Universal Consistency and Bloat in GP

Sylvain Gelly, Olivier Teytaud, Nicolas Bredeche, Marc Schoenauer

Sylvain Gelly — Olivier Teytaud — Nicolas Bredeche
Marc Schoenauer

Universal Consistency and Bloat in GP

RSTI – RIA 20/2006

Universal Consistency and Bloat in GP

Some theoretical considerations about Genetic Programming from a Statistical Learning Theory viewpoint

Sylvain Gelly — Olivier Teytaud — Nicolas Bredeche
Marc Schoenauer

Equipe TAO - INRIA Futurs, LRI, Bat. 490
University Paris-Sud, F-91405 Orsay Cedex

ABSTRACT. *In this paper, we provide an analysis of Genetic Programming (GP) from the Statistical Learning Theory viewpoint in the scope of symbolic regression. Firstly, we are interested in Universal Consistency, i.e. the fact that the solution minimizing the empirical error does converge to the best possible error when the number of examples goes to infinity, and secondly, we focus our attention on the uncontrolled growth of program length (i.e. bloat), which is a well-known problem in GP. Results show that (1) several kinds of code bloats may be identified and that (2) Universal consistency can be obtained as well as avoiding bloat under some conditions. We conclude by describing an* ad hoc *method that makes it possible simultaneously to avoid bloat and to ensure universal consistency.*

RÉSUMÉ. *Dans cet article, nous proposons une étude de la Programmation Génétique (PG) du point de vue de la théorie de l'Apprentissage Statistique dans le cadre de la régression symbolique. En particulier, nous nous sommes intéressés à la consistence universelle en PG, c'est-à-dire la convergence presque sûre vers l'erreur bayésienne à mesure que le nombre d'exemples augmente, ainsi qu'au problème bien connu en PG de la croissance incontrôlée de la taille du code (i.e. le "bloat"). Les résultats que nous avons obtenus montrent d'une part que l'on peut identifier plusieurs types de bloat et d'autre part que la consistence universelle et l'absence de bloat peuvent être obtenues sous certaines conditions. Nous proposons finalement une méthode* ad hoc *évitant justement le bloat tout en garantissant la consistence universelle.*

KEYWORDS: *genetic programming, statistical learning theory, symbolic regression, universal consistency, bloat.*

MOTS-CLÉS : *programmation génétique, théorie de l'apprentissage, régression symbolique, consistence universelle, bloat.*

DRAFT. Original in: RSTI – RIA – 20/2006. New Methods in Machine Learning, pages 805 to 827

1. Introduction

This paper is about two important issues in Genetic Programming (GP), that is Universal Consistency (UC) and code bloat. UC consists in the convergence to the optimal error rate with regards to an unknown distribution of examples. A restricted version of UC is consistency, which focus on the convergence to the optimal error rate within a restricted search space. Both UC and consistency are well studied in the field of statistical learning theory. Despite their possible benefits, they have not been widely studied in the field of GP.

Code bloat is the uncontrolled growth of program size that may occur in GP when relying on a variable length representation (Koza, 1992, Langdon, 1998). This has been identified as a key problem in GP for which there have been several empirical studies. However, very few theoretical studies addressed this issue directly.

The work presented in this paper is intended to provide some theoretical insights on the bloat phenomenon and its link with UC in the context of GP-based learning taking a statistical learning theory perspective (Vapnik, 1995). Statistical learning theory provides several theoretical tools to analyze some aspects of learning accuracy. Our main objective consists in performing both an in-depth analysis of bloat as well as providing appropriate solutions to avoid it.

The following section gives an overview of current code bloat theories, describing the results presented in this paper from a GP perspective and providing a short discussion on these results and their benefits for the GP practitioner. Section 2 and 3 formally prove all the aforementioned results about code bloat avoidance and UC and suggest a new approach ensuring both. Section 4 finally concludes this paper with a discussion on the consequences of those theoretical results for GP practitioners and uncover some perspectives of work.

1.1. *Code Bloat in Genetic Programming*

As quoted from (Banzhaf *et al.,* 1998, p. 182):

> [...] in 1994, Angeline noted that many of the evolved solutions in Koza's book contained code segments that were extraneous. By extraneous, he meant that if those code segments were removed from the solution, this would not alter the result produced by the solution. Examples of such code would be: (1) $a = a + 0$ or (2) $b = b * 1$.

While bloat is well-defined and can be identified, there are currently no consensual explanations on why it occurs. Indeed, three popular theories can be found in the literature to explain it:

– The *introns* theory states that bloat acts as a protective mechanism in order to avoid the destructive effects of operators once relevant solutions have been found (Blickle *et al.,* 1994, McPhee *et al.,* 1995, Nordin *et al.,* 1995). Introns are pieces

of code that have no influence on the fitness: either sub-programs that are never executed, or sub-programs which have no effect;

– The *fitness causes bloat* theory relies on the assumption that there is a greater probability to find a bigger program with the same behavior (i.e. semantically equivalent) than to find a shorter one. Thus, once a good solution is found, programs naturally tend to grow because of fitness pressure (Langdon *et al.,* 1997). This theory states that code bloat is operator-independent and may happen for any variable length representation-based algorithm. As a consequence, code bloat is not to be limited to population-based stochastic algorithm (such as GP), but may be extended to many algorithms using variable length representation (Langdon, 1998);

– The *removal bias* theory states that removing longer sub-programs is more dangerous to do than removing shorter ones (because of possible destructive consequence), so there is a natural bias that benefits to the preservation of longer programs (Soule, 2002).

While it is now considered that each of these theories somewhat capture part of the problem (Banzhaf *et al.,* 2002), there has not been any definitive global explanation of the bloat phenomenon. At the same time, no definitive practical solution has been proposed that would avoid the drawbacks of bloat (i.e. increasing evaluation time of large trees) while maintaining the good performances of GP on difficult problems. Some common solutions rely either on specific operators (e.g. size-fair crossover (Langdon, 2000), or different fair mutation (Langdon *et al.,* 1999)), on some parsimony-based penalization of the fitness (Soule *et al.,* 1998) or on abrupt limitation of the program size such as the one originally used by Koza (Koza, 1992). Also, some multi-objective approachs have been proposed ((Luke *et al.,* 2002, Silva *et al.,* 2003, Ekart *et al.,* 2002, De Jong *et al.,* 2001, Bleuler *et al.,* 2001)). Some other more particular solutions have been proposed but are not widely used yet (Ratle *et al.,* 2001, Zhang *et al.,* 1995).

1.2. *Structural and Functional Bloat*

Although code bloat is not clearly understood, it is yet possible to distinguish at least two kinds of code bloat. We first define *structural bloat* as the code bloat that necessarily takes place when no optimal solution can be approximated by a set of programs with bounded length. In such a situation, optimal solutions of increasing accuracy will also exhibit an increasing complexity (larger programs), as larger and larger code will be generated in order to better approximate the target function. This extreme case of structural bloat has also been demonstrated in (Gustafson *et al.,* 2004). The authors use some polynomial functions of increasing difficulty, and demonstrate that a precise fit can only be obtained through an increased bloat (see also (Daida *et al.,* 2001) for related issues about problem complexity in GP).

Another form of bloat is the *functional bloat*, which takes place when program length keeps on growing even though an optimal solution (of known complexity) does

lie in the search space. In order to clarify this point, let us use a simple symbolic regression problem defined as follow: given a set $\mathcal{S}$ of test cases, the goal is to find a function f (here, a GP-tree) that minimizes the Mean Square Error (or MSE). If we intend to approximate a polynomial (e.g. $14 * x^2$ with $x \in [0, 1]$), we may observe code bloat since it is possible to find arbitrarily long polynomials that gives the exact solution (e.g. $14x^2+0*x^3+\ldots$), or sequences of polynomials of length growing to ∞ and accuracy converging to the optimal accuracy (e.g. $P_n(x) = 14x^2 + \sum_{i=1}^{n} \frac{1}{n!i!}x^i$). Most of the works cited earlier are in fact concerned with functional bloat which is the most surprising, and the most disappointing kind of bloat.

We will consider various levels of functional bloat: cases where the length of programs found by GP runs to infinity as the number of test cases run to infinity whereas a bounded-length solution exists, and also cases where large programs are found with high probability by GP whereas a small program is optimal.

1.3. *Universal Consistency*

Another important issue is to study the convergence of the function given by GP, under some sufficient conditions, when the number of test cases goes to infinity, toward the actual function used to generate the test cases. This property is known in statistical learning as *Universal Consistency* (UC). Note that this notion is slightly different from that of universal approximation, to which people usually refer when doing symbolic regression in GP: because polynomial for instance are known to be able to approximate any continuous function, GP search using operators $\{+, *\}$ is also assumed to be able to approximate any continuous function. However, UC is concerned with the behavior of the algorithm when the number of test cases goes to infinity: the existence of a polynomial that approximates a given function at any arbitrary precision does not imply that any polynomial approximation built from a set of sample points will converge to that given function when the number of points goes to infinity.

1.4. *Results Overview*

The UC of GP is investigated in Sections 2 and 3, with a detailed study of structural and functional bloats that might occur when searching a program space with GP. A formal and detailed definition of the space of programs that we consider in this paper is given in Definition 2.1. Note that various definitions could be considered also, the main elements being: i) universal approximation, and ii) a measure of program-complexity which leads to the finiteness of the VC-dimension for a given program-complexity. Two types of results are derived: i) *UC-related results*, that is whether the probability of misclassification of GP-solutions converges to the optimal value when the number of test cases goes to infinity, and ii) *bloat-related results*, regarding first structural bloat and then functional bloat with various fitness penalization and complexity bounds.

Let us now state precisely, yet informally, the main results of this paper. Section 2 precisely defines the computing machine under examination, and proves the resulting GP search space fulfills the conditions of some standard statistical learning theorems listed in Appendix A. Applying those statistical learning theorems to GP lead to Theorem 2.4, which demonstrates the UC of GP when the fitness measure includes some complexity penalization. Proposition 2.5, a bloat-related theoretical result, unsurprisingly proves that if optimality can not be reached within finite complexity, then converging to the optimal error implies an infinite increase of bloat. And then, Theorem 2.6 proves a negative result about bloat, that is minimizing the MSE alone might lead to bloat even if an optimal function with bounded length belongs to the program search space (i.e. empirical solutions complexity goes to infinity with the sample size).

Then in Section 3, Theorems 2.7 and 3.1 show that it is possible to carefully adjust the parsimony pressure in order to obtain both UC and bounds on the empirical solution complexity (i.e. no bloat). These are the best positive results one could expect considering the previous findings. Note that, though all proofs in Section 2 are stated and proved in the context of binary classification (i.e. find a function from $\mathbb{R}^d$ into $\{0, 1\}$), their generalization to regression (i.e. find a function from $\mathbb{R}^d$ into $\mathbb{R}$) is straightforward.

2. A First Step Towards No Bloat

The following pages present formal proofs of the results detailed in previous section. Note there are intensive references to elements from statistical learning theory that are presented in the Appendix A. The reader unfamiliar with this theory is advised to read the appendix before reading the present section.

First, Definition 2.1 precisely defines the programs space under examination and Lemma 2.2 carefully shows that it satisfies the hypotheses of Theorems A.1 to A.4 of Appendix A. As stated by Theorem 2.3, this allows the evaluation of the VC-dimension (Vapnik, 1995) of sets of programs. Then, Theorem 2.4, Proposition 2.5 and Theorem 2.6 are derived from these preliminary results.

It should be noted the mildness of the hypothesis behind Lemma 2.2. We consider any programs of bounded length, working with real variables, provided that the computation time is *a priori* bounded. Usual families of programs in GP verify this hypothesis and much stronger hypothesis. For example, usual tree-based representations avoid loops and therefore all quantities that have to be bounded in lemma below (typically, number of times each operator is used) are bounded for trees of bounded depths. This is also true for direct acyclic graphs. We here deal with a very general case; much better constants can be derived for specific cases, without changing the fundamental results in the sequel of the paper. Lemma 2.2 is necessary because Theorem A.4 deals with the VC-dimension of one program with parameters, whereas we want to deal with families of programs. Lemma 2.2 provides a (tedious, but concep-

tually simple) generalization of Theorem A.4 to families of programs, by simulating a family of programs by a computing machine that is a general parametric program. Then, Theorem 2.4, Proposition 2.5 and Theorem 2.6 are derived from these preliminary results.

Definition 2.1 (Set of programs studied)**.** *Let $F(n,t,q,m,z)$ be the set of functions from $\mathbb{R}^{z-m}$ towards $\{0,1\}$ which can be computed by a program with a maximum of n lines as follows:*

– The program uses at most t operations.

– Each line contains one operation among the followings.

The operations are taken among the following set of instructions:

– Operations $\alpha \mapsto \exp(\alpha)$ (at most q times);

– Operations $+$, $-$, $\times$, and $/$;

– Jumps conditioned on $>$, $\geq$, $<$, $\leq$, and $=$;

– Output 0;

– Output 1;

– Labels for jumps;

– Constants (at most m different);

– Variables (at most z different, with $z \geq m$).

We note $F(n,t,q,m,z)$ as F for short when there is no ambiguity. The parameters n,t,q,m,z are then implicit.

Lemma 2.2 (Computing machine for the set of programs studied)**.** *We note $\log_2(x)$ the integer part (ceil) of $\log(x)/\log(2)$.*

With $F(n,t,q,m,z)$ as defined in the Definition 2.1 above, following the notations of Theorem A.4, there exists a parameterized program h such that $F(n,t,q,m,z)$ is included in $H = \{x \mapsto h(a,x); a \in \mathbb{R}^d\}$, where h is a program with at most:

– $t' = T(n,t,z) = t + t\max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + \max(9\log_2(z), 0) + \max(3\log_2(z) - 3, 0))$ instructions.

– $q' = q$ exponentials.

– $d' = 1 + m$ dimensions of parameters.

written with the same set of instructions.

Interpretation 1. *This lemma states that a family of programs as defined above is included in the parameterizations of one well-chosen program. Equivalently, we can say that for any $n \in \mathbb{N}$, $t \in \mathbb{N}$, $q \in \mathbb{N}$, $m \in \mathbb{N}$ and $z \in \mathbb{N}$, there exists some program h (constrained by t', q' and d' as stated above) such that any program in $F(n,t,q,m,z)$ is of the form $x \mapsto h(a,x)$ for some $a \in \mathbb{R}^d$. This replaces a family of programs by one parametric program (i.e. a computing machine), and that will be useful for the evaluation of the VC-dimension of a family of programs by Theorem A.4.*

Proof. In order to prove this result, we define below a program h as in theorem A.4 that can emulate any of the programs in $F(n, t, q, m, z)$, with at most $t' = T(n, t, z)$, $q' = q$, and $d' = 1 + m$. Let x be the input variable of dimension $dim(x) \leq z - m$.

The program goes as follow:

– Label INPUT;

– Initialize $variable(1)$ with value $x(1)$;

– Initialize $variable(2)$ with value $x(2)$;

...

– Initialize $variable(dim(x))$ with value $x(dim(x))$;

– Label CONSTANTS;

– Initialize $variable(dim(x) + 1)$ with value a_1;

– Initialize $variable(dim(x) + 2)$ with value a_2;

...

– Initialize $variable(dim(x) + m)$ with value a_m;

– Label DECODE-INTO-C;

– Operation decode c;

– Label BEGIN-OF-MAIN-LOOP;

– Label LINE(1);

– Operation $c(1, 1)$ with variables $c(1, 2)$, $c(1, 3)$, and $c(1, 4)$;

– Label LINE(2);

– Operation $c(2, 1)$ with variables $c(2, 2)$, $c(2, 3)$, and $c(2, 4)$;

...

– Label LINE(n);

– Operation $c(n, 1)$ with variables $c(n, 2)$, $c(n, 3)$, and $c(n, 4)$;

– Label OUTPUT(0);

– Output 0;

– Label OUTPUT(1);

– Output 1.

We need m real numbers, for parameters, and $4n$ integers $c(.,.)$ for decoding inputs. "Operation decode c" translates each input real number y (in $[0, 1]$) into a set of four integers $c(.,.)$. This operation can be developped as follows :

1) Let $y \in [0, 1]$;

2) For each $i \in [1, \ldots n]$:

- $c(i, 1) = 0$;

- $y = y * 2$;

- If $(y > 1)$ then $c(i, 1) = 1$ and $y = y - 1$;

- $y = y * 2$;
- If $(y > 1)$ then $c(i,1) = c(i,1) + 2$ and $y = y - 1$;
- $y = y * 2$;
- If $(y > 1)$ then $c(i,1) = c(i,1) + 4$ and $y = y - 1$.

3) For each $j \in [2,4]$ and $i \in [1, \dots n]$:

- $c(i,j) = 0$;
- $y = y * 2$;
- If $(y > 1)$ then $c(i,j) = 1$ and $y = y - 1$;
- $y = y * 2$;
- If $(y > 1)$ then $c(i,j) = c(i,j) + 2$ and $y = y - 1$;
- $y = y * 2$;
- If $(y > 1)$ then $c(i,j) = c(i,j) + 4$ and $y = y - 1$;

...

- $y = y * 2$;
- If $(y > 1)$ then $c(i,j) = c(i,j) + 2^{log_2(z)-1}$ and $y = y - 1$.

After decoding, $c(i,1)$ stands for the code instruction, $c(i,2)$ and $c(i,3)$ gives the memory addresses where input values can be reached, and $c(i,4)$ gives the memory address where the output value should be written.

The cost of this is $n\left(3 + \max\left(3\log_2\left(z\right), 0\right)\right)$ "if then", and $n\left(3 + \max\left(3\log_2\left(z\right), 0\right)\right)$ operators $\times$, and $n\left(2 + \max\left(3\left(\log_2\left(z\right) - 1\right), 0\right)\right)$ operators $+$, and $n\left(3 + \max\left(3\log_2\left(z\right), 0\right)\right)$ operators $-$. The overall sum is bounded by $n\left(11 + \max\left(9\log_2\left(z\right), 0\right) + \max\left(3\log_2\left(z\right) - 3, 0\right)\right)$.

The result then derives from the rewriting of "operation $c(i,1)$ with variables $c(i,2)$, $c(i,3)$, and $c(i,4)$". This operation interprets one code instruction and its parameters (as decoded before). It can be developed as follows:

– If $c(i,1) == 0$ then goto OUTPUT(1);

– If $c(i,1) == 1$ then goto OUTPUT(0);

– If $c(i,2) == 1$ then $c = variable(1)$;

– If $c(i,2) == 2$ then $c = variable(2)$;

...

– If $c(i,2) == z$ then $c = variable(z)$;

– If $c(i,1) == 7$ then goto LINE(c) (must be encoded by dichotomy with $\log_2(n)$ lines);

– If $c(i,1) == 6$ then goto EXPONENTIAL(i);

– If $c(i,3) == 1$ then $b = variable(1)$;

– If $c(i,3) == 2$ then $b = variable(2)$;

...

– If $c(i,3) == z$ then $b = variable(z)$;

– If $c(i,1) == 2$ then $a = c + b$;

– If $c(i,1) == 3$ then $a = c - b$;

– If $c(i,1) == 4$ then $a = c \times b$;

– If $c(i,1) == 5$ then $a = c/b$;

– If $c(i,4) == 1$ then $variable(1) = a$;

– If $c(i,4) == 2$ then $variable(2) = a$;

...

– If $c(i,4) == z$ then $variable(z) = a$;

– Label END-OF-INSTRUCTION(i).

For each such instruction, at the end of the program, we add the following three lines:

– Label EXPONENTIAL(i);

– $a = \exp(c)$;

– Goto END-OF-INSTRUCTION(i).

Each sequence of the form "if x=... then" (p times) can be encoded by dichotomy with $\log_2(p)$ tests "if ... then goto". Hence, the expected result. □

Theorem 2.3 (Finite VC-dimension of the computing machine)**.** *Consider q', t' and $d' \geq 0$. Let $F(n,t,q,m,z)$ be the set of programs described by Definition 2.1, where $q \leq q'$, $T(n,t,z) \leq t'$, and $1 + m \leq d'$.*

$$\begin{aligned} VCdim(F) &\leq t'^2 d' \left(d' + 19 \log_2(9d')\right) \\ &\leq (d'(q'+1))^2 + 11 d'(q'+1)(t' + \log_2(9d'(q'+1))) \end{aligned}$$

If $q = 0$ (no exponential) then $VCdim(F) \leq 4d'(t'+2)$.

Interpretation 2. *The theorem demonstrates that interesting and natural families of programs have finite VC-dimension. Effective methods can associate a VC-dimension to these families of programs.*

Proof. Just plug Lemma 2.2 in Theorem A.4. □

We now consider how to use such results in order to ensure UC. In all of this paper, $\Pr(.)$ denotes probabilities, as the traditional notation $P(.)$ is used for programs.

First, we show why simple empirical risk minimization (i.e. minimizing the error observed without taking into account programs complexity) does not ensure consistency. More precisely, for some distribution of test cases and some i.i.d. (independent identically distributed) sequence of test cases $\{(x_1, y_1), \ldots, (x_n, y_n), \ldots\}$, there exists $P_1, \ldots, P_n, \ldots$ such that

$$\forall n \in \mathbb{N}, \forall i \in \{1, 2, \ldots, n\} \;\; P_n\left(x_i\right) = y_i,$$

and however

$$\forall n \in \mathbb{N} \quad \Pr\left(P_n\left(x\right) = y\right) = 0.$$

This can be proved by considering that x is uniformly distributed in $[0, 1]$ and y is a constant equal to 1. Then, consider P_n, the program that compares its entry to $x_1, x_2, \ldots, x_n$, and outputs 1 if the entry is equal to x_j for some $j \leq n$, and otherwise outputs 0. With probability 1, this program output 0, whereas almost surely the desired output y is 1.

We therefore conclude that minimizing the empirical risk is not enough for ensuring any satisfactory form of consistency. Let's now show that structural risk minimization (i.e. taking into account a penalization for complex structures) can ensure UC and fast convergence when the solution can be written within finite complexity.

Theorem 2.4 (Universal consistency of genetic programming with structural risk minimization)**.** *Consider q_k, t_k, m_k, n_k, and z_k increasing integer sequences. Define $\mathcal{F}_k$ the set of programs with at most t_k lines executed, z_k variables, n_k lines, q_k exponentials, and m_k constants ($\mathcal{F}_k = F(n_k, t_k, q_k, m_k, z_k)$ of Definition 2.1) and $\mathcal{F} = \cup_k \mathcal{F}_k$. Then with ${q'}_k = q_k$, ${t'}_k = T(n_k, t_k, z_k)$, and ${d'}_k = 1 + m_k$, define V_k as:*

– If $\forall k\ q_k = 0$, then $V_k = 4{d'}_k({t'}_k + 2)$.

– Otherwise, $V_k = ({d'}_k({q'}_k + 1))^2 + 11{d'}_k({q'}_k + 1)({t'}_k + \log_2(9{d'}_k({q'}_k + 1)))$.

Now given s test cases, consider $P \in \mathcal{F}$ minimizing $\hat{L}(P) + \sqrt{\frac{32}{s} V(P) \log(es)}$, where $V(P) = V_k$ where k is minimal such that $P \in \mathcal{F}_k$. Then,

– The generalization error, with probability 1, converges to L^*;

– If one optimal function belongs to $\mathcal{F}_k$, then for any s and ϵ such that $V_k \log(es) \leq s\epsilon^2/512$, the generalization error with s test cases is larger than $L^* + \epsilon$ with probability at most $\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \exp(-s\epsilon^2/512)$ where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$.

Interpretation 3. *This theorem shows that genetic programming for binary classification, provided that structural risk minimization is performed (i.e. if we optimize an* ad hoc *compromise between complexity of programs and accuracy on empirical data), is universally consistent and verifies some convergence rate properties.*

Proof. Just plug Theorem A.5 in Theorem 2.3. Note that Δ is finite because we assumed that the integer sequences were increasing. □

We now prove the non-surprising fact that if it is possible to approximate the optimal function (the Bayesian classifier) without reaching it exactly, then the complexity of the program runs to infinity as soon as there is convergence of the generalization error to the optimal one.

Proposition 2.5 (Structural bloat in genetic programming)**.** *Consider $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots$, where $\mathcal{F}_V$ is a set of functions from X to $\{0,1\}$ with VC-dimension bounded by V. Consider $(V(s))_{s\in\mathbb{N}}$ a non decreasing sequence of integers and $(P_s)_{s\in\mathbb{N}}$ a sequence of functions such that $P_s \in \mathcal{F}_{V(s)}$.*

Define $L_V = \inf_{P\in\mathcal{F}_V} L(P)$ and $V(P) = \inf\{V; P \in \mathcal{F}_V\}$ and suppose that $\forall V \;\; L_V > L^*$. Then, $\left(L(P_s) \overset{s\to\infty}{\longrightarrow} L^*\right) \Longrightarrow \left(V(P_s) \overset{s\to\infty}{\longrightarrow} \infty\right)$.

Interpretation 4. *This is structural bloat: if the space of programs approximates but does not contain the optimal function and cannot approximate it within bounded size, then bloat occurs. Note that the assumption $\forall V \;\; L_V > L^*$ holds simultaneously for all V for many distributions, as we consider countable unions of families with finite VC-dimension (e.g. see (Devroye* et al., *1997, chap. 18)).*

Proof. Define $\epsilon(V) = L_V - L^*$. ϵ is non-increasing as the $\mathcal{F}_i$ are nested. Consider, as assumed in the proposition, that $L(P_s) \overset{s\to\infty}{\longrightarrow} L^*$. Consider V_0 a positive integer and let's prove that if s is large enough, then $V(P_s) \geq V_0$. There exists ϵ_0 such that $\epsilon(V_0) > \epsilon_0 > 0$. So, for s large enough, $L(P_s) \leq L^* + \epsilon_0 \Longrightarrow L_{V_s} \leq L^* + \epsilon_0 \Longrightarrow L^* + \epsilon(V_s) \leq L^* + \epsilon_0 \Longrightarrow \epsilon(V_s) \leq \epsilon_0 \Longrightarrow V_s \geq V_0$. □

We now show that, even in cases in which an optimal short program exists, the usual procedure (known as the method of Sieves; see also (Silva *et al.,* 2004)) defined below, consisting in defining a maximum VC-dimension depending upon the sample size (as usually done in practice and as recommended in Theorem A.3) and then using a moderate family of functions, leads to bloat.

Theorem 2.6 (Bloat with the method of Sieves)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k, \ldots$ be non-empty sets of functions with finite VC-dimensions $V_1, \ldots, V_k, \ldots$, and let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then given s i.i.d. test cases, consider $\hat{P} \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ in $\mathcal{F}_s$.*

From Theorem A.3, we already know that if $V_s = o(s/\log(s))$ and $V_s \to \infty$, then

$$\Pr\left(L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s, V_s, \delta)\right) \geq 1 - \delta$$

and almost surely

$$L(\hat{P}) \to \inf_{P\in\mathcal{F}} L(P).$$

We now state that if $V_s \to \infty$, and noting $V(P) = \min\{V_k; P \in \mathcal{F}_k\}$, then $\forall V_0,\ \delta_0 > 0$, $\exists \Pr$, a distribution of probability on X and Y, such that $\exists g \in \mathcal{F}_1$ such that $L(g) = L^*$, and for s sufficiently large $\Pr\left(V(\hat{P}) \leq V_0\right) \leq \delta_0$.

Interpretation 5. *The result in particular implies that for any V_0, there is a distribution of test cases such that $\exists g; V(g) = V_1$ and $L(g) = L^*$, with probability 1, $V(\hat{f}) \geq V_0$ infinitely often as s increases. This shows that bloat can occur if we use only an abrupt limit on code size, if this limit depends upon the number of test cases (*a

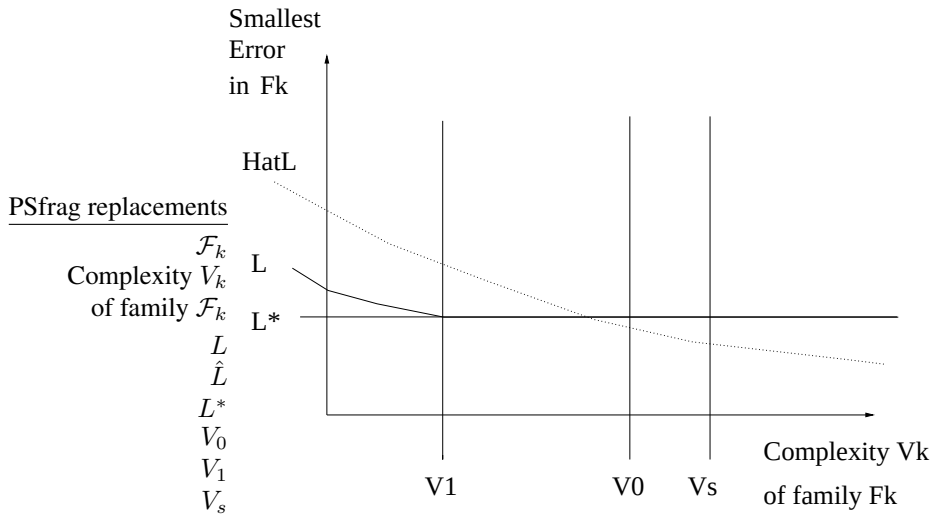

Figure 1. *Illustration of the proof of Theorem 2.6 . For V_0, δ_0 and $\Pr$ fixed, there exists V_1 such that $\inf_{f \in \mathcal{F}_1} L(f) = L^*$, that is to say there exists a simple solution (in $\mathcal{F}_1$) of error L^*. However, even if the limit V_s ensures that $L(argmin_{f \in \mathcal{F}_k, V_k \leq V_s} \hat{L})$ is close to L^*, the complexity of the solution can be arbitrarily high ($\geq V_0$ for any V_0)*

fortiori *if there's no limit). Note that this result, proved thanks to a particular distribution, could indeed be proved for the whole class of classification problems for which the conditional probability of $Y = 1$ (conditionally to X) is equal to $\frac{1}{2}$ in an open subset of the domain.*

Proof. The proof is given for the part that is not Theorem A.3. Figure 1 gives an illustration of the proof. Consider $V_0 > 0$, $\delta_0 > 0$, α such that $(e\alpha/2^\alpha)^{V_0} \leq \delta_0/2$, s such that $V_s \geq \alpha V_0$, and $d = \alpha V_0$. Consider that $x_1, \ldots, x_d$ are d points shattered by $\mathcal{F}_d$; such a family of d points exist, by definition of $\mathcal{F}_d$. Define the probability measure $\Pr$ by the fact that X and Y are independent and $\Pr(Y = 1) = \frac{1}{2}$ and $\Pr(X = x_i) = \frac{1}{d}$. Then, the following holds, with Q the empirical distribution (the average of Dirac masses on the x_i's):

1) No empty x_i: $\Pr(E_1) \to 0$, where E_1 is the fact that $\exists i; Q(X = x_i) = 0$, as $s \to \infty$;

2) No equality: $\Pr(E_2) \to 0$, where E_2 is the fact that E_1 occurs or $\exists i; Q(Y = 1 | X = x_i) = \frac{1}{2}$;

3) The best function is not in $\mathcal{F}_{V_0}$: $\Pr(E_3 | \neg E_2) \leq S(d, d/\alpha)/2^d$, where E_3 is the fact that $\exists g \in \mathcal{F}_{d/\alpha = V_0}; \hat{L}(g) = \inf_{\mathcal{F}_d} \hat{L}$, with $S(d, d/\alpha)$ the relevant shattering coefficient, that is the cardinal of $\mathcal{F}_{d/\alpha}$ restricted to $\{x_1, \ldots, x_d\}$.

It is now sufficient to rely on classical results. It is well known in the VC-theory that $S(a,b) \leq (ea/b)^b$ (see for example (Devroye *et al.,* 1997, chap.13)), hence $S(d, d/\alpha) \leq (ed/(d/\alpha))^{d/\alpha}$ and $\Pr(E_3|\neg E_2) \leq (e\alpha)^{d/\alpha}/2^d \leq \delta_0/2$. If s is sufficiently large to ensure that $\Pr(E_2) \leq \delta_0/2$ (we have proved above that $\Pr(E_2) \to 0$ as $s \to \infty$) then

$$\begin{aligned}\Pr(E_3) &\leq \Pr(E_3|\neg E_2)\Pr(\neg E_2) + \Pr(E_2) \\ &\leq \Pr(E_3|\neg E_2) + \Pr(E_2) \leq \delta_0/2 + \delta_0/2 \leq \delta_0\end{aligned}$$

This concludes the proof. □

Let's now show that it is possible to optimize a compromise between optimality and complexity in an explicit manner (e.g. by replacing 1% precision with 10 lines of programs or 10 minutes of CPU).

Theorem 2.7 (Bloat-control for regularized empirical risk minimization with relevant VC-dimension)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k, \ldots$ be non-empty sets of functions with finite VC-dimensions $V_1, \ldots, V_k, \ldots$. Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Consider W a user-defined complexity penalization term. Then, given s test cases, consider $P \in \mathcal{F}_s$ minimizing the regularized empirical risk $\hat{\tilde{L}}(P) = \hat{L}(P) + W(P)$ among F_s. If $V_s = o(s/\log(s))$ and $V_s \to \infty$, then $\tilde{L}(\hat{P}) \to \inf_{P\in\mathcal{F}} \tilde{L}(P)$ almost surely, where $\tilde{L}(P) = L(P) + W(P)$.*

Interpretation 6. *This theorem shows that, using a relevant* a priori *bound on the complexity of the program and adding a user-defined complexity penalization to the fitness, can lead to convergence toward a user-defined compromise (Zhang* et al., *1995, Zhang* et al., *1997) between classification rate and program complexity (i.e. we ensure almost sure convergence to a compromise of the form "λ_1 CPU time + λ_2 misclassification rate + λ_3 number of lines", where the λ_i are user-defined).*

Remark 1. *The drawback of this approach is that we have lost UC and consistency. This means that the misclassification rate in generalization does not converge to the Bayes error in the general case, and in spite of the fact that an optimal program exists, there is not necessarily convergence to its efficiency.*

Proof. See Figure 2 for an illustration of the proof. By Theorem A.2,

$$\sup_{P\in\mathcal{F}_s} \left|\hat{\tilde{L}}(P) - \tilde{L}(P)\right| \leq \sup_{P\in\mathcal{F}_s} \left|\hat{L}(P) - L(P)\right| \leq \epsilon(s, V_s)$$

and $\epsilon(s, V_s) \to 0$ almost surely.

Hence the expected result. □

Previous results have shown that: i) UC can be reached thanks to usual results of learning theory applied to GP (method of Sieves or structural risk minimization) but

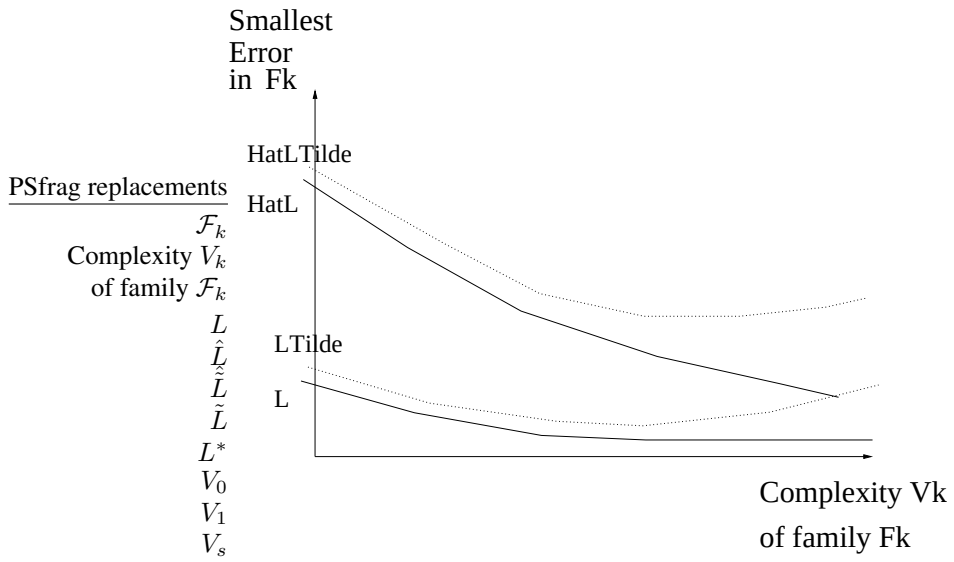

Figure 2. *Illustration of the proof of Theorem 2.7. In non-bold plots: smallest $\hat{L}$ (upper plot) or L (lower plot) in $\mathcal{F}_k$. Bold plots: smallest $\hat{\tilde{L}}$ (upper plot) or $\tilde{L}$ (loewer plot) in $\mathcal{F}_k$, i.e. error+penalization. With a larger k, $\mathcal{F}_k$ has a smaller best error $\hat{L}$, but the penalization is stronger than the gain of error by increasing the complexity*

sometimes leads to bloat, and ii) bloat can be simply avoided by a strong penalization of size, but this leads to a loss in terms of UC.

In the next section, results will make it possible to present a new approach that combines an *a priori* limit on VC-dimension (i.e. size limit) and a complexity penalization (i.e. parsimony pressure). Indeed, Theorem 3.1 will state that this leads to both UC and convergence to an optimal complexity of the program (i.e. no bloat).

3. Universal Consistency without Bloat

In this section, we consider a more complicated case where the goal is to ensure UC, while simultaneously avoiding non-necessary bloat. This means that an optimal program does exist in a given family of functions and convergence towards the minimal error rate is performed without increasing the program complexity. This is achieved by: i) merging regularization and bounding of the VC-dimension, and ii) penalization of the complexity (i.e. length) of programs by a penalty term $R(s, P) = R(s)R'(P)$ depending upon the sample size and the program. $R(.,.)$ is user-defined and the algorithm looks for a classifier with a small value of both R' and L. In the following, we study both the UC of this algorithm (i.e. $L \rightarrow L^*$)

and the no-bloat theorem (i.e. $R' \to R'(P^*)$ when P^* exists). Note that the bound $V_s = o(\log(s))$ is much stronger than the usual limit used in the method of Sieves (see Theorem 2.6).

Theorem 3.1 (No-bloat theorem)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k, \ldots$ with finite VC-dimensions $V_1, \ldots, V_k, \ldots$. Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Define $V(P) = V_k$ with $k = \inf\{t | P \in \mathcal{F}_t\}$. Define $L_V = \inf_{P \in \mathcal{F}_V} L(P)$. Consider $V_s = o(\log(s))$ and $V_s \to \infty$. Consider also that $\hat{P}_s$ minimizes $\hat{\tilde{L}}(P) = \hat{L}(P) + R(s, P)$ in $\mathcal{F}_s$, and assume that $R(s, .) \geq 0$. Then consistency is attained as whenever $\sup_{P \in \mathcal{F}_{V_s}} R(s, P) = o(1)$, we have that $L(\hat{P}_s) \to \inf_{P \in \mathcal{F}} L(P)$ almost surely. Note that for well chosen family of functions, $\inf_{P \in \mathcal{F}} L(P) = L^*$. Moreover, assume that $\exists P^* \in \mathcal{F}_{V^*} \;\; L(P^*) = L^*$. With $R(s, P) = R(s) R'(P)$ and with $R'(s) = \sup_{P \in \mathcal{F}_{V_s}} R'(P)$, we get the following results:*

*1) **Non-asymptotic no-bloat theorem:** For any $\delta \in]0, 1]$, $R'(\hat{P}_s) \leq R'(P^*) + (1/R(s)) 2\epsilon(s, V_s, \delta)$ with probability at least $1 - \delta$. This result is in particular interesting for $\epsilon(s, V_s, \delta)/R(s) \to 0$, which is possible for usual regularization terms as in Theorem A.5 of the Appendix;*

*2) **Almost-sure no-bloat theorem:** If for some $\alpha > 0$, $R(s) s^{(1-\alpha)/2} = O(1)$, then almost surely $R'(\hat{P}_s) \to R'(P^*)$ and if $R'(P)$ has discrete values (such as the number of instructions in P or many complexity measures for programs) then for s sufficiently large, $R'(\hat{P}_s) = R'(P^*)$;*

*3) **Convergence rate:** For any $\delta \in]0, 1]$, With probability at least $1 - \delta$,*

$$L\left(\hat{P}_s\right) \leq \inf_{P \in \mathcal{F}_{V_s}} L(P) + \underbrace{R(s) R'(s)}_{=o(1) \text{ by hypothesis}} + 2\epsilon(s, V_s, \delta),$$

where

$$\epsilon(s, V, \delta) = \sqrt{\frac{4 - \log(\delta / (4 s^{2V}))}{2s - 4}}$$

is such that with probability at least $1 - \delta$, $\epsilon(s, V, \delta) \geq \epsilon(s, V)$ where

$$\epsilon(s, V) = \sup_{f \in \mathcal{F}_V} |\hat{L}(f) - L(f)|,$$

given by Theorem A.1.

Interpretation 7. *Combining a code limitation and a penalization leads to UC without bloat.*

Remark 2. *The usual $R(s, P)$ as used in Theorem A.5 or Theorem 2.4 provides consistency and non-asymptotic no-bloat, when this penalization term is used in conjunction with a limitation depending on the sample size s ($P \in \mathcal{F}_s$ with s much more restrictive than in the method of Sieves). A stronger regularization leads to the same results, plus almost sure no-bloat. The asymptotic convergence rate depends upon the regularization. The result is not limited to GP and could be used*

in other areas. The strongest limitation to this results is not the GP-framework, but the fact that, as shown in Proposition 2.5, the no-bloat results require the fact that $\exists V^* \exists P^* \in \mathcal{F}_{V^*} \ L(P^*) = L^*$.

Proof. Let's define $\epsilon(s, V) = \sup_{f \in \mathcal{F}_V} |\hat{L}(f) - L(f)|$. For any P, $\hat{L}(\hat{P}_s) + R(s, \hat{P}_s) \leq \hat{L}(P) + R(s, P)$. On the other hand, $L(\hat{P}_s) \leq \hat{L}(\hat{P}_s) + \epsilon(s, V_s)$. So consistency is proved by the following:

$$\begin{aligned} L(\hat{P}_s) &\leq \left(\inf_{P \in \mathcal{F}_{V_s}} (\hat{L}(P) + R(s, P)) \right) - R(s, \hat{P}_s) + \epsilon(s, V_s), \\ &\leq \left(\inf_{P \in \mathcal{F}_{V_s}} (L(P) + \epsilon(s, V_s) + R(s, P)) \right) - R(s, \hat{P}_s) + \epsilon(s, V_s), \\ &\leq \left(\inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s, P)) \right) + 2\epsilon(s, V_s). \end{aligned}$$

As $\epsilon(s, V_s) \to 0$ almost surely (see Theorem A.2) and $\inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s, P)) \to \inf_{P \in \mathcal{F}} L(P)$, we conclude that $L(\hat{P}_s) \to \inf_{P \in \mathcal{F}} L(P)$ almost surely.

Let's now focus on the proof of the no-bloat result. By definition of the algorithm, for s sufficiently large to ensure $P^* \in \mathcal{F}_{V_s}$, $\hat{L}(\hat{P}_s) + R(s, \hat{P}_s) \leq \hat{L}(P^*) + R(s, P^*)$, hence with probability at least $1 - \delta$,

$$\begin{aligned} R'(\hat{P}_s) &\leq R'(P^*) + (1/R(s))(L^* + \epsilon(s, V_s, \delta) - L(\hat{P}_s) + \epsilon(s, V_s, \delta)), \\ &\leq R'(V^*) + (1/R(s))(L^* - L(\hat{P}_s) + 2\epsilon(s, V_s, \delta)). \end{aligned}$$

As $L^* \leq L(\hat{P}_s)$, this leads to the non-asymptotic no-bloat version of the theorem.

The almost-sure no-bloat theorem is derived as follows.

$$\begin{aligned} R'(\hat{P}_s) &\leq R'(P^*) + 1/R(s)(L^* + \epsilon(s, V_s) - L(\hat{P}_s) + \epsilon(s, V_s)), \\ &\leq R'(P^*) + 1/R(s)(L^* - L(\hat{P}_s) + 2\epsilon(s, V_s)), \\ &\leq R'(P^*) + 1/R(s)2\epsilon(s, V_s). \end{aligned}$$

All we need is the fact that $\epsilon(s, V_s)/R(s) \to 0$ almost surely.

For any $\epsilon > 0$, we consider the probability of $\epsilon(s, V_s)/R(s) > \epsilon$, and we sum over $s > 0$. By the Borel-Cantelli lemma[1], the finiteness of this sum is sufficient for the almost sure convergence to 0. The probability of $\epsilon(s, V_s)/R(s) > \epsilon$ is the probability of $\epsilon(s, V_s) > \epsilon R(s)$. By Theorem A.1, this is bounded above by $O(\exp(2V_s \log(s) - 2s\epsilon^2 R(s)^2))$. This has finite sum for $R(s) = \Omega(s^{-(1-\alpha)/2})$.

1. If $\sum_n \Pr(X_n > \epsilon)$ is finite for any $\epsilon > 0$ and $X_n > 0$, then $X_n \to 0$ almost surely.

Let us now consider the convergence rate. Consider s sufficiently large to ensure $L_{V_s} = L^*$. As shown above during the proof of the consistency,

$$\begin{aligned} L(\hat{P}_s) &\leq \inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s,P)) + 2\epsilon(s, V_s), \\ &\leq \inf_{P \in \mathcal{F}_{V_s}} (L(P) + R(s)R'(P)) + 2\epsilon(s, V_s), \\ &\leq \inf_{P \in \mathcal{F}_{V_s}} (L(P)) + R(s)R'(s) + 2\epsilon(s, V_s). \end{aligned}$$

So with probability at least $1 - \delta$,

$$L(\hat{P}_s) \leq \inf_{P \in \mathcal{F}_{V_s}} L(P) + R(s)R'(s) + 2\epsilon(s, V_s, \delta).$$

□

4. Conclusion

In this paper, we have proposed a theoretical study of two important issues in Genetic Programming (GP) known as Universal Consistency (UC) and code bloat. We have shown that the understanding of the bloat phenomenon in GP could benefit from classical results from statistical learning theory.

The first limit of our work is the fact that all these results consider that GP finds a program which is empirically the best, in the sense that given a set of test cases and a fitness function based on the empirical error (and possibly including some parsimony penalization), it will be assumed that GP does find one program in that search space that minimizes this fitness – and it is the behavior of this ideal solution, which is a random function of the number of test cases, that is theoretically studied.

Of course, we all know that GP is not such an ideal search procedure, and hence such results might look rather far away from GP practice, where the user desperately tries to find a program that gives a reasonably low empirical approximation error. Nevertheless, UC is vital for the practitioner too: indeed, it would be totally pointless to fight to approximate an empirically optimal function without any guarantee that this empirical optimum is anywhere close to the ideal optimal solution we are in fact looking for. Furthermore, the bloat-related results give some useful hints about the type of parsimony that has a chance to efficiently fight the unwanted bloat, while maintaining the UC property.

Application of theorems from learning theory has led to two original outcomes with both positive and negative results. Firstly, results on UC of GP: there is almost sure asymptotic convergence to the optimal error rate in the context of binary classification with GP. Secondly, results on code bloat: i) if the ideal target function does not have a finite description then code bloat is unavoidable (structural bloat), and ii)

code bloat can be avoided by simultaneously bounding the length of the programs with some *ad hoc* limit and using some parsimony pressure in the fitness function (functional bloat). An important point is that all methods leading to no-bloat use a regularization term.

Interestingly, all those results (both positive and negative) about code bloat are also valid in different contexts, such as for instance that of Neural Networks (the number of neurons replaces the complexity of GP programs). Moreover, results presented here are not limited to the scope of binary classification problems, and may be applied to variable length representation algorithms in different contexts such as control or identification tasks.

Finally, going back to the debate about the causes of bloat in practice, it is clear that our results can only partly explain the actual cause of bloat in a real GP run – and tend to give arguments to the "fitness causes bloat" explanation (Langdon *et al.,* 1997). It might be possible to study the impact of size-preserving mechanisms (e.g. specific variation operators, like size-fair crossover (Langdon, 2000) or fair mutations (Langdon *et al.,* 1999)) as somehow contributing to the regularization term in our final result ensuring both UC and no bloat.

Acknowledgements

This work was supported in part by the PASCAL Network of Excellence. We thank C. Gagné, anonymous reviewers and P. Ezequel for their help in improving the quality of the final paper. We thank Bill Langdon for very helpful comments.

5. References

Antony M., Bartlett P., *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, 1999.

Banzhaf W., Langdon W. B., « Some Considerations on the Reason for Bloat. », *Genetic Programming and Evolvable Machines*, vol. 3, n° 1, p. 81-91, 2002.

Banzhaf W., Nordin P., Keller R., Francone F., *Genetic Programming : an introduction*, Morgan Kaufmann Publisher Inc., San Francisco, CA, USA, 1998.

Bleuler S., Brack M., Thiele L., Zitzler E., « Multiobjective Genetic Programming: Reducing Bloat Using SPEA2 », *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, p. 536-543, 27-30, 2001.

Blickle T., Thiele L., « Genetic Programming and Redundancy », *in* , J. Hopf (ed.), *Genetic Algorithms Workshop at KI-94*, Max-Planck-Institut für Informatik, p. 33-38, 1994.

Daida J. M., Bertram R. R., Stanhope S. A., Khoo J. C., Chaudhary S. A., Chaudhri O. A., Polito J. A. I., « What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming », *Genetic Programming and Evolvable Machines*, vol. 2, n° 2, p. 165 - 191, 2001.

De Jong E. D., Watson R. A., Pollack J. B., « Reducing Bloat and Promoting Diversity using Multi-Objective Methods », *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, Morgan Kaufmann Publishers, San Francisco, CA, p. 11-18, 2001.

Devroye L., Györfi L., Lugosi G., *A Probabilistic Theory of Pattern Recognition*, Springer, 1997.

Ekart A., Nemeth S., « Maintaining the Diversity of Genetic Programs », *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, Springer-Verlag, London, UK, p. 162-171, 2002.

Grenander U., *Abstract Inference*, Wiley, New York, 1981.

Gustafson S., Ekart A., Burke E., Kendall G., « Problem difficulty and code growth in genetic programming », *Genetic Programming and Evolvable Machines*, vol. 4, n° 3, p. 271-290, 2004.

Koza J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

Langdon W. B., « The Evolution of Size in Variable Length Representations », *IEEE International Congress on Evolutionary Computations (ICEC 1998)*, IEEE Press, p. 633-638, 1998.

Langdon W. B., « Size fair and homologous tree genetic programming crossovers », *Genetic Programming And Evolvable Machines*, vol. 1, n° 1/2, p. 95-119, 2000.

Langdon W. B., Poli R., « Fitness Causes Bloat: Mutation », *Late Breaking Papers at GP'97*, Stanford Bookstore, p. 132-140, 1997.

Langdon W. B., Soule T., Poli R., Foster J. A., « The Evolution of Size and Shape », *Advances in Genetic Programming III*, MIT Press, p. 163-190, 1999.

Luke S., Panait L., « Lexicographic Parsimony Pressure », *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers, p. 829-836, 2002.

McPhee N. F., Miller J. D., « Accurate Replication in Genetic Programming », *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann, Pittsburgh, PA, USA, p. 303-309, 1995.

Nordin P., Banzhaf W., « Complexity Compression and Evolution », *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann, Pittsburgh, PA, USA, p. 310-317, 15-19, 1995.

Ratle A., Sebag. M., « Avoiding the bloat with Probabilistic Grammar-guided Genetic Programming », *Artificial Evolution VI*, Springer Verlag, 2001.

Silva S., Almeida J., « Dynamic Maximum Tree Depth : A Simple Technique for Avoiding Bloat in Tree-Based GP », *Genetic and Evolutionary Computation – GECCO-2003*, LNCS, Springer-Verlag, p. 1776-1787, 2003.

Silva S., Costa E., « Dynamic Limits for Bloat Control: Variations on Size and Depth. », *GECCO (2)*, p. 666-677, 2004.

Soule T., « Exons and Code Growth in Genetic Programming », *European Conference on Genetic Programming (EuroGP 2002)*, vol. 2278 of *LNCS*, Springer-Verlag, p. 142-151, 2002.

Soule T., Foster J. A., « Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming », *Evolutionary Computation*, vol. 6, n° 4, p. 293-309, 1998.

Vapnik V., *The Nature of Statistical Learning Theory*, Springer, 1995.

Zhang B.-T., Mühlenbein H., « Balancing Accuracy and Parsimony in Genetic Programming », *Evolutionary Computation*, 1995.

Zhang B.-T., Ohm P., Mühlenbein H., « Evolutionary Induction of Sparse Neural Trees », *Evolutionary Computation*, vol. 5, n° 2, p. 213-236, 1997.

A. Elements of Statistical Learning Theory

In the frameworks of regression and classification, statistical learning theory (Vapnik, 1995) is concerned with giving some bounds on the generalization error (i.e. the error on yet unseen data points) in terms of the actual empirical error and some fixed quantity depending only on the search space. More precisely, we will use here the notion of *Vapnik-Chervonenkis dimension* (in short, VC-dimension or $VCdim$) of a space of functions. Roughly, the VC-dimension is an indicator of the theoretical capability of a learning machine to discriminate data. It is often used to in the computation of bounds on the difference between the empirical error and the generalization error.

Consider a set of s examples $(x_i, y_i)_{i\in\{1,\ldots,s\}}$. These examples are drawn from a distribution P on the couple (X, Y). They are independent identically distributed, $Y = \{0, 1\}$ (classification problem), and typically $X = \mathbb{R}^d$ for some dimension d. For any function f, define the *loss* $L(f)$ *to be the expectation of* $|f(X) - Y|$. *Similarly, define the* empirical loss $\hat{L}(f)$ as the loss observed on the examples: $\hat{L}(f) = \frac{1}{s}\sum_i |f(x_i) - y_i|$. Finally, define L^*, the *Bayes error*, as the smallest possible generalization error for any mapping from X to $\{0, 1\}$.

The following four theorems are well-known in the statistical learning community:

Theorem A.1 (Bound on the empirical risk with finite VC-dimension)**.** *Consider $\mathcal{F}$ a family of functions from a domain X to $\{0, 1\}$ and V its VC-dimension. Then, for any $\epsilon > 0$,*

$$\Pr\left(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon\right) \quad \leq \quad 4\exp(4\epsilon + 4\epsilon^2)s^{2V}\exp(-2s\epsilon^2),$$

and for any $\delta \in]0, 1]$,

$$\Pr\left(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon(s, V, \delta)\right) \quad \leq \quad \delta,$$

where

$$\epsilon(s, V, \delta) = \sqrt{\frac{4 - \log(\delta/(4s^{2V}))}{2s - 4}}.$$

Interpretation 8. *This theorem states that in a family of finite VC-dimension, the empirical errors and the generalization errors are probably closely related.*

Proof. See (Devroye *et al.,* 1997, Th. 12.8, p. 206). □

Other forms of this theorem have no $\log(n)$ factor; they are known as Alexander's bound, but the constant is so large that this result is not better than the result above unless s is huge (see (Devroye *et al.,* 1997, p. 207)). If $s \geq 64/\epsilon^2$,

$$\Pr\left(\sup_{P\in\mathcal{F}} |L\left(P\right) - \hat{L}\left(P\right)| \geq \epsilon\right) \leq 16\left(\sqrt{s}\epsilon\right)^{4096V} \exp\left(-2s\epsilon^2\right)$$

We classically derive the following result from Theorem A.1:

Theorem A.2 (Convergence of the empirical error to the generalization error with infinite number of examples)**.** *For $s \geq 0$, consider $\mathcal{F}_s$ a family of functions from a domain X to $\{0,1\}$ and V_s its VC-dimension. Then,*

$$\sup_{P\in\mathcal{F}_s} |L\left(P\right) - \hat{L}\left(P\right)| \to 0 \text{ as } s \to \infty$$

almost surely provided that $V_s = o(s/\log(s))$ (i.e. $V_s \log(s)/s \to 0$).

Interpretation 9. *The maximal difference between the empirical error and the generalization error goes almost surely to 0 if the VC-dimension is finite.*

Proof. We use the classical Borel-Cantelli lemma, for any $\epsilon \in [0,1]$:

$$\begin{aligned}
&\sum_{s\geq 64/\epsilon^2} \Pr(|L(P) - \hat{L}(P)| > \epsilon) \\
&\leq 16 \sum_{s\geq 64/\epsilon^2} (\sqrt{s}\epsilon)^{4096V_s} \exp(-2s\epsilon^2), \\
&= 16 \sum_{s\geq 64/\epsilon^2} \exp(4096V_s(\log(\sqrt{s}) + \log(\epsilon)) - 2s\epsilon^2),
\end{aligned}$$

which is finite as soon as $V_s = o(s/\log(s))$. □

Theorem A.3 (Universal consistency with finite VC-dimension)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k, \ldots$ families of functions with finite VC-dimensions, $VCdim(\mathcal{F}_i) = V_i$, and let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then, given s examples, consider $\hat{P}_s \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ among F_s. Then, if $V_s = o(s/log(s))$ and $V_s \to \infty$, for any $\delta \in]0,1]$,*

$$\begin{aligned}
\Pr\left(L(\hat{P}_s) \leq \hat{L}(\hat{P}_s) + \epsilon(s, V_s, \delta)\right) &\geq 1 - \delta, \\
\Pr\left(L(\hat{P}_s) \leq \inf_{P\in\mathcal{F}_s} L(P) + 2\epsilon(s, V_s, \delta)\right) &\geq 1 - \delta.
\end{aligned}$$

Also, $L(\hat{P}_s) \rightarrow \inf_{P \in \mathcal{F}} L(P)$ almost surely. Note that for a well chosen family of functions (typically programs), $\inf_{P \in \mathcal{F}} L(P) = L^*$ for any distribution. Thus, this theorem leads to universal consistency (i.e. $\forall P \quad L(\hat{P}_s) \rightarrow L^*$), for a well-chosen family of functions.

Interpretation 10. *If the VC-dimension increases slowly enough as a function of the number of examples, then the generalization error goes to the optimal one. If the family of functions is well-chosen, this slow increase of VC-dimension leads to universal consistency.*

Proof. See (Devroye *et al.,* 1997, Th. 18.2, p. 290) and (Grenander, 1981). □

In the following theorem, we use d', t', q' instead of d, t, q for the sake of consistency of notations as in other results we need d, t and q but we will apply Theorem A.4 with some *ad hoc* d', t' and q'.

Theorem A.4 (Bound on VC-dimension for a computing machine)**.** *Let $H_{h,d'} = \{x \mapsto h(a,x); a \in R^{d'}\}$ where h can be computed with at most t' operations among $\alpha \mapsto \exp(\alpha)$; $+$, $-$, $\times$, $/$; jumps conditioned on $>$, $\geq$, $<$, $\leq$, $=$; output 0; output 1 (same set of instructions as in other parts of this paper). We note $H_{h,d'}$ as H when there is no ambiguity.*

Then: $VCdim(H) \leq t'^2 d'(d' + 19 \log_2(9d'))$.

Furthermore, if $\exp(.)$ is used at most q' times, and if there are at most t' operations executed among arithmetic operators, conditional jumps, exponentials, then:

$$\pi(H,m) \leq 2^{\left(d'\left(q'+1\right)\right)2/2} \left(9d'\left(q'+1\right)2^t\right)^{5d'\left(q'+1\right)} \left(em\left(2^{t'}-2\right)/d'\right)^{d'}$$

where $\pi(H,m)$ is the m^{th} shattering coefficient of H, and hence

$$VCdim\left(H\right) \leq \left(d'\left(q'+1\right)\right)^2 + 11d'\left(q'+1\right)\left(t' + \log_2\left(9d'\left(q'+1\right)\right)\right)$$

Finally, if $q' = 0$ then $VCdim(H) \leq 4d'(t'+2)$.

Interpretation 11. *The VC-dimension of the set of the possible parameterizations of a program as defined above is bounded.*

Proof. See (Antony *et al.,* 1999, 8.14 and 8.4). □

Theorem A.5 (Structural risk minimization)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ with finite VC-dimensions $VCdim(\mathcal{F}_i) = V_i$. Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Assume that all distributions lead to $L_{\mathcal{F}} = L^*$ where L^* is the optimal possible error (spaces of functions ensuring this exist). Given s examples, consider $f \in \mathcal{F}$ minimizing $\hat{L}(f) + \sqrt{\frac{32}{s} V(f) \log(es)}$, where $V(f)$ is V_k with k minimal such that $f \in \mathcal{F}_k$. Then:*

– If additionally one optimal function belongs to $\mathcal{F}_k$, then for any s and ϵ such that $V_k \log(es) \leq s\epsilon^2/512$, the generalization error is larger than ϵ with probability at most $\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \exp(-s\epsilon^2/512)$ where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is assumed finite;

– The generalization error, with probability 1*, converges to L^*.*

Interpretation 12. *The optimization of a compromise between empirical accuracy and regularization leads to the same properties as in Theorem A.3, plus a stronger convergence rate property.*

Proof. See (Vapnik, 1995) and (Devroye *et al.,* 1997, p. 294). □