# A Statistical Learning Perspective of Genetic Programming

Merve Amil, Nicolas Bredeche, Christian Gagné, Sylvain Gelly, Marc Schoenauer, Olivier Teytaud

# A Statistical Learning Perspective of Genetic Programming

Nur Merve Amil, Nicolas Bredeche, Christian Gagné,
Sylvain Gelly, Marc Schoenauer, and Olivier Teytaud

Equipe TAO – INRIA Futurs, LRI, Bat. 490,
Université Paris-Sud, 91405 Orsay CEDEX, France
olivier.teytaud@inria.fr

**Abstract.** This paper proposes a theoretical analysis of Genetic Programming (GP) from the perspective of statistical learning theory, a well grounded mathematical toolbox for machine learning. By computing the Vapnik-Chervonenkis dimension of the family of programs that can be inferred by a specific setting of GP, it is proved that a parsimonious fitness ensures universal consistency. This means that the empirical error minimization allows convergence to the best possible error when the number of test cases goes to infinity. However, it is also proved that the standard method consisting in putting a hard limit on the program size still results in programs of infinitely increasing size in function of their accuracy. It is also shown that cross-validation or hold-out for choosing the complexity level that optimizes the error rate in generalization also leads to bloat. So a more complicated modification of the fitness is proposed in order to avoid unnecessary bloat while nevertheless preserving universal consistency.

## 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 [10,11]. 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 [23].

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. Section 2 shortly exposes issues of code bloat with GP. Section 3 and 4 present all the aforementioned results about code bloat avoidance and UC and propose a new approach ensuring both. Then, Section 5 provides some extensions of the previous theoretical results on the use of cross-validation and hold-out methodologies. Follows some experimental results in Section 6, illustrating the accuracy of the theoretical results. Section 7 finally concludes this paper with a discussion on the consequences of those theoretical results for GP practitioners and uncover some perspectives of work.

## 2 Code Bloat in GP

Due to length constraints, we do not introduce here some important theories around code bloat: introns, fitness causes bloat, and removal bias. The reader is refered to [1,3,11,13,16,17,21] for more informations around that. Some common solutions against bloat rely either on specific operators (e.g. size-fair crossover [12], or different fair mutation [14]), on some parsimony-based penalization of the fitness [22] or on abrupt limitation of the program size such as the one originally used by Koza [10]. Also, some multi-objective approachs have been proposed [2,5,7,15,19]. Some other more particular solutions have been proposed but are not widely used yet [18,24]. Also, all proofs are removed due to length constraints. Readers familiar with mathematics like in e.g. [6] should however be able to guess the main ideas.

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 [9]. 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). 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 $\infty$ 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

length of programs found by GP runs to infinity as the number of test cases runs 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.

Another important issue is to study the convergence of the function given by GP toward the actual function used to generate the test cases, under some sufficient conditions and when the number of test cases goes to infinity. This property is known in statistical learning as *Universal Consistency* (UC). Note that this notion is slightly different from that of universal approximation, commonly referred in symbolic regression, where GP search using operators $\{+, *\}$ is assumed to be able to approximate any continuous function. UC is rather 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. Or more precisely, UC can be stated informally as follows (a formal definition will be given later):

A GP setting corresponds to symbolic regression from examples if it takes as inputs a finite number of examples $x_1, \ldots, x_n$ with their associated labels $y_1, \ldots, y_n$ and outputs a program $P_n$. Universal consistency holds if, when pairs $(x_1, y_1), \ldots, (x_n, y_n)$ (test cases) are identically independently distributed as the random variable $(x, y)$, $L(P_n) \rightarrow L^*$ where $L(p) = Pr(y \neq p(x))$ and where $L^* = \inf_{p \text{ measurable}} L(p)$. In all of this paper, Pr(.) denotes probabilities, as the traditional notation $P(.)$ is used for programs.

## 3 Negative Results without Regularization and Resampling

Definition 1 precisely defines the programs space under examination. Theorem 1 evaluates its VC-dimension [23]. Many theorems, in the sequel, are based only on VC-dimensions and hold for other sets of programs as well.

It should be noted the mildness of the hypothesis behind our results. 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.

**Definition 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:*

*(1) A run uses at most $t$ operations. (2) Each line contains one operation among the followings:*

- *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.*

The following property is central for our results.

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

We now consider how to use such results in order to ensure UC. 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 $\forall n \in \mathbb{N}, \forall i \in \{1,2,\ldots,n\}$ $P_n(x_i) = y_i$, and however $\forall n \in \mathbb{N}$ $\Pr(P_n(x) = y) = 0$. 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 (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 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* $V_k = 4d'_k(t'_k + 2)$.
– *Otherwise,* $V_k = (d'_k(q'_k + 1))^2 + 11d'_k(q'_k + 1)(t'_k + \log_2(9d'_k(q'_k + 1)))$.

*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* $L^*$; *moreover, if one optimal program belongs to* $\mathcal{F}_k$, *then for any* $s$ *and* $\epsilon$ *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:** *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.*

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 1 (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* $\forall V \quad L_V > L^*$. *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:** *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 for any* $\mathcal{F}_1, \mathcal{F}_2, \ldots$, *the assumption* $\forall V \quad L_V > L^*$ *holds simultaneously for all* $V$ *for many distributions, as we consider countable unions of families with finite VC-dimension (e.g. see [6, chap. 18]).*

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 [20]) defined below, consisting in defining a maximum VC-dimension depending upon the sample size and then using a family of functions accordingly, leads to bloat.

**Theorem 3 (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 theorems about the method of Sieves, 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:** *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, $V(\hat{P}) \geq V_0$ *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 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.*

## 4 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. $L \to L^*$) 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 3).

**Theorem 4 (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* $R(s, .) \geq 0$. *Assume that* $\sup_{P \in \mathcal{F}_{V_s}} R(s, P) = o(1)$. *Then,* $L(\hat{P}_s) \to \inf_{P \in \mathcal{F}} L(P)$ *almost surely. Note that for well chosen family of functions,* $\inf_{P \in \mathcal{F}} L(P) = L^*$. *Moreover, assume that* $\exists P^* \in \mathcal{F}_{V^*} \quad L(P^*) = L^*$. *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* $\epsilon(s, V_s, \delta)/R(s) \to 0$.
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* $1-\delta$*,*

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

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

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

## 5 Some Negative Results with Subsampling: Hold-Out or Cross-Validation

When one tries to learn a relation between $x$ and $y$, the "true" cost function (typically the mean squared error of a given approximate relation, which is an expectation under some usually unknown law of probability) is generally not available. It is usually replaced by its empirical mean on a finite sample. Minimizing this empirical mean is natural, but this can be done over various families of functions (e.g. trees with depth 1, 2, 3, and so on). Choosing between these various levels is hard. Typically, the empirical mean decreases as the complexity is increased, but this decrease is not generally a decrease of the generalization error, as trees of larger depth have usually a very bad generalization error due to overfitting. Therefore, the problem is somewhat multi-objective: there is a conflict between the empirical error and the complexity level. This multi-objective optimization setting has been studied in [2,5,7].

This section is devoted to hold-out and cross-validation as tools for UC without bloat. First, let's consider hold-out for choosing the complexity level. Consider $X_0, \ldots, X_N$, $Y_0, \ldots, Y_N$, $2(N+1)$ samples (each of them consisting in $n$ examples, i.e. $X_i = (X_{i,1}, X_{i,2}, \ldots, X_{i,n})$ and $Y_i = (Y_{i,1}, Y_{i,2}, \ldots, Y_{i,n})$), the $X_i$'s being learning sets, the $Y_i$'s being (hold-out) test sets. 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 $F_0$ is non-empty and $F_i \neq F_{i+1}$. Note $\hat{L}_k\left(f\right)$ the error rate of the function $f$ in the set $X_k$ of examples: $\hat{L}_k\left(f\right) = \frac{1}{n}\sum_{i=1}^{n} l\left(f, X_{k,i}\right)$ where $l\left(f,x\right) = 1$ if $f$ fails on $x$ and 0 otherwise. Define $f_k = \arg\min_{F_k} \hat{L}_k\left(.\right)$. In hold-out, after the complete learning, the resulting classifier is $f_{k^*(n)}$, where $k^*(n) = \arg\min_{k \leq N(n)} l_k$ and $l_k = \frac{1}{n}\sum_{i=1}^{n} l\left(f_k, Y_{k,i}\right)$. In the sequel, we assume that $f \in F_k \Rightarrow 1 - f \in F_k$ and that $VCdim(F_k) \to \infty$ as $k \to \infty$. The case with hold-out leads to different cases, namely:

**Greedy case:** all $X_k$'s and $Y_k$'s are independent; this means that we test separately each complexity level $F_k$ with different learning sets $X_k$ and test sets $Y_k$.

**Case with pairing:** $X_0$ is independent of $Y_0$, $\forall k, X_k = X_0$ and $\forall k, Y_k = Y_0$; this means that we use the same learning set for all complexity levels and the same test set for all complexity levels. This case is far more usual.

**Theorem 5 (No bloat avoidance with greedy hold-out).** *Consider greedy hold-out for choosing between complexity levels* $0, 1, \ldots, N(n)$. *If* $N(n)$ *is a constant, then for some distribution of examples* $\forall k \in [0, N], P(k^*(n) = k) \to 1/(N+1)$. *If* $N(n) \to \infty$ *as* $n \to \infty$, *then for some distribution of examples such that an optimal function lies in* $F_0$, *greedy hold-out leads to* $k^*(n) \to \infty$ *as* $n \to \infty$ *and therefore* $\limsup_{n\to\infty} k^*(n) = \infty$.

All the following results are in the general case of $N$ a non decreasing function of $n$.

**Proposition 2 (Bloat cannot be controlled by hold-out with pairing, first result).** *Consider the case with pairing. For arbitrarily large* $v$, *there exists a distribution with optimal function in* $F_0$ *such that* $\liminf_{n\to\infty} \Pr(k^*(n) \geq v) > 0$.

Now, let's consider a distribution that depends on $n$. This is interesting, as it provides lower bounds on what can be guaranteed, for a given value of $n$, independently of the distribution. For technical reasons, and without loss of generality with renumbering of the $F_k$, we assume that $F_{v+1}$ has a VC-dimension larger than $F_v$. We can show that, *with a distribution dependent on* $n$, $\limsup_{n\to\infty} k^*(n) \to \infty$. This leads to this other negative theorem about the control of bloat by hold-out.

**Proposition 3 (Bloat can not be controlled by hold-out with pairing, second result).** $\limsup_n k^*(n) = \infty$, *where the distribution depends on* $n$ *but is always such that an optimal function lies in* $F_0$.

This result above is in the setting of a distribution which depends on $n$; it is of course not interesting for modelizing the evolution of one particular problem as the number of examples increases, but it shows that no bound on $k^*(n)$ for $n \geq n_0$ can be provided, whatever may be $n_0$, for hold-out with pairing, unless the distribution of problems is taken into account.

**Cross-Validation for the Control of Bloat.** We now turn our attention to the case of cross-validation. We formalize $N$-folds cross-validation as follows:

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

where for any $i$ and $k$, $X_k^i$ is a sample of $n$ points.

Greedy cross-validation could be considered as in the case of hold-out above: all $X_k^i$ could be independent. This leads to the same result (for some distribution, $k^*(n) \to \infty$) with roughly the same proof. We therefore only consider cross-validation with pairing, i.e. $\forall i, k, k',\ X_k^i = X_{k'}^i$. For short, we note $X_k^i = X^i$.

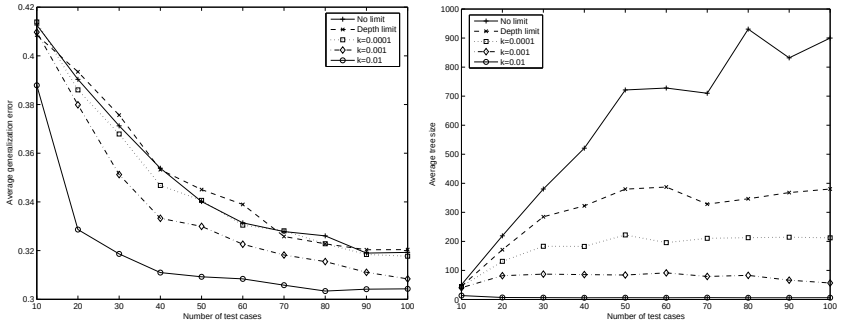
**Theorem 6 (Negative result on subsampling).** *Assume that $F_k$ has a VC-dimension going to $\infty$ as $k \to \infty$. One can not avoid bloat with only hold-out or cross-validation, in the sense that with paired hold-out, or greedy hold-out, or cross-validation, for any $V$, there exists some distribution for which almost surely, $k^*(n) > V$ infinitely often whereas an optimal function lies in $F_0$.*

Note that propositions above show in some cases stronger forms of bloat. If we consider greedy hold-out, hold out with pairing and cross-validation with pairing, then: (1) For some well-chosen distribution of examples, greedy hold-out almost surely leads to (i) $k^*(n) \to \infty$ if $N \to \infty$ (ii) $k^*(n)$ asymptotically uniformly distributed in $[[0, N]]$ if $N$ finite, whereas an optimal function lies in $F_0$ (theorem 5). (2) Whatever may be $V = VCdim(F_v)$, for some well-chosen distribution, hold-out with pairing almost surely leads to $k^*(n) > V$ infinitely often whereas an optimal function lies in $F_0$ (proposition 2). (3) Whatever may be $V = VCdim(F_v)$, for some well-chosen distribution, cross-validation with pairing almost surely leads to $k^*(n) > V$ infinitely often whereas an optimal function lies in $F_0$.

## 6 Experimental Results

Some theoretical elements presented in Sections 3 and 4 are verified experimentally in this section. The experimentation are conducted using Koza-style GP [10], with a problem setup similar to the classical symbolic regression example, modified for binary classification. This is covered by theoretical results above. The GP branches used are the addition, subtraction, multiplication, protected division, and if-less-than. This last branch takes four arguments, returning the third argument if the first argument is less than the second one, otherwise returning the fourth argument. The GP terminals are the $x$ variable, and the 0 and 1 constants. The learning task consists in minimizing the error $e(i)$ between the desired output $y_i = \{-1, 1\}$ and the obtained output $\hat{y}_i$ of the tested GP tree for the $x_i$ input, as in the following: $e(i) = \max(1 - y_i\hat{y}_i, 0)$. The fitness measure used in the experiments consists in minimizing the sum of the errors to which is added a complexity factor that approximate the VC-dimension of the GP program: $f = \frac{1}{s}\sum_{i=1}^{s} e(i) + k\sqrt{\frac{t^2 \log_2(t)}{s}}$, where $t$ is the number of nodes of the GP program tested, $s$ is the number of test cases used for fitness evaluation, and $k$ is a trade-off weight in the composition of the complexity penalization relatively to the accuracy term. The $s$ test cases are distributed uniformly in $x_i \in [0, 1]$, with associated $y_i = \{-1, 1\}$. For $x_i < 0.4$, each $y_i$ are equal to 1 with probability 0.25 (so $y_i = -1$ with probability 0.75), for $x_i \in [0.4, 0.6[$, $y_i = 1$ with probability 0.5, and for $x_i \geq 0.6$, $y_i = 1$ with probability 0.75. Thus, the associated classifier with best generalization capabilities would return $y_i^* = -1$ for $x_i < 0.4$, $y_i^* = 1$ for $x_i \geq 0.6$ and a random output for $x_i \in [0.4, 0.6[$, with a minimal generalization error of 0.3. After the evolutions, each best-of-run classifier is thus evaluated by a fine sampling of the input space, with the generalization error evaluated as the difference between the output given by the

tested best-of-run classifier and the output obtained by a classifier with best generalization capabilities. Five types of GP evolutions have been tested: i) no limitation on the tree size (no depth limit and complexity trade-off $k = 0$), ii) depth limitation on the tree size of 17 levels (complexity trade-off $k = 0$), iii) soft complexity penalty in the fitness (complexity trade-off $k = 0.0001$), iv) medium complexity penalty in the fitness (complexity trade-off $k = 0.001$), and v) important complexity penalty in the fitness (complexity trade-off $k = 0.01$). For the three last approaches, the depth limitation of 17 levels is still maintained. The selection method used is lexicographic parsimony pressure [15], that is regular tournament selection 4 participants, with the smallest participant taken in case of ties. Other GP parameters are: population of 1000 individuals; evolutions on 200 generations; crossover probability of 0.8; subtree, swap and shrink mutation of probability 0.05 each; and finally half-and-half initialization with maximal depth of 5. All the experiments have been implemented using the GP facilities of the Open BEAGLE (`http://beagle.gel.ulaval.ca`, [8]) C++ framework for evolutionary computations. The experiments have been conducted different number of test cases varying from $s = 10$ to $s = 100$ by steps of 10. One hundred evolutions is done for each combinations of approaches tested and number of test cases, for a total of 50 000 evolutions. Figure 1 shows the average generalization errors and tree size obtained for the different approaches in function of the number of test cases used for fitness evaluation. These results show that bloat occurs




(a) (b)

**Fig. 1.** Generalization errors and tree sizes observed for different size limitations. Figure (a) shows the average generalization errors observed, with apparently better results for the approaches where the fitness includes some parsimony pressure. Figure (b) shows the average tree sizes obtained, where important bloat is observed for the no limitation and maximum depth limitations

when no limitation of size occurs, even when lexicographic parsimony pressure is used (see curve *No limit* of Figure 1b), which validates Theorem 3. Then, as stated by Theorem 2, UC is achieved using moderate complexity penalization

in the fitness measure, with a convergence toward optimal generalization error of 0.3 (see curve *k=0.001* of Figure 1a). Third, as predicted by Theorem 4, increasing the penalization leads to both UC and no bloat (see curve *k=0.01* of both Figures 1a and 1b). Note that Theorem 3 asserts that this result cannot be achieved by *a priori* scaling of the complexity, and that Section 5 shows that this can not be achieved by cross-validation.

## 7 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 main limit of our work is that it deals only with the statistical elements of genetic programming (effect of noise) and not with the dynamics (the effect of bounded computational power). 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 with any of the classical forms of regularizations (from learning theory): the method of Sieves, or Structural Risk Minimization. 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), i.e. by combining Structural Risk Minimization and Sieves. An important point is that all methods leading to no-bloat use a regularization term; in particular, cross-validation or hold-out methods do not reach no-bloat.

**Acknowledgements** This work was supported in part by the PASCAL Network of Excellence, and by postdoctoral fellowships from the ERCIM (Europe) and the FQRNT (Québec) to C. Gagné. We thank Bill Langdon for very helpful comments.

## References

1. W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming : an introduction.* Morgan Kaufmann Publisher Inc., San Francisco, CA, USA, 1998.
2. Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.
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, R. R. Bertram, S. A. Stanhope, J. C. Khoo, S. A. Chaudhary, O. A. Chaudhri, and J. A. Ii Polito. 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. Edwin D. De Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pages 11–18, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
6. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1997.
7. A. Ekart and S. Nemeth. Maintaining the diversity of genetic programs. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, pages 162–171, London, UK, 2002. Springer-Verlag.
8. C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, April 2006.
9. S. Gustafson, A. Ekart, E. Burke, and G. Kendall. Problem difficulty and code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 4(3):271–290, 2004.
10. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
11. W. B. Langdon. The evolution of size in variable length representations. In *IEEE International Congress on Evolutionary Computations (ICEC 1998)*, pages 633–638. IEEE Press, 1998.
12. W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming And Evolvable Machines*, 1(1/2):95–119, 2000.
13. W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In *Late Breaking Papers at GP'97*, pages 132–140. Stanford Bookstore, 1997.
14. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In *Advances in Genetic Programming III*, pages 163–190. MIT Press, 1999.
15. S. Luke and L. Panait. Lexicographic parsimony pressure. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann Publishers, 2002.
16. N. F. McPhee and J. D. Miller. Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.
17. P. Nordin and W. Banzhaf. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.
18. A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In *Artificial Evolution VI*. Springer Verlag, 2001.
19. S. Silva and J. Almeida. Dynamic maximum tree depth : A simple technique for avoiding bloat in tree-based GP. In *Genetic and Evolutionary Computation – GECCO-2003*, LNCS, pages 1776–1787. Springer-Verlag, 2003.
20. Sara Silva and Ernesto Costa. Dynamic limits for bloat control: Variations on size and depth. In *GECCO (2)*, pages 666–677, 2004.
21. T. Soule. Exons and code growth in genetic programming. In *European Conference on Genetic Programming (EuroGP 2002)*, volume 2278 of *LNCS*, pages 142–151. Springer-Verlag, 2002.

22. 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.
23. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
24. B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1), 1995.