,z,w) = c_5 * x$
g_1	AIC	$g_1(x) = k_0 - c_5 - k_2$
	BIC	$g_1(x) = -c_1 + c_2 + c_4$
	VCD	$g_1(x) = 2c_3 - k_1$
g_2	AIC	$g_2(x) = x - 9c_3$
	BIC	$g_2(x) = -x - c_2 + c_5$
	VCD	$g_2(x) = x + c_1 - c_4$
g_3	AIC	$g_3(x_1,x_2) = -2c_1 + 2c_2 - c_4$
	BIC	$g_3(x_1,x_2) = -c_1 + c_4 + c_5$
	VCD	$g_3(x_1,x_2) = -c_1 + 3c_5$
f_1	AIC	$f_1(x) = x^4 + 2x^3 + x^2 - 5x$
	BIC	$f_1(x) = x^4 + 2x^3 + x^2$
	VCD	$f_1(x) = -x^4 + 3x^3 - x^2 + 3x$
f_2	AIC	$f_2(x) = 12x + 16c_1$
	BIC	$f_2(x) = 2\exp(2x)$
	VCD	$f_2(x) = 2c_1 + \exp(x)/c_1/x - \sin(2x)$
f_3	AIC	$f_3(x) = 3x - 3c_1$
	BIC	$f_3(x) = -5x + 2c_1$
	VCD	$f_3(x) = 2x^2 + x$
f_4	AIC	$f_4(x) = 0$
	BIC	$f_4(x) = 0$
	VCD	$f_4(x) = 0$
f_5	AIC	$f_5(x) = k_0 - c_1$
	BIC	$f_5(x) = k_0$
	VCD	$f_5(x) = \sin(x) + \cos(k_2/x)$

Table 13: Mathematical expressions for the best models: second experiment

Target function	Method	Mathematical expression
Abalone	AIC	$f(x_1, \ldots, x_8) = \exp(\exp(c_2))$
	BIC	$f(x_1, \ldots, x_8) = \exp((c_2 + x_2)/c_2)$
	VCD	$f(x_1, \ldots, x_8) = \exp(2x_2 + sign(c_4)) + c_5 + \exp(c_6)$
Ailerons	AIC	$f(x_1, \ldots, x_{40}) = 0$
	BIC	$f(x_1, \ldots, x_{40}) = x_{24} - x_{40}$
	VCD	$f(x_1, \ldots, x_{40}) = \log(\cos(x_{20}))$
*AutoMPG*8	AIC	$f(x_1, \ldots, x_7) = x_5 + c_2 + 8c_5$
	BIC	$f(x_1, \ldots, x_7) = -x_1 - x_5 - c_2 - 5c_6$
	VCD	$f(x_1, \ldots, x_7) = x_5 + \sqrt{x_6}$

Table 14: Mathematical expressions for the best models: third experiment