Apprentissage statistique et programmation génétique: la croissance du code est-elle inévitable ?

Sylvain Gelly, Olivier Teytaud, Nicolas Bredeche, Marc Schoenauer

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

Abstract. N. Bredeche, S. Gelly, M. Schoenauer, O. Teytaud. A Statistical Learning Approach to bloat and universal consistency in genetic programming. Poster of Gecco 2005.
S. Gelly, O. Teytaud, N. Bredeche, M. Schoenauer. Apprentissage statistique et programmation genetique : la croissance du code est-elle inevitable ? pp163-178. Proceedings of CAP'2005.
Universal Consistency, the convergence to the minimum possible error rate in learning through genetic programming (GP), and Code bloat, the excessive increase of code size, are important issues in GP. This paper proposes a theoretical analysis of universal consistency and code bloat in the framework of symbolic regression in GP, from the viewpoint of Statistical Learning Theory, a well grounded mathematical toolbox for Machine Learning. Two kinds of bloat must be distinguished in that context, depending whether the target function has finite description length or not. Then, the Vapnik-Chervonenkis dimension of programs is computed, and we prove that a parsimonious fitness ensures 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). However, it is proved that the standard method consisting in choosing a maximal program size depending on the number of examples might still result in programs of infinitely increasing size with their accuracy; a fitness biased by parsimony pressure is proposed. This fitness avoids unnecessary bloat while nevertheless preserving the Universal Consistency.

1 Introduction

Universal Consistency denotes the convergence of the error rate, in expectation on the unknown distribution of examples, to the optimal one. Despite it's a fundamental element of learning, it has not been widely studied yet in Genetic Programming (GP). Its restricted version, consistency, i.e. convergence to the optimum when the optimum lies in the search space, has not been more studied. Code bloat (or code growth) denotes the growth of program size during the course of Genetic Programming runs. It has been identified as a key problem in GP from the very beginning [7], and to any variable length representations based learning algorithm [8]. It is today a well studied phenomenon, and empirical solutions have been proposed to address the issues of code bloat (see section 2). However, very few theoretical studies have addressed the issue of bloat. The purpose of this paper is to provide some theoretical insights into the bloat

phenomenon and its link with universal consistency, in the context of symbolic regression by GP, from the Statistical Learning Theory viewpoint [19]. Statistical Learning Theory is a recent, yet mature, area of Machine Learning that provides efficient theoretical tools to analyse aspects of learning accuracy and algorithm complexity. Our goal is both to perform an in-depth analysis of bloat and to provide appropriate solutions to avoid it. The paper is organized as follows : in the section below, we briefly survey some explanations for code bloat that have been proposed in the literature, and provide an informal description of our results from a GP perspective before discussing their interest for the GP practitioner. Section 2 gives a brief overview of the basic results of Learning Theory that will be used in Section 3 to formally prove all the advertised results. Finally, section 5 discusses the consequences of those theoretical results for GP practitioners and gives some perspectives about this work.

The several theories that intend to explain code bloat are :
- the *introns* theory states that code bloat acts as a protective mechanism in order to avoid the destructive effects of operators once relevant solutions have been found [14, 13, 3]. 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 tends to grow because of fitness pressure [10]. 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 [8];
- the *removal bias* theory states that removing longer sub-programs is more tacky than removing shorter ones (because of possible destructive consequence), so there is a natural bias that benefits to the preservation of longer programs [17].

While it is now considered that each of these theories somewhat captures part of the problem [2], 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 (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 [9], or different Fair Mutation [11]), on some parsimony-based penalization of the fitness [18] or on abrupt limitation of the program size such as the one originally used by Koza [7]. Some other more particular solutions have been proposed but are not widely used yet [15, 16, 12].

In this paper, we prove, under some sufficient conditions, that the solution given by GP actually converges, when the number of examples goes to infinity, toward the actual function used to generate the examples. This property is known in Statistical Learning as **Universal Consistency**. Note that this notion is a slightly different from that of Universal Approximation, that people usually refer to 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, Universal Consistency is concerned

with the behavior of the algorithm when the number of examples goes to infinity: being able to find a polynomial that approximates a given function at any arbitrary precision does not imply that any interpolation polynomial built from an arbitrary set of sample points will converge to that given function when the number of points goes to infinity.

But going back to bloat, and sticking to the polynomial example, it is also clear that the degree of the interpolation polynomial of a set of examples increases linearly with the number of examples. This leads us to start our bloat analysis by defining two kinds of bloat. On the one hand, we define the **structural bloat** as the code bloat that unavoidably takes place when no optimal solution (i.e. no function that exactly matches all possible examples) is approximated by the search space. In such a situation, optimal solutions of increasing accuracy will also exhibit an increasing complexity, as larger and larger code will be generated in order to better approximate the target function. The extreme case of structural bloat has also been demonstrated in [6]. 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 [4] for related issues about problem complexity in GP). On the other hand, we define the **functional bloat** as the bloat that takes place when programs 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 *examples*, the goal is to find a function f (here, a GP-tree) that minimized the Least Square Error (or LSE). If we intend to approximate a polynomial (ex. : $14 * x^2$), we may observe code bloat since it is possible to find arbitrarily long polynomials that gives the exact solution (ex. : $14 * x^2 + 0 * x^3 + ...$). Most of the works cited in section 1 are in fact concerned with functional bloat which is the simplest, yet already problematic, kind of bloat.

Overview of results. In section 3, we shall investigate the Universal Consistency of Genetic Programming, and study in detail structural and functional bloat that might take place when searching program spaces using GP.
A formal and detailed definition of the program space in GP is given in Lemma 1, section 3, and two types of results will then be derived: i) *Universal Consistency* results, i.e. does the probability of misclassification of the solution given by GP converges to the optimal probability of misclassification when the number of examples goes to infinity? ii) *Bloat-related results*, first regarding structural bloat, and second with respect to functional bloat in front of various types of fitness penalization and/or bounds on the complexity of the programs.

Let us now state precisely, yet informally, our main results. First, as already mentioned, we will precisely define the set of programs under examination, and prove that such a search space fulfills the conditions of the standard theorems of Statistical Learning Theory listed in Section 2. Second, applying those theorems will immediately lead to a first Universal Consistency result for GP, provided that some penalization for complexity is added to the fitness (Theorem 3). Third: the first bloat-related result, Proposition 4, unsurprisingly proves that if no optimal function belongs to the search space, then converging to the optimal error implies an infinite increase of bloat. Fourth, theorem 5 is also a negative result about bloat, as it proves that even if the optimal function belongs to the search space, minimizing the LSE alone might lead to bloat (i.e. the com-

plexity of the empirical solutions goes to infinity with the sample size). Finally, the last two theorems (5' and 6) are the best positive results one could expect considering the previous findings: it is possible to carefully adjust the parsimony pressure so as to obtain both Universal Consistency and bounds on the complexity of the empirical solution (i.e. no bloat). Section 4 discuss some properties of alternate solutions for complexity penalization : cross-validation or hold out, with various pairing of data sets.

Note that, though all proofs in Section 3 will be stated and proved in the context of 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.

Discussion 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 examples and a fitness function based on the Least Square 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 examples, 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, Universal Consistency 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 Universal Consistency property – though some actual experiments will have to be run to confirm the usefulness of those theoretical hints.

2 Elements of Learning theory

In the frameworks of regression and classification, Statistical Learning Theory [19] 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 (the LSE error above) and some fixed quantity depending only on the search space. More precisely, we will use here the notion of *Vapnik-Chervonenkis dimension* (in short, VCdim) of a space of functions. Roughly, VC-dim provides 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 4 theorems are well-known in the Statistical Learning community:

Theorem A [5, Th. 12.8, p206] : Consider $\mathcal{F}$ a family of functions from a domain X to $\{0,1\}$ and V its VC-dimension. Then, for any $\epsilon > 0$

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

$$\text{and for any } \delta \in]0,1] P(\sup_{P\in\mathcal{F}} |L(P) - \hat{L}(P)| \geq \epsilon(s,V,\delta)) \leq \delta$$

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

Interpretation : In a family of finite VC-dimension, the empirical errors and the generalization errors are probably closely related.

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 ([5, p207]): if $s \geq 64/\epsilon^2$,

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

We classically derive the following result from theorem A:

Theorem A' : Consider $\mathcal{F}_s$ for $s \geq 0$ a family of functions from a domain X to $\{0,1\}$ and V_s its VC-dimension. Then,

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

almost surely whenever $V_s = o(s/\log(s))$.

Interpretation : 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 Borell-Cantelli lemma[1], for any $\epsilon \in [0,1]$:

$$\sum_{s\geq 64/\epsilon^2} P(|L(P) - \hat{L}(P)| > \epsilon) \leq 16 \sum_{s\geq 64/\epsilon^2} (\sqrt{s}\epsilon)^{4096V_s}\exp(-2s\epsilon^2)$$

$$\leq 16 \sum_{s\geq 64/\epsilon^2} \exp(4096V_s(\log(\sqrt{s}) + \log(\epsilon)) - 2s\epsilon^2)$$

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

Theorem B in [5, Th. 18.2, p290] : Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then, being given s examples, consider $\hat{P} \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ among F_s. Then, if $V_s = o(s/log(s))$ and $V_s \to \infty$,

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

$$P(L(\hat{P}) \leq \inf_{P\in\mathcal{F}_s} L(P) + 2\epsilon(s,V_s,\delta)) \geq 1-\delta$$

$$\text{and } L(\hat{P}) \to \inf_{P\in\mathcal{F}} L(P) \text{ a.s.}$$

Note that for a well chosen family of functions (typically, programs), $\inf_{P\in\mathcal{F}} L(P) = L^*$ for any distribution ; so, theorem B leads to universal consistency (i.e. $\forall P; L(\hat{P}) \to L^*$), for a well-chosen family of functions.

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

Interpretation : 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.

In the following theorem, we use d', t', q' instead of d, t, q for the sake of notations in a corollary below.

Theorem C (8.14 and 8.4 in [1]) : Let $H = \{x \mapsto h(a,x); a \in R^{d'}\}$ 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. 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,

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

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

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

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

Interpretation : The VC-dimension of the set of the possible parametrizations of a program as defined above is bounded.

Theorem D : structural risk minimization, [19], [5] p. 294. Let $\mathcal{F}_1$, ..., $\mathcal{F}_k$ ...with finite VC-dimensions V_1, ..., V_k, ...Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Assume that all distribution lead to $L_{\mathcal{F}} = L^*$ where L^* is the optimal possible error (spaces of functions ensuring this exist). Then, given s examples, consider $f \in \mathcal{F}$ minimizing $\hat{L}(f) + \sqrt{\frac{32}{s}V(f)\log(e \times s)}$, 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 $V_k \log(e \times s) \leq s\epsilon^2/512$, the generalization error is lower than ϵ with probability at most $\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \times \exp(-s\epsilon^2/512)$ where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is assumed finite.

• the generalization error, with probability 1, converges to L^*.

Interpretation : The optimization of a compromise between empirical accuracy and regularization lead to the same properties as in theorem B, plus a stronger convergence rate property.

3 Results

This section presents in details results surveyed above. They make an intensive use of the results of Statistical Learning Theory presented in the previous section.

More precisely, Lemma 1 defines precisely the space of programs considered here, and carefully shows that it satisfies the hypotheses of Theorems A-C. This allows us to evaluate the VC-dimension of sets of programs, stated in Theorem 2. Then, announced results are derived. Finally, next we propose a new approach combining an a priori limit on VC-dimension (i.e. *size limit*) and a complexity penalization (i.e. *parsimony pressure*) and state in theorem 6 that this leads to both universal consistency and convergence to an optimal complexity of the program (i.e. *no bloat*).

We first prove the following

Lemma 1 : Let F be the set of functions which can be computed with at most t operations among :

- operations $\alpha \mapsto \exp(\alpha)$ (at most q times);
- operations $+$, $-$, $\times$, $/$;
- jumps conditioned on $>$, $\geq$, $=$, $\leq$, $=$;

and

- output 0 ;
- output 1 ;
- labels for jumps ;
- at most m constants ;
- at most z variables

by a program with at most n lines. We note $\log_2(x)$ the integer part (ceil) of $\log(x)/\log(2)$. Then F is included in H as defined in theorem C, for a given P with $t' = t + t\ max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $q' = q$, $d' = 1 + m$.

Interpretation : This lemma states that a family of programs as defined above is included in the parametrizations of one well-chosen program. This replaces a family of programs by one parametric program, and it will be useful for the computation of VC-dimension of a family of programs by theorem C.

Proof : In order to prove this result, we define below a program as in theorem above that can emulate any of these programs, with at most $t' = t + t\ max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $q' = q$, $d' = 1 + m$.
The program is as follows :

- label "inputs"
- initialize $variable(1)$ at value $x(1)$
- initialize $variable(2)$ at value $x(2)$
- ...
- initialize $variable(dim(x))$ at value $x(dim(x))$
- label "constants"
- initialize $variable(dim(x) + 1)$ at value a_1
- initialize $variable(dim(x) + 2)$ at value a_2
- ...
- initialize $variable(dim(x) + m)$ at value a_m
- label "Decode the program into c"
- operation decode c
- label "Line 1"
- operation $c(1, 1)$ with variables $c(1, 2)$ and $c(1, 3)$ and $c(1, 4)$
- label "Line 2"
- operation $c(2, 1)$ with variables $c(2, 2)$ and $c(2, 3)$ and $c(2, 4)$
- ...
- label "Line n"
- operation $c(n, 1)$ with variables $c(n, 2)$and $c(n, 3)$ and $c(n, 4)$

- label "output 0"
- output 0
- label "output 1"
- output 1

"operation decode c" can be developed as follows. Indeed, we need m real numbers, for parameters, and $4n$ integers $c(.,.)$, that we will encode as only one real number in $[0,1]$ as follows :
1. let $y \in [0,1]$
2. for each $i \in [1,\dots n]$:

- $c(i,1) = 0$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,1) = 1\ ;\ y = y - 1\ \}$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,1) = c(i,1) + 2\ ;\ y = y - 1\ \}$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,1) = c(i,1) + 4\ ;\ y = y - 1\ \}$

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

- $c(i,j) = 0$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,j) = 1\ ;\ y = y - 1\ \}$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,j) = c(i,j) + 2\ ;\ y = y - 1\ \}$
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,j) = c(i,j) + 4\ ;\ y = y - 1\ \}$
- ...
- $y = y * 2$
- if $(y > 1)$ then $\{\ c(i,j) = c(i,j) + 2^{log_2(z)-1}\ ;\ y = y - 1\ \}$

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

The result then derives from the rewriting of "operation $c(i,1)$ with variables c(i,2) and c(i,3)". This expression can be developed as follows:

- if $c(i,1) == 0$ then goto "output1"
- 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 "endOfInstruction(i)"

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

- label "exponential(i)"
- $a = exp(c)$
- goto "endOfInstruction(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 : Let F be the set of programs as in lemma 1, where $q' \geq q$, $t' \geq t + t\ max(3 + \log_2(n) + \log_2(z), 7 + 3\log_2(z)) + n(11 + max(9log_2(z), 0) + max(3log_2(z) - 3, 0))$, $d' \geq 1 + m$.

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

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

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

Interpretation : 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 1 in Theorem C. ■

We now consider how to use such results in order to ensure universal consistency. First, we show why simple empirical minimization (consisting in choosing one function such that $\hat{L}$ is minimum) does not ensure consistency. Precisely, we state that, for some distribution of examples, and some i.i.d sequence of examples $(x_1, y_1), \ldots, (x_n, y_n)$, there exists $P_1, \ldots, P_n, \ldots$ such that

$$\forall i \in [[1, n]] P_n(x_i) = y_i$$

and however

$$\forall n \in \mathbb{N} P(f(x) = y) = 0.$$

The proof is as follows. Consider the distribution with x uniformly drawn in $[0, 1]$ and y constant equal to 1. Consider P_n the program that compares its entry to x_1, x_2, ...,

x_n, and outputs 1 if the entry is equal to x_j for some $j \leq n$, and 0 otherwise else. With probability 1, this program output 0, whereas $y = 1$ with probability 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 accound a penalization for complex structures, can do the job, i.e. ensure universal consistency, and fast convergence when the solution can be written within finite complexity.

Theorem 3 : Consider q_f, t_f, m_f, n_f and z_f integer sequences, non-decreasing functions of f. Define $V_f = VCdim(H_f)$, where H_f is the set of programs with at most t_f lines executed, with z_f variables, n_f lines, q_f exponentials, and m_f constants.

Then with ${q'}_f = q_f$, ${t'}_f = t_f + t_f\, max(3 + \log_2(n_f) + \log_2(z_f), 7 + 3\log_2(z_f)) + n_f(11 + max(9log_2(z_f), 0) + max(3log_2(z_f) - 3, 0))$, ${d'}_f = 1 + m_f$,

$$V_f = ({d'}_f({q'}_f + 1))^2 + 11{d'}_f({q'}_f + 1)({t'}_f + \log_2(9{d'}_f({q'}_f + 1)))$$

or, if $\forall f\ q_f = 0$ then define $V_f = 4{d'}_f({t'}_f + 2)$.

Then, being given s examples, consider $f \in \mathcal{F}$ minimizing $\hat{L}(f) + \sqrt{\frac{32}{s} V(f) \log(e \times s)}$, where $V(f)$ is the min of all k such that $f \in \mathcal{F}_k$.

Then, if $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is finite,

- the generalization error, with probability 1, converges to L^*.
- if one optimal rule belongs to $\mathcal{F}_k$, then for any s and ϵ such that $V_k \log(e \times s) \leq s\epsilon^2/512$, the generalization error is lower than $L^* + \epsilon$ with probability at most $\Delta \exp(-s\epsilon^2/128) + 8s^{V_k} \times \exp(-s\epsilon^2/512)$ where $\Delta = \sum_{j=1}^{\infty} \exp(-V_j)$ is assumed finite.

Interpretation : Genetic programming for bi-class classification, provided that structural risk minimization is performed, is universally consistent and verifies some convergence rate properties.

Proof : Just plug theorem D in theorem 2. ■

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 4:

Consider P_s a sequence of functions such that $P_s \in \mathcal{F}_{V(s)}$, with $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots$, where $\mathcal{F}_V$ is a set of functions from X to $\{0, 1\}$ with VC-dimension bounded by V.
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, $(L(P_s) \overset{s \to \infty}{\longrightarrow} L^*) \Longrightarrow (V(P_s) \overset{s \to \infty}{\longrightarrow} \infty)$.

Interpretation : This is structural bloat : if your space of programs approximates but does not contain the optimal function, then bloat occurs.

Proof:

Define $\epsilon(V) = L_V - L^*$. Assume that $\forall V\ \epsilon(V) > 0$. ϵ is necessarily non-increasing.

Consider V_0 a positive integer ; let us prove that if s is large enough, then $V(P_s) \geq V_0$.

There exists ϵ_0 such that $\epsilon(V_0) > \epsilon_0 > 0$.

For s large enough, $L(P_s) \leq L^* + \epsilon_0$, hence $L_{V_s} \leq L^* + \epsilon_0$, hence $L^* + \epsilon(V_s) \leq L^* + \epsilon_0$, hence $\epsilon(V_s) \leq \epsilon_0$, hence $V_s > V_0$. ■

We now show that the usual procedure defined below, consisting in defining a maximum VC-dimension depending upon the sample size (as usually done in practice and as recommended by theorem B) and then using a moderate family of functions, leads to bloat. With the same hypotheses as in theorem B, we can state

Theorem 5 (bloat theorem for empirical risk minimization with relevant VC-dimension): Let $\mathcal{F}_1, \ldots, \mathcal{F}_k \ldots$ non-empty sets of functions with finite VC-dimensions $V_1, \ldots, V_k, \ldots$ Let $\mathcal{F} = \cup_n \mathcal{F}_n$. Then, given s examples, consider $\hat{P} \in \mathcal{F}_s$ minimizing the empirical risk $\hat{L}$ in F_s.
¿From Theorem B we already know that if $V_s = o(s/log(s))$ and $V_s \to \infty$, then $P(L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s, V_s, \delta)) \geq 1 - \delta$, and $L(\hat{P}) \to \inf_{P \in \mathcal{F}} L(P)$ a.s..

We will now state that if $V_s \to \infty$, and noting $V(f) = min\{V_k; f \in \mathcal{F}_k\}$, then $\forall V_0, P_0 > 0$, $\exists P$, 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 $P(V(\hat{P}) \leq V_0) \leq P_0$.

Interpretation : The result in particular implies that for any V_0, there is a distribution of examples 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, even if this limit depends upon the number of examples (a fortiori if there's no limit).

Proof (of the part which is not theorem B) :

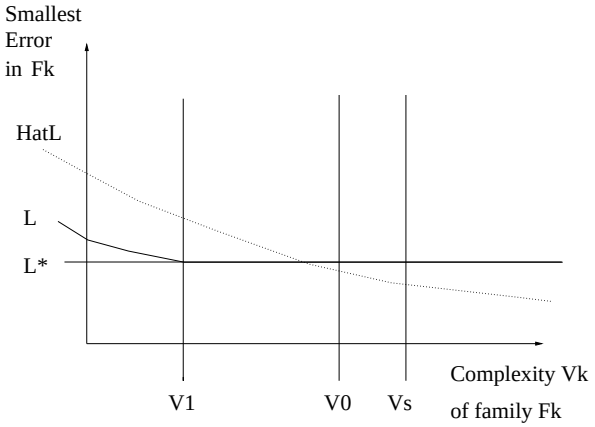

Fig. 1. Illustration of the proof. With a larger k, $\mathcal{F}_k$ has a smaller best error.

See figure 3 for a figure illustrating the proof. Consider $V_0 > 0$ and $P_0 > 0$. Consider α such that $(e\alpha/2^\alpha)^{V_0} \leq P_0/2$. Consider s such that $V_s \geq \alpha V_0$. Let $d = \alpha V_0$. Consider $x_1, \ldots, x_d$ d points shattered by $\mathcal{F}_d$; such a family of d points exist, by definition of $\mathcal{F}_d$. Define the probability measure P by the fact that X and Y are

independent and $P(Y=1)=\frac{1}{2}$ and $P(X=x_i)=\frac{1}{d}$. Then, the following holds, with Q the empirical distribution (the average of Dirac masses on the x_i's) :

1. no empty x_i's : $P(E_1) \rightarrow 0$ where E_1 is the fact that $\exists i; Q(X=x_i)=0$, as $s \rightarrow \infty$.
2. no equality : $P(E_2) \rightarrow 0$ 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 $\mathcal{F}_{V_0}$: $P(E_3|E_2 \text{ does not hold}) \leq S(d,d/\alpha)/2^d$ 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, i.e. the cardinal of $\mathcal{F}_{d/\alpha}$ restricted to $\{x_1,\ldots,x_d\}$.

We now only have to use classical results. It is well known in VC-theory that $S(a,b) \leq (ea/b)^b$ (see for example [5, chap.13]), hence $S(d,d/\alpha) \leq (ed/(d/\alpha))^{d/\alpha}$ and $P(E_3|E_2 \text{ does not hold}) \leq (e\alpha)^{d/\alpha}/2^d \leq P_0/2$. If n is sufficiently large to ensure that $P(E_2) \leq P_0/2$ (we have proved above that $P(E_2) \rightarrow 0$ as $s \rightarrow \infty$) then

$$P(E_3) \leq P(E_3|\neg E_2) \times P(\neg E_2) + P(E_2) \leq P(E_3|\neg E_2) + P(E_2) \leq P_0/2 + P_0/2 \leq P_0$$

■

We now show that, on the other hand, it is possible to optimize a compromise between optimality and complexity in an explicit manner (e.g., replacing 1 % precision with 10 lines of programs or 10 minutes of CPU) :

Theorem 5' (bloat-control theorem 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, being given s examples, 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 $V_s \rightarrow \infty$, then $\tilde{L}(\hat{P}) \rightarrow \inf_{P \in \mathcal{F}} \tilde{L}(P)$ a.s. where $\tilde{L}(P) = L(P) + W(P)$.

Interpretation : Theorem 5' 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 ([20, 21]) 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 : the drawback of this approach is that we have lost universal consistency and consistency (in the general case, the misclassification rate in generalization will not converge to the Bayes error, and whenever an optimal program exists, we will not necessarily converge to its efficiency).

Proof : See figure 3 for a figure illustrating the proof. $\sup_{P \in \mathcal{F}_s} |\hat{\tilde{L}}(P) - \tilde{L}(P)| \leq \sup_{P \in \mathcal{F}_s} |\hat{L}(P) - L(P)| \leq \epsilon(s,V_s) \rightarrow 0$, almost surely, by theorem A'. Hence the expected result. ■

We now turn our attention to a more complicated case where we want to ensure universal consistency, but we want to avoid a non-necessary bloat ; e.g., we require that if an optimal program exists in our family of functions, then we want to converge to its error rate, without increasing the complexity of the program. We consider a merge between regularization and bounding of the VC-dimension ; we penalize the complexity

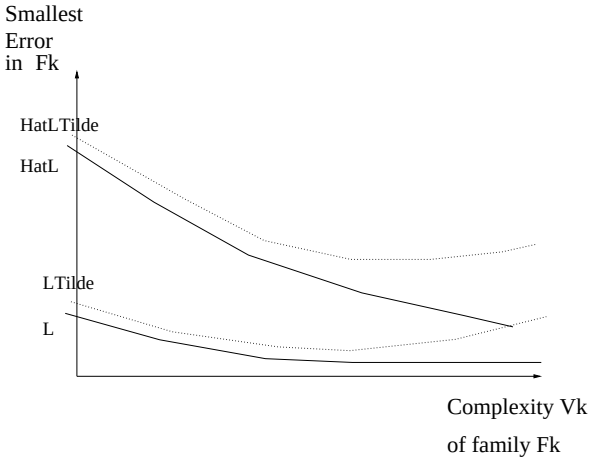

Fig. 2. Illustration of the proof. With a larger k, $\mathcal{F}_k$ has a smaller best error, but the penalization is stronger than the difference of error.

(e.g., length) of programs by a penalty term $R(s,P) = R(s)R'(P)$ depending upon the sample size and upon the program ; $R(.,.)$ is user-defined and the algorithm will look for a classifier with a small value of both R' and L. We study both the universal consistency of this algorithm (i.e. $L \to L^*$) and the no-bloat theorem (i.e. $R' \to R'(P^*)$ when P^* exists).

Theorem 6 : 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 $\hat{P}$ minimizing $\hat{\hat{L}}(P) = \hat{L}(P) + R(s,P)$ in $\mathcal{F}_s$ and assume that $R(s,.) \geq 0$. Then (consistency), whenever $\sup_{P \in \mathcal{F}_{V_s}} R(s,P) = o(1)$, $L(\hat{P}) \to \inf_{P \in \mathcal{F}} L(P)$ almost surely (note that for well chosen family of functions, $\inf_{P \in \mathcal{F}} L(P) = L^*$). Moreover, assume that $\exists P^* \in \mathcal{F}_{V^*}\ L(P^*) = L^*$. Then with $R(s,P) = R(s)R'(P)$ and with $R'(s) = \sup_{P \in \mathcal{F}_{V_s}} R'(P)$:

1. **non-asymptotic no-bloat theorem :** $R'(\hat{P}) \leq R'(P^*) + (1/R(s))2\epsilon(s, V_s, \delta)$ with probability at least $1 - \delta$ (this result is in particular interesting for $\epsilon(s, V_s, \delta)/R(s) \to 0$, what is possible for usual regularization terms as in theorem D,
2. **almost-sure no-bloat theorem :** if $R(s)s^{(1-\alpha)/2} = O(1)$, then almost surely $R'(\hat{P}) \to R'(P^*)$ 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, $R'(\hat{P}) = R'(P^*)$.
3. **convergence rate :** with probability at least $1 - \delta$,

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

where $\epsilon(s,V,\delta) = \sqrt{\frac{4-\log(\delta/(4s^{2V}))}{2s-4}}$ is an upper bound on $\epsilon(s,V) = \sup_{f\in\mathcal{F}_V} |\hat{L}(f) - L(f)|$ (given by theorem A), true with probability at least $1-\delta$.

Interpretation : Combining a code limitation and a penalization leads to universal consistency without bloat.

Remarks : The usual $R(s,P)$ as used in theorem D or theorem 3 provides consistency and non-asymptotic no-bloat. 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 genetic programming and could be used in other areas.

As shown in proposition 4, the no-bloat results require the fact that $\exists V^* \exists P^* \in \mathcal{F}_{V^*}\ L(P^*) = L^*$.

Interestingly, the convergence rate is reduced when the regularization is increased in order to get the almost sure no-bloat theorem.

Proof : Define $\epsilon(s,V) = \sup_{f\in\mathcal{F}_V} |\hat{L}(f) - L(f)|$. Let us prove the consistency. For any P, $\hat{L}(\hat{P}) + R(s,\hat{P}) \leq \hat{L}(P) + R(s,P)$. On the other hand, $L(\hat{P}) \leq \hat{L}(\hat{P}) + \epsilon(s,V_s)$. So :

$$L(\hat{P}) \leq (\inf_{P\in\mathcal{F}_{V_s}} (\hat{L}(P) + R(s,P))) - R(s,\hat{P}) + \epsilon(s,V_s)$$

$$\leq (\inf_{P\in\mathcal{F}_{V_s}} (L(P) + \epsilon(s,V_s) + R(s,P))) - R(s,\hat{P}) + \epsilon(s,V_s)$$

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

as $\epsilon(s,V_s) \to 0$ almost surely[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 $L(\hat{P}) \to \inf_{P\in\mathcal{F}} L(P)$ a.s.

We now focus on the proof of the "no bloat" result :

By definition of the algorithm, for s sufficiently large to ensure $P^* \in \mathcal{F}_{V_s}$, $\hat{L}(\hat{P}) + R(s,\hat{P}) \leq \hat{L}(P^*) + R(s,P^*)$ hence with probability at least $1-\delta$,

$$R'(\hat{P}) \leq R'(P^*) + (1/R(s))(L^* + \epsilon(s,V_s,\delta) - L(\hat{P}) + \epsilon(s,V_s,\delta))$$

$$\text{hence } R'(\hat{P}) \leq R'(V^*) + (1/R(s))(L^* - L(\hat{P}) + 2\epsilon(s,V_s,\delta))$$

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

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

$$R'(\hat{P}) \leq R'(P^*) + 1/R(s)(L^* + \epsilon(s,V_s) - L(\hat{P}) + \epsilon(s,V_s))$$

$$\text{hence } R'(\hat{P}) \leq R'(P^*) + 1/R(s)(L^* - L(\hat{P}) + 2\epsilon(s,V_s))$$

$$R'(\hat{P}) \leq R'(P^*) + 1/R(s)2\epsilon(s,V_s)$$

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

For any $\epsilon > 0$, we consider the probability of $\epsilon(s,V_s)/R(s) > \epsilon$, and we sum over $s > 0$. By the Borell-Cantelli lemma, the finiteness of this sum is sufficient for the almost sure convergence to 0.

[2] See theorem A'

The probability of $\epsilon(s,V_s)/R(s) > \epsilon$ is the probability of $\epsilon(s,V_s) > \epsilon R(s)$. By theorem A, this is bounded above by $O(\exp(2V_s \log(s) - 2s\epsilon^2 R(s)^2))$. This has finite sum for $R(s) = \Omega(s^{-(1-\alpha)/2})$.

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,

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

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

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

4 Extensions

We have studied above :

- the method consisting in minimizing the empirical error, i.e. the error observed on examples (leading to bloat (this is an a fortiori consequence of theorem 5) without universal consistency (see remark before theorem 3)) ;
- the method consisting in minimizing the empirical error, i.e. the error observed on examples, with a hard bound on the complexity (leading to universal consistency but bloat, see theorem 5) ;
- the method, inspired from (but slightly adapted against bloat) structural risk minimization, consisting in minimizing a compromize between the empirical error *and* a complexity bound including size and computation-time (see theorem 6).

We study the following other cases now :

- the case in which the level of complexity is chosen through resamplings, i.e. cross-validation or hold out ;
- the case in which the complexity penalization does not include any time bound but only size bounds ;

We mainly conclude that penalization is necessary, cannot be replaced by cross-validation, cannot be replaced by hold-out, and must include time-penalization.

4.1 About the use of cross-validation or hold-out for avoiding bloat and choosing the complexity level

Note UC for universal consistency and ERM for empirical risk minimization. We considered above different cases :

- evolutionary programming with only "ERM" fitness ;
- evolutionary programming with ERM+bound (leading to UC + bloat) ;
- evolutionary programming with ERM+penalization+bound (leading to UC without bloat).

One can now consider some other cases :

– hold out in order to choose between different complexity classes (i.e., in the Pareto-front corresponding to the compromise between ERM and complexity, choose the function by hold out) ;
– idem through cross-validation.

This section is devoted to these cases.

First, let's consider hold-out for choosing the complexity level. Consider that the function can be chosen in many complexity levels, $F_0 \subset F_1 \subset F_2 \subset F_3 \subset \ldots$, where $F_i \neq F_{i+1}$. Note $L(f, X)$ the error rate of the function f in the set X of examples:

$$L(f, X) = \frac{1}{n} \sum_{i=1}^{n} l(f, X_i)$$

where $l(f, X_i) = 1$ if f fails on X_i and 0 otherwise. Define $f_k = \arg\min_{F_k} L(., X_k)$. In hold-out, $f = f_{k*}$ where $k* = \arg\min_k l_k$ where $l_k = L(f_k, Y_k)$.

In all the sequel, we assume that $f \in F_k \Rightarrow 1 - f \in F_k$ and that $VC - dim(F_k) \rightarrow \infty$ as $k \rightarrow \infty$. We consider that all X_k's and Y_k's have the same size n.

There are different cases : $X_k = Y_k$ and $\forall k, X_k = X_0$ is the naive case (studied above). The case with hold out leads to different cases also :

– **Greedy case:** all X_k's and Y_k's are independent.
– **Case with pairing:** X_0 is independent of Y_0, $\forall k, X_k = X_0 \wedge Y_k = Y_0$.

Case of greedy hold-out.

– consider the case of an output y independent of the input x, and $P(y = 1) = P(y = 0) = \frac{1}{2}$.
– $k^* = \inf\{n \in \mathbb{N}; l_n = 0\}$.
– k^* is therefore a Poisson law with parameter $1/2^n$. Its expectation is TODO and its standard deviation is TODO
– therefore, almost surely, $k^* \rightarrow \infty$ as $n \rightarrow \infty$. This is shown with *one* distribution, which does not depend upon the number of examples. This happens whereas an optimal function lies in F_0.

Case of hold-out with pairing.

– Consider $V \in \mathbb{N} = VC - dim(F_v)$ with v minimal realizing this condition.
– Consider $A = \{a_1, \ldots, a_V\}$, a set of points shattered by F_v.
– Consider a distribution of examples with x uniform on A and y independent of x with $P(y = 1) = P(y = 0) = \frac{1}{2}$.
– Consider $\hat{P}_X$ the empirical distribution associated to X and $\hat{P}_Y$ the empirical distribution associated to Y.
– Then, as $n \rightarrow \infty$, with $E_X = \{\exists i, \hat{P}_X(y = 1|x = a_i) = \frac{1}{2}\}$, $P(E_X) \rightarrow 0$.
– Then, as $n \rightarrow \infty$, with $E_Y = \{\exists i, \hat{P}_Y(y = 1|x = a_i) = \frac{1}{2}\}$, $P(E_Y) \rightarrow 0$.
– There is at least one function on A which does not belong in F_{k-1}.
– With probability at least $(1 - P(E_Y))/2^V$, this function is optimal for $L(., Y_0)$.
– With probability at least $(1 - P(E_X))/2^V$, f_k is equal to this function.

– Combining the two probabilities above, as the events are independent, we see that with probability at least $p(v,n) = ((1-\epsilon(v,n))/2^V)^2$, $k^* \geq v$, where $\epsilon(v,n) \to 0$ as $n \to \infty$:

$$P(k^* \geq v) > p(v,n)$$

– this implies the **first result** : $P(k^* \geq v)$ does not go to 0, whereas a function in F_0 is optimal.
– Now, let's consider that we can change the distribution as n moves.
– For n sufficiently large, choose v maximal such that $p(v,n) =\geq 1/n$ and F_v has VC-dimension greater than the VC-dimension of F_{v-1}. Consider the distribution associated to v as above (uniform on A, a set of shattered point).
– Consider $N = |\{n \in \mathbb{N}; k^* \geq v\}|$.
– N has infinite expectation $\leq \sum_{n \geq n_0} 1/n$.
– Therefore, infinitely often (almost surely), $k^* \geq v$ and $v \to 0$, therefore $\limsup k^* = \infty$.

We have therefore shown, *with a distribution dependent on* n, that $k^* \to \infty$. And for a distribution that does not depend upon n, that $P(k^* < v)$ is lower bounded. In both cases, an optimal function lies in F_0.

We now turn our attention to the case of cross-validation. We formalize N-cross-validation as follows :

$$f_k^i = \arg\min_{F_k} L(., {X'}_k^i)$$
$${X'}_k^i = (X_k^1, X_k^2, \ldots, X_k^{i-1}, X_k^{i+1}, X_k^{i+2}, \ldots, X_k^N) \text{ for } i \leq N$$
$$k^* = \arg\min \frac{1}{N} \sum_{i=1}^{N} L(f_k^i, X_k^i)$$

Greedy cross-validation could be considered as in the case of hold out above. This leads to the same result (for some distribution, $k^* \to \infty$). We therefore only consider cross-validation with pairing :

$$X_k^i = X^i$$

We only consider cross-validation as a method for computing k^*, and not for computing the classifier. We note $\hat{P}_i$ the empirical law associated to X^i.

We consider A a set of points shattered by F_v, $|A| = V$, A not shattered by F_{v-1}. We consider $f \in F_v$ realizing a dichotomy of A that is not realized by F_{v-1}. We define E_i the event $\{\forall a \in A; \hat{P}_i(y = f(a)|x = a) > \frac{1}{2}\}$. We assume that the distribution of examples is, for x, uniform on A, and for y, independently of x, uniform on $\{0,1\}$. The probability of E_i goes to $(\frac{1}{2})^{|A|}$. The probability of $E = \cap_i E_i$ goes to $(\frac{1}{2})^{N|A|} > 0$ as $n \to \infty$. In particular, almost surely, infinitely often as $n \to \infty$, E occurs. When E occurs,

– all the f_k^i are equal to f ;
– for any $g \in F_{v-1}$, $L(g, X_k^i) < L(f, X_k^i)$;
– therefore, $k^* \geq v$.

We therefore have the following result :

Theorem : one can not avoid bloat with only hold-out or cross-validation.

Consider greedy hold-out, hold out with pairing and cross-validation with pairing. Then,

- for some well-chosen distribution of examples, greedy hold-out almost surely leads to $k^* \rightarrow \infty$ whereas an optimal function lies in F_0.
- whatever may be $V = VC - dimension(F_v)$, for some well-chosen distribution, hold-out with pairing almost surely leads to $k^* > V$ infinitely often whereas an optimal function lies in F_0.
- whatever may be $V = VC - dimension(F_v)$, for some well-chosen distribution, cross-validation with pairing almost surely leads to $k^* > V$ infinitely often whereas an optimal function lies in F_0.

4.2 Is time-complexity required ?

Consider any learning algorithm working on a sequence of i.i.d examples $(x_1, y_1), \ldots, (x_n, y_n)$ and outputting a program. We formalize as follows the fact that this algorithm does not take into account any form of time-complexity but only the size of programs :

If the learning program outputs P, then there is not program P' with the same length as P that has a better empirical error rate.

We show in the sequel that such a learning program can not verify convergence rates as shown in theorem 6, i.e. a guaranteed convergence rate in $O(1/\sqrt{n})$ when an optimal function has bounded complexity. In the sequel, we assume that the reader is familiar with statistical learning theory and shattering properties ; the interested reader is referred to [5].

The main element is that theorem C does not hold without bounded time. The following program has bounded length, only one parameter α, but generates as $\alpha \in \mathbb{R}$ a family of functions which shatters an infinite set :

- consider x the entry in $\mathbb{R}$ and $\alpha \in \mathbb{R}$ a parameter ;
- if $x \leq 0$ then go to FINISH.
- label BEGIN.
- if $x > \frac{1}{2}$, go to FINISH.
- $x \leftarrow 2x$.
- $\alpha \leftarrow 2\alpha$.
- PROJECT
- if $\alpha > 1$ then $\alpha \leftarrow \alpha - 1$; goto PROJECT.
- goto BEGIN.
- label FINISH.
- if $\alpha \geq 0.5$, output 1 and stop.
- output 0 and stop.

Consider $(\alpha, x) \in]0, 1] \times [0, 1]$.

This program shifts α and x to the left until $x > \frac{1}{2}$. It then replies 1 if and only if α, after shift, has its first digit equal to 1. Therefore, this program can realize any

dichotomy of $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots\}$. This is exactly the definition of the fact that this set is shattered.

So, we have shown that an family of functions shattering an infinite set was included in the set of programs with bounded length.

Now, consider a learning program which has a guaranteed convergence rate in a family of functions including the family of functions computed by the program above. Ie, we assume that

Theorem : fitnesses without time-complexity-pressure do not ensure consistency. What ever may be the sequence $a_1, \dots, a_n, \dots$ decreasing to 0, there's no learning program ensuring that for any distribution of examples such that $P(y = f(x)) = 1$ for some f with bounded length, the expectation of $P(P_n(x) \neq y)$ is $O(a_n)$.

5 Conclusion

In this paper, we have proposed a theoretical study of two important issues in Genetic Programming known as universal consistency and code bloat. We have shown that GP trees used in symbolic regression (involving the four arithmetic operations, the exponential function, and ephemeral constants, as well as test and jump instructions) could benefit from classical results from Statistical Learning Theory (thanks to Theorem C and Lemma 1). This has led to two kinds of original outcomes : i) some results about Universal Consistency of GP, i.e. almost sure asymptotic convergence to the optimal error rate, ii) results about the bloat. Both the unavoidable structural bloat in case the ideal target function does not have a finite description, and the functional bloat, for which we prove that it can be avoided by simultaneously bounding the length of the programs with some *ad hoc* bound and using some parsimony pressure in the fitness function. Some negative results have been obtained, too, such as the fact though structural bloat was known to be unavoidable, functional bloat might indeed happen even when the target function does lie in the search space, but no parsimony pressure is used. Interestingly, all those results (both positive and negative) about 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 regression problems, but 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 tends to give arguments to the "fitness causes bloat" explanation [10]. It might be possible to study the impact of size-preserving mechanisms (e.g. specific variation operators, like size-fair crossover [9] or fair mutations [11]) as somehow contributing to the regularization term in our final result ensuring both Universal Consistency and no-bloat.

Acknowledgements This work was supported in part by the PASCAL Network of Excellence. We thank Bill Langdon for very helpful comments.

References

1. M. Antony and P. Bartlett. Neural network learning : Theoretical foundations, cambridge university press. 1999.

2. W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.
3. T. Blickle and L. Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms Workshop at KI-94*, pages 33–38. Max-Planck-Institut für Informatik, 1994.
4. J. M. Daida. What makes a problem gp-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165 – 191, 2001.
5. L. Devroye, L. Györfi, and G. Lugosi. A probabilistic theory of pattern recognition, springer. 1997.
6. S. Gustafson, A. Ekart, E. Burke, and G. Kendall. Problem difficulty and code growth in Genetic Programming. *GP and Evolvable Machines*, 4(3):271–290, 2004.
7. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
8. W. B. Langdon. The evolution of size in variable length representations. In *ICEC'98*, pages 633–638. IEEE Press, 1998.
9. W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming And Evolvable Machines*, 1(1/2):95–119, 2000.
10. W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In J. Koza, editor, *Late Breaking Papers at GP'97*, pages 132–140. Stanford Bookstore, 1997.
11. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming III*, pages 163–190. MIT Press, 1999.
12. S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. L. et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann Publishers, 2002.
13. N. F. McPhee and J. D. Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.
14. P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.
15. A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. C. et al., editor, *Artificial Evolution VI*. Springer Verlag, 2001.
16. S. Silva and J. Almeida. Dynamic maximum tree depth : A simple technique for avoiding bloat in tree-based gp. In E. C.-P. et al., editor, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1776–1787. Springer-Verlag, 2003.
17. T. Soule. Exons and code growth in genetic programming. In J. A. F. et al., editor, *EuroGP 2002*, volume 2278 of *LNCS*, pages 142–151. Springer-Verlag, 2002.
18. T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, 1998.
19. V. Vapnik. The nature of statistical learning theory, springer. 1995.
20. B.-T. Zhang and H. Mulenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation Vol. 3(1)*, 1995.
21. B.-T. Zhang, P. Ohm, and H. Muhlenbein. Evolutionary induction of sparse neural trees. *Evolutionary Computation, vol. 5, no. 2, pp. 213-236*, 1997.