# Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics

Alain Ratle, Michèle Sebag

# Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics

Alain Ratle [a,*], Michèle Sebag [b]

[a] *Institut Supérieur de l'Automobile et des Transports, 49 rue Melle Bourgeois, BP 31-58027, Nevers Cedex, France*
[b] *Lab. de Mécanique des Solides, CNRS UMR 76-49, Ecole Polytechnique, 91128 Palaiseau, France*

**Abstract**

Although genetic programming has often successfully been applied to non-parametric modeling, it is frequently impaired by the huge size of the search space explored. Domain knowledge is a powerful way to trim out the size of the space, by restricting the search to a priori relevant models. A most natural domain knowledge in scientific modeling is known as dimensional analysis, stipulating that the models must be consistent with regards to the variable measurement units.

In this paper, it is shown that dimensional analysis can automatically be expressed as a context free grammar. Dimensionally-aware GP is thus achieved by employing the dimensional grammar within the grammar-guided GP framework first investigated by Gruau [On using syntactic constraints with genetic programming, in: P. Angeline, K.E. Kinnear Jr. (Eds.), Advances in Genetic Programming II, MIT Press, Cambridge, MA, 1996, pp. 377–394.].

However, grammar-guided genetic programming encounters severe difficulties when it involves a complex grammar, which might explain why this approach has not been widely used so far. The drawback is blamed on the initialization step, which hardly constructs a sufficiently diversified initial population, thus hindering the success of evolution. This limitation is addressed by a new CFG compliant initialization procedure.

The approach is validated on two problems related to the identification of mechanical properties of materials.

*Keywords:* Genetic programming; Context-free grammar; Grammatical evolution; Domain knowledge; Constrained genetic programming; Dimensional analysis; Machine discovery; Inverse problems; Identification in mechanics

## 1. Introduction

Scientific discovery consists of modeling a physical (mechanical, chemical, biological, etc.) phenomenon from the available observations and current theories. This paper is concerned with the automatic discovery of such empirical laws, referred to as *machine discovery* (MD). Machine discovery has primarily been investigated in the machine learning framework [1–5]. The machine learning approach used to make restrictive assumptions on the available data [1,2]; it still heavily requires the domain expert or the MDer's support, tuning a wide range of declarative heuristics [3–5]. This might make it questionable for machine discovery to scale up to high dimensional

* Corresponding author. Tel.: +33-86-71-5000;
fax: +33-86-71-5001.
*E-mail addresses:* alain.ratle_isat@u-bourgogne.fr (A. Ratle), michele.sebag@polytechnique.fr (M. Sebag).

problems, or achieve exploratory analysis in ill-known domains.

The discovery of empirical laws has also been tackled by a number of modeling (or identification) techniques, ranging from data analysis [6] to support vector machines [7]. Modeling techniques can be divided into parametric and non-parametric approaches. Parametric modeling is concerned with finding a fixed number of coefficients, whose interpretation is governed by a prescribed model. Non-parametric modeling, which will be the only one considered in the following, determines both the model and the coefficients thereof.

Major breakthroughs in non-parametric modeling have been achieved by genetic programming (GP) [8–10], which extends the principles of genetic algorithms and evolutionary computation [11,12] to structured (infinite) search spaces. Its stochastic search allows GP to explore practically and robustly, if not necessarily efficiently, huge search spaces. Indeed, many applications concerned with non-parametric modeling have been successfully tackled by GP [13–17].

However, GP suffers from a major limitation with respect to MD. Although the knowledge-based issues of evolutionary computation have been widely acknowledged [18,19], canonical GP offers no straightforward way of exploiting *domain knowledge*, namely the expert expectations or requirements regarding the sought model. In MD, a most trivial kind of domain knowledge is dimensional analysis; variables are typed depending on their measurement units (e.g. meter, second, kilogram, etc.), and admissible models are dimensionally consistent (meter and second should not be added). Dimensional consistency, or more generally domain knowledge, can thus be viewed as constraints on the model space. Such constraints are meant to drastically and soundly reduce the search space, thereby increasing the chances of success everything else being equal. Such a constraint can be enforced within GP by means of penalization on non-consistent models [20].

Still, the most natural way of dealing with constraints in optimization is, if possible, to get rid of them by re-designing the search space and/or the evolution initialization and operators [21]. Within GP and generalizing the strongly typed GP framework [22,23], Gruau shows that context-free grammars (CFGs) can be both used to declaratively specify the set of admissible models, and to enforce the production of admissible offspring from admissible parents [24]. The coupling of CFGs and GP has been studied by several authors (see [25–27] among others). The expressiveness of CFGs is demonstrated by showing how these can be used to enforce dimensional consistency; an automatic grammar generator implementing dimensional consistency, first described in [28], is presented.

Unfortunately, grammar-guided GP ($G^3P$) is observed to perform poorly when it involves a reasonably complex grammar (i.e. when the admissible search space is small, which is the case for dimensionally consistent solutions) [26]. We blamed this shortcoming on the initialization step, for the following reasons. On one hand, the standard sampling of the admissible search space might result in poorly diversified individuals. On the other hand, as emphasized by [29,30], the initial population quality can have a considerable impact on the overall result (up to several orders of magnitude). A new initialization procedure for $G^3P$, that produces admissible individuals of limited size that are *sufficiently diverse* was therefore proposed [28].

### *1.1. Applications*

In this paper, the main application concerns the identification of an analytic model of materials behavior from experimental data [37]. Such a task is essential for material scientists in order to characterize recently developed materials. One of the most common test for on-site applications is the indentation test described in Fig. 1. This test consists of pushing a hard indenter with a standardized shape against the material to be tested. The applied force is recorded together with the penetration depth of the indenter, and the result is a force–displacement curve, or a force–time curve for time dependent materials, as shown in Fig. 1.

The goal of the learning task is to find out an analytical model which can explain the physics lying behind these curves. It should be noted, however, that except for some classes of materials, this is an open problem which has no solution. Therefore, in order to develop the automatic learning tool in a better controlled environment, a simpler application is also considered. This second case consists of a visco-elastic model known as the Kelvin–Voigt model. The model is sketched in Fig. 2a, and consists of a spring $K$ in parallel with a dashpot $C$. When this model is

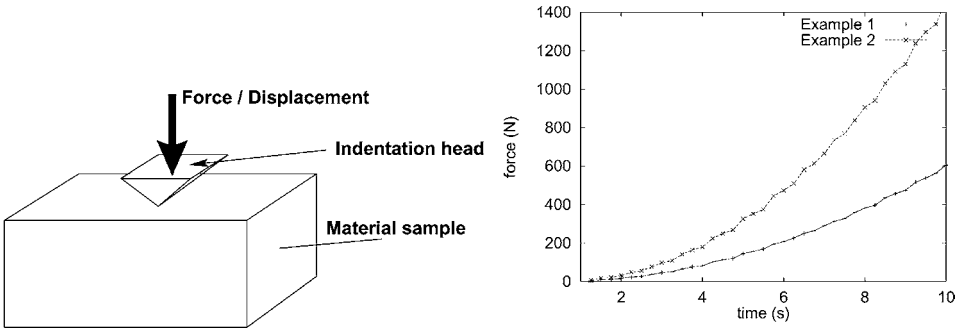


Fig. 1. Sketch of the experimental setup, and typical results.

submitted to a step-function force of amplitude $F$ from the time $t = 0$, the displacement $u$ is easily derived from the equation of motion of the system:

$$u(t) = \frac{F}{K}(1 - \mathrm{e}^{-Kt/C}) \tag{1}$$

A second, more complex case, is the four-element model sketched in Fig. 2b. This model has a response to a step function force given by:

$$x(t) = \frac{F}{K_1} + \frac{Ft}{C_1} + \frac{F}{K_2}(1 - \mathrm{e}^{-Kt/C_2}) \tag{2}$$

### *1.2. Organization*

This paper is organized as follows. First of all, Section 2 presents a brief overview of classical genetic programming. Section 3 reviews some related works and discusses how to take into account prior knowledge within GP. In order for this paper to be self contained, Section 4 briefly describes context-free grammars. Section 5 demonstrates how grammars can be used to automatically encode dimensional consistency. Section 6 details how to modify the GP components to accommodate a CFG, notably focusing on the initialization procedure. Last sections are devoted to the empirical validation of the presented G$^3$P scheme on problems inspired from real-world applications in mechanical modeling, the identification of materials behavioral law. Section 7 describes the test applications. Section 7.2 presents our empirical setting and discusses the experimental results. The paper ends with some perspectives for further research.

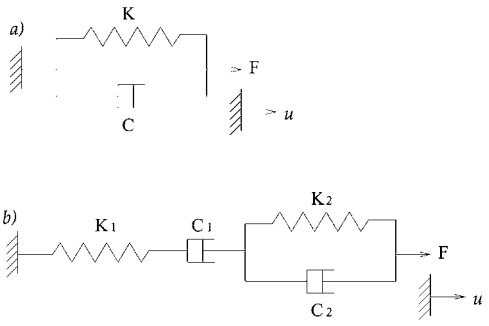


Fig. 2. The Kelvin–Voigt model, and the four-element model.

## 2. Elementary genetic programming

Genetic programming consists of a special class of evolutionary algorithms for which the evolution takes place over tree representations of computer programs rather than over linear genomes. The basic scheme of an evolutionary algorithm can be described as follows:

- Create a random population of solutions.
- Evaluate the fitness of each individual (solution).
- While the stopping criteria is not satisfied,
  - select the best parents in the current population according to fitness;
  - recombine them in order to generate offsprings;
  - mutate some of them in a random direction.
- End of the algorithm.

The tree representation, and the crossover (recombination) and mutation operators are illustrated by the following example, where a function of the three variables $a$, $b$ and $c$ is required using the four basic

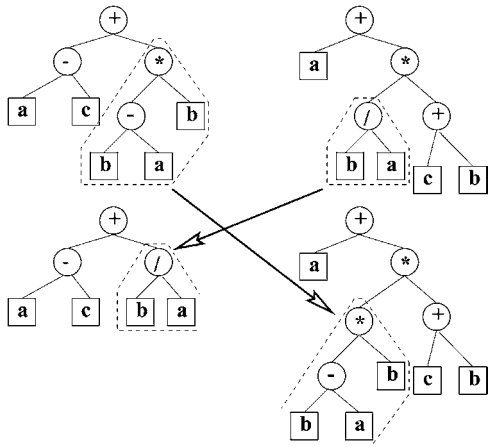

Fig. 3. Function representation and crossover operator of the basic GP algorithm.

arithmetic operators ($+$, $-$, $*$, $/$). Fig. 3 shows in the upper part two individuals equivalent to the functions:

$$f_1(a,b,c) = (a-c) + (b-a) * b \quad \text{and}$$
$$f_2(a,b,c) = a + \left(\frac{b}{a}\right) * (c+b) \tag{3}$$

The crossover operator consists of selecting two subtrees in two preselected parents, and swapping these subtrees in order to generate new solutions which contains the building blocks of the "good" parents. Fig. 3 shows in the bottom part the two offsprings obtained by the recombination of the two parents. These offsprings correspond to the following functions:

$$f_3(a,b,c) = (a-c) + \left(\frac{b}{a}\right) \quad \text{and}$$
$$f_4(a,b,c) = a + ((b-a) * b) * (c+b) \tag{4}$$

The mutation operator of the basic GP algorithm consists in applying the crossover operator between a selected parent and a randomly generated tree. In the context of genetic programming for symbolic regression, or function identification, the fitness function is based on the distance between the solution given by the algorithm and the experimental results to be learned. In the present application, the quadratic distance between the experimental curves and the learned functions was employed. This fitness function should, of course, be minimized by the algorithm.

## 3. Evolutionary non-parametric modeling

The branch of evolutionary computation concerned with non-parametric modeling is genetic programming [8–10]. Genetic programming extends the principles of genetic algorithms to tree-structured spaces, described from a set of operators or nodes $\mathcal{N}$, and a set of terminals or leaves $\mathcal{T}$. As trees of any depth can be generated, evolution truly explores an unbounded search space. [1] As an example, consider the tree-structured space defined from nodes plus and multiply and leaves $x$ (variable) and $\mathcal{R}$ (any real-valued constant). This space includes polynomials of variable $x$ of any degree.

Canonical GP relies on the hypothesis of closure of the search space [8], which assumes that the return value of any subtree is a valid argument for any function. This ensures that simple crossover and mutation (respectively swapping sub-trees and replacing an arbitrary (part of) subtree by a random one) shall produce admissible offsprings. On the other hand, this assumption forbids any syntactic or semantic restrictions to be done. Some simple restrictions can be handled through the operator design, e.g. a protected division is designed to handle the case of a null divisor. All other restrictions must be accounted for through the fitness function design, e.g. penalizing non-viable individuals. The closure assumption thus implies strong limitations:

- The search space is huge, even for problems of moderate difficulty [31]: it is exponential with respect to the number of terminals and operators and the maximum tree depth allowed.
- No consideration is taken for the variable types (integers, real numbers, complex numbers, etc.) or dimensions (meter, second, kilogram, etc.), which implies that the search space includes a number of irrelevant individuals [32].
- The general shape of the generated trees is arbitrary; the expert prior knowledge can only dictate the operator set.

In summary, the price to pay for simple GP evolution operators is that the search space is much larger

[1] Practically, an upper bound is set on the maximum tree depth, which might raise difficulties during the initialization step; this point is discussed in more detail in Section 6.

than the set of relevant individuals. GP implicitly deals with a constrained optimization problem, where the constraints reflect the domain expert or GPer prior knowledge. The importance for evolutionary computation to take the expert knowledge into account is now generally acknowledged [18,19], and several authors have addressed the coding of expert knowledge through GP biases.

A first class of bias concerns the shape of the sought solution. A significant improvement in the success rate of a GP application can be obtained by biasing the shape of the parse trees toward some shapes judged interesting [31,33]. This can be done through setting or adapting the selection probabilities of the operators [33], or via syntactic constraints [31].

Actually, most GP biases are expressed as syntactic constraints. Koza used syntactic constraints (e.g. setting the root operator or restricting the crossover according to the operator types) to either enforce the production of viable individuals or improve GP efficiency in his early work [8]. Automatically defined functions (ADF) can also be viewed as particular kinds of syntactic constraints, controlling the operators hierarchy in order to enforce GP scalability [9].

Syntactic constraints might also regard the types of the variables manipulated by the tree expression. These constraints might express the operator restrictions (e.g. the cosine operator should be applied on angular variables only) and the admissible combinations of operators. In the strongly typed GP (STGP) proposed by Montana [22] and extended by Haynes et al. [23], a type label is associated to every terminal, and every operator argument. The crossover and mutation procedures are then constrained accordingly; crossover operates by swapping a subtree with another subtree of the same type, and mutation replaces a subtree with a random subtree of the same type. Along the same lines, type constraints might reflect semantic restrictions, such as those related to dimensional analysis. A first step toward dimensionally aware GP was proposed by Keijzer and Babovic [20]. The dimensionality of each expression is encoded by a label, listing the exponents of the basic units. The requirements on the label of a subtree derive from those of its parent and sibling nodes.

As shown by Montana [22], it is relatively easy for crossover and mutation to accommodate syntactic constraints. But these might render the initialization step significantly more difficult. In Keijzer and Babovic's approach for instance [20], during initialization one has to generate a subtree of any prescribed label (compound unit); still, there does not necessarily exist a terminal associated to the desired label. Initialization thus calls an ad hoc function, `DimTransform`, which might introduce non-physically meaningful constructs hindering the physical relevance of the final tree. The fitness design hence takes care of this undesired side effect, by penalizing the individuals with many calls to `DimTransform`.

Next section describes a general framework for specifying or constraining the GP search space, namely context-free grammars [34]. How to accommodate these specifications within genetic programming is studied in Section 6.

## 4. Context-free grammars

As reminded by Gruau, syntactic constraints historically pertain to the theory of formal language and grammars; a grammar is meant to describe all admissible expressions of a language. In his pioneering work [24], Gruau showed how context-free grammars (CFGs) allow one to bias the GP search space through syntactic constraints, neatly expressing the admissible search space and significantly reducing the exploration effort. A publicly available GP software implementing the use of context-free grammars has been realized by Hörner [25].

In order for this paper to be self-contained, this section briefly reminds the CFG formalism. A standard way of characterizing a context-free grammar, known as Backus–Naur expression, is by a four-tuple $\{S, N, T, P\}$, where $S$ denotes the start symbol, $N$ the set of non-terminal symbols, $T$ the set of terminal symbols, and $P$ is the set of production rules. Fig. 4 shows the CFG describing the polynomials of variable $X$, to be compared with the standard GP description from the node set $\mathcal{N} = \{+, \times\}$ and terminal set $\mathcal{T} = \{x, \mathcal{R}\}$.

Any expression is built up from the start symbol $S$. For each non-terminal symbol (e.g. $\langle E \rangle$) there exists a production rule stating all possible ways of rewriting the non-terminal symbol, named *derivations* (e.g. the two possible derivations for $\langle E \rangle$ are $\langle O \rangle \langle E \rangle \langle E \rangle$ and $\langle V \rangle$). Each non-terminal symbol is rewritten by

$$
\begin{aligned}
N &= \{< E >,\ < O >,\ < V >\} \\
T &= \{+,\ \times,\ x,\ \mathcal{R}\ \} \qquad //\mathcal{R} \text{ stands for any real-valued constant} \\
P &= \left\{ \begin{array}{l} S \ := \ < E > \ ; \\ < E > := \ < O > \ < E > \ < E > \ \mid < V > \ ; \\ < O > := \ + \mid \times \ ; \\ < V > := \ x \mid \mathcal{R} \ ; \end{array} \right\}
\end{aligned}
$$

Fig. 4. The grammar for polynomials of any degree of variable $x$.

selecting a derivation until the expression contains terminals only. In evolutionary terms, one could consider the derivation tree as the genotype of the individual. The derivation tree gives rise to a parse tree that constitutes the phenotype and whose fitness is ultimately computed (Fig. 5).

The distinct roles of *non-terminals* and *terminals* in GP and CFG are shown from the derivation tree and the corresponding expression tree (Fig. 5). In both cases, non-terminal symbols correspond to the nodes of the tree, while terminal symbols are the leaves. But operators are *leaves* of the derivation tree (CFG terminal symbols), whereas they are *nodes* of the expression tree (GP non-terminal symbols). This difference implies that crossover and mutation will operate in different ways, depending on whether they apply on the derivation or the expression tree; this point will be discussed further in Section 6.1.

The advantage of CFGs is to allow for fine-grained restrictions on the combinations of operators and variables. Assume for instance that the parent node of a plus node should only be a multiply node, and conversely. Such a restriction is written down by modifying the above CFG as follows (only modified items are shown):

$$N := \{\langle \text{add\_}E\rangle, \langle \text{mult\_}E\rangle, \langle O\rangle, \langle V\rangle\}$$

$$S := \langle \text{add\_}E\rangle | \langle \text{mult\_}E\rangle$$

$$\langle \text{add\_}E\rangle := (+\langle \text{mult\_}E\rangle\langle \text{mult\_}E\rangle) | \langle V\rangle$$

$$\langle \text{mult\_}E >:= (\times\langle \text{add\_}E\rangle\langle \text{add\_}E\rangle) | \langle V\rangle$$

Canonical GP cannot declaratively accommodate such restrictions, i.e. through the description of $\mathcal{N}$ and $\mathcal{T}$ only. Procedural modifications are required to avoid or discourage non-complying individuals, through either the initialization procedure and evolution operators, or the fitness design.

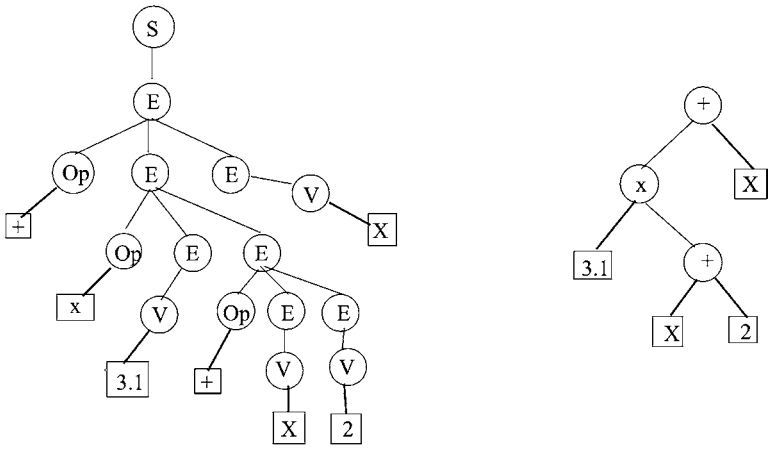


Fig. 5. Derivation tree and corresponding parse tree.

## 5. Dimensional analysis through CFGs

For our purpose, it remains to show that context-free grammars can be used to encode reasonably complex prior knowledge in the domain of machine discovery. The particular prior knowledge considered in the following is dimensional consistency, for two reasons. First of all, dimensional consistency constitutes a fairly general domain knowledge relevant to machine discovery. Secondly, it is very efficient in the sense that it drastically reduces the number of admissible solutions.

Two restrictions are done. The set of admissible compound units is finite, and only arithmetic operators (`plus`, `multiply` and `divide`) are applied on *non-dimensionless* expressions.

Under these restrictions, dimensional analysis can be expressed through a CFG [28].

Let $u_1, \ldots, u_K$ denote the $K$ elementary units of the problem domain. For instance, in the domain of macro-mechanical modeling the three elementary units are `meter`, `second` and `kilogram`. Borrowing to the formalism used in [20], any compound unit based on these $K$ elementary units is described as a vector of $\mathbb{R}^K$; e.g. the `Newton` unit (`kilogram` $\times$ `meter/second` $^2$) is represented as $(1, -2, 1)$. The set $D$ of allowed compound units is given as a finite subset of $\mathbb{R}^K$ (e.g. in the example domain, $K = 3$ and $D$ is $\{-2, -1, 0, 1, 2\}^3$).

To each unit $u$ in $D$ is associated a non-terminal symbol $\langle N_u \rangle$. The production rule associated to $\langle N_u \rangle$ describes all possible ways of constructing an expression of unit $u$:

1. by selecting a terminal symbol (problem variable) with unit $u$;

**Automatic Generation of Production Rules**

*# Production rule for $S$ states the unit of the sought solution*

$P = \{ S := < N_{u_s} > ; \}$

*# Construct the derivations for each unit $u$*

For each $u$ in $D$

*# Additive and subtractive derivations*

$exp_u = (< N_u > + < N_u >) \mid (< N_u > - < N_u >)$

For each $V$ in $\mathcal{V}_u$ *# Terminal derivations if any*

$exp_u = exp_u \mid V$

For each $v, w$ in $D$ such that $v +_\bullet w = u$ *# Multiplicative derivations*

$exp_u = exp_u \mid (< N_v > \times < N_w >)$

For each $v, w$ in $D$ such that $v -_\bullet w = u$ *# Divisive derivations*

$exp_u = exp_u \mid (< N_v > \div < N_w >)$

*# Log and exponential derivations for dimensionless expressions*

If $u = (0, ..0)$

$exp_u = exp_u \mid log(< N_u >) \mid e^{< N_u >}$

*# Add the production rule for $u$ to $P$*

$P = P \cup \{ < N_u > := exp_u ; \}$

Return $P$

Fig. 6. Generation of the dimensional grammar.

2. by adding or subtracting expressions with unit $u$;
3. by multiplying two expressions with respective units $v$ and $w$, such that $u = v +_\bullet w$, or
4. by dividing an expression of unit $v$ by an expression of unit $w$, such that $u = v -_\bullet w$, where $+_\bullet$ and $-_\bullet$, respectively, denote the vector addition and subtraction on $\mathbb{R}^K$.

Other possibilities, not considered in the following, would be by taking the square root of an expression of unit $2u$, or the cubic root of an expression of unit $3u$; but clearly, the use of fractional power operators cannot be extended beyond certain limits.

In the particular case of dimensionless expressions ($u = (0, \ldots, 0)$), any other operator (e.g. `exp`, `log`, etc.) can be used.

The number $|D|$ of allowed units is exponential in the number of exponents allowed for an elementary unit on average (e.g. $|D| = 5^3 = 125$ in our example), which makes it necessary to devise an automatic generator for the CFG. Note however, that although the grammar size is actually exponential, using it to enforce dimensional consistency entails no computational overhead compared to using some dimensionally-consistent crossover or mutation procedures.

The automatic CFG generator takes as input the unit of each terminal symbol, the set $D$ of admissible units, and the unit $u_s$ of the sought solution. The set of domain variables with unit $u$, possibly empty, is noted $\mathcal{V}_u$. The algorithm for automatically generating the production rules is depicted in Fig. 6.

Note that any additional cue on the shape of the solution might easily be given by the expert, through the production rule associated to the start symbol. For instance, if the sought model should involve an inverse exponential of the time variable $t$, the first production rule becomes:

$$S := \langle N_{u_s} \rangle \, \mathrm{e}^{-(\langle N_0 \rangle t)}$$

## 6. Grammar-guided genetic programming

CFGs allow one to declaratively characterize the GP search space. As the closure hypothesis does no longer hold, the standard GP components need be modified, giving rise to the grammar-guided GP (G$^3$P) framework. After briefly discussing the case of G$^3$P crossover and mutation, this section focuses on the initialization procedure.

### 6.1. G$^3$P crossover and mutation

CFG compliant crossover and mutation operators should produce admissible offspring from admissible parents. As mentioned earlier on, the required modifications are rather straightforward [22,24,25].

Within G$^3$P, genetic individuals are provided with two descriptions: the derivation tree, or genotype, encodes the expression tree, or phenotype. G$^3$P crossover and mutation apply on the genotypes, contrasting with canonical GP crossover and mutation applying on the phenotypes.

G$^3$P crossover selects with uniform probability a node in the first parent carrying any non-terminal symbol,[2] say $\langle N_u \rangle$. If the second parent has no node carrying symbol $\langle N_u \rangle$, the crossover is rejected. Otherwise, a node carrying symbol $\langle N_u \rangle$ is selected with uniform probability in the second parent, and the two subtrees rooted in the selected nodes are swapped.

G$^3$P mutation likewise selects with uniform probability a node carrying a non-terminal symbol $\langle N_u \rangle$, and replaces the subtree rooted from this node with any expression rewriting $\langle N_u \rangle$ (same as crossover with a random parent).

It is worth noting that handling the genotypes instead of the phenotypes remarkably modifies the crossover effects. As a proof, which is new to our best knowledge, note that phenotypic GP crossover (operating on the expression trees) cannot produce an offspring which differs from a parent by an operator only; phenotypic crossover can but modify a whole expression subtree. Quite the contrary, genotypic G$^3$P crossover (operating on the derivation trees) can access a leave of the derivation tree, hence modify an operator independently from its arguments (Fig 5).

The propensity of G$^3$P crossover to achieve such restricted modifications appears quite desirable. In opposition, the canonical GP setting requires the individuals to grow before canonical GP crossover could operate small modifications [8]; and such growth clearly congests the evolution process.

[2] Note that the crossover point must be a non-terminal; otherwise crossover can only be ineffective, swapping a terminal with itself.

$$
\begin{array}{lll}
N & = & \{< E >,\ < V >,\ < op >\} \\
T & = & \{\ +,\ -,\ \times,\ \div,\ a,\ b,\ c\ \} \\
S & := & < E >\ ; \\
< E > & := & < op >\ < E >\ < E > \mid < op >\ < E >\ < V > \mid \\
 & & \quad < op >\ < V >\ < E > \mid < op >\ < V >\ < V >\ ; \\
< V > & := & a \mid b \mid c\ ; \\
< op > & := & +\ \mid\ -\ \mid\ \times\ \mid\ \div\ ;
\end{array}
$$

Fig. 7. The grammar for polynomial fractions of $a$, $b$, and $c$.

### *6.2. CFG compliant initialization*

In the G$^3$P framework, initialization should both sample the search space as uniformly as possible, and respect the maximum tree-depth prescribed by the user.

However, in a reasonably complex CFG such as the dimensional one (Section 5), most non-terminal symbols cannot be resolved directly into a terminal. And in all cases, the fraction of terminal derivations is so small anyway that there is little chance for a uniform process to select a terminal derivation. Furthermore, in the general case the rewriting rules impose no limitations on the number of recursive calls to a non-terminal symbol.[3] A uniform initialization procedure thus tends to build up *very* deep trees, as noted by Ryan et al. [26], and oversized trees are massively rejected.

Similar difficulties are met in all constrained evolutionary schemes using a death penalty, when the admissible search space constitutes a small fraction of the search space [21].

A possibility would be to favor the selection of terminal variables whenever possible, e.g. through setting high probabilities on the corresponding derivations. Unfortunately, unless these probabilities are very carefully tuned, this leads to construct poorly diversified individuals, and evolution practically never recovers from such a poor initial population. As stressed by Daida [29], the importance of the initial population cannot be overestimated.

[3] Although one might upper bound the number of such recursions [24] this requires one to have fairly precise ideas on the shape of the solutions sought.

The heuristics first presented in [28] for overcoming the initialization difficulties within G$^3$P, proceeds by dynamically masking the derivations that would lead to oversized trees.

To each symbol $\langle N_u \rangle$ is associated an integer index $i(N_u)$ giving the depth of the smallest tree resolving $\langle N_u \rangle$ into terminal symbols; an integer index is similarly associated to each derivation 'deriv'. These indices are recursively computed from the following considerations:

- The index of a terminal variable is 1.
- The index of a non-terminal symbol is the minimum index of its derivations:

$$
\begin{aligned}
\langle N_u \rangle := \text{deriv}_1 | \ldots | \text{deriv}_K ; \Rightarrow i(N_u) \\
= \min_{k=1,\ldots,K} i(\text{deriv}_k)
\end{aligned}
$$

- The index of a derivation involving an operator is one plus the maximum index of its operands:

$$
i(\langle \text{op} \rangle \langle N_u \rangle \langle N_v \rangle) = 1 + \max(i(N_u), i(N_v))
$$

As an example, let us consider the CFG describing all polynomial fractions of variables $a$, $b$, and $c$ with rational coefficients (Fig 7).

The index of non-terminal symbol $V$ is 1. Hence, the index of derivation $(\langle \text{op} \rangle \langle V \rangle \langle V \rangle)$ is 2, and the index of non-terminal symbol $E$ is also 2.

Let $D_{\text{Max}}$ denote the maximum tree depth set by the expert. Obviously $D_{\text{Max}}$ must be greater than the index of the start symbol $S$ ($D_{\text{Max}} \geq i(S)$), otherwise no individual in the CFG space satisfies the depth limitation and the admissible space is empty. Along the same lines, any non-terminal symbol $N_u$ such that $i(N_u)$ is

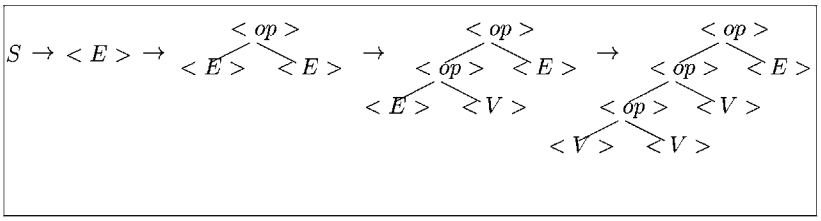


Fig. 8. Dynamically pruning the derivations.

greater than $D_{\text{Max}}$ is useless and should therefore removed from the CFG, together with all derivations involving them. For instance, if $D_{\text{Max}}$ were set to 2, all derivations of $E$ but the last one in the above example, should be removed.

Indices are employed to enforce the upper bound on tree size as follows. Consider a given occurrence of some non-terminal symbol $N_u$, situated at level $m$ of the tree under construction. The maximum depth allowed for rewriting this occurrence of $N_u$ is $D_{\text{Max}} - m$. Only those derivations of $N_u$ with index less than $D_{\text{Max}} - m$ are therefore allowed; and at least one such derivation must exist, since otherwise symbol $N_u$ would not have been allowed to use at level $m$. Other derivations are locally masked since they would lead to construct an oversized tree.

Practically, the derivation used to rewrite this occurrence of $N_u$ is uniformly selected among the admissible derivations. This way, it is impossible for the initialization algorithm to engage into a path providing no solution within the allowed tree depth: all constructed individuals are admissible with respect to both the CFG and the maximum tree-depth.

As an example, let $D_{\text{Max}}$ be set to 4, and see how the derivation tree develops. It starts with $S$, readily rewritten as $\langle E \rangle$. Assume that the first derivation $\langle \text{op} \rangle \langle E \rangle \langle E \rangle$ is chosen. Assume then that the leftmost $\langle E \rangle$ symbol is rewritten using the second derivation $\langle \text{op} \rangle \langle E \rangle \langle V \rangle$. At that point, the limitation on the tree depth imposes the leftmost $\langle E \rangle$ symbol to be rewritten using the shortest derivation $\langle \text{op} \rangle \langle V \rangle \langle V \rangle$ (Fig. 8).

The advantage of this procedure is that the initial population does not suffer from any loss of diversity; the set of admissible individuals is uniformly sampled, as the dynamic masking of the derivations does not introduce any unnecessary restrictions. [4]

## 7. Numerical experiments

In this section the G$^3$P is empirically validated on the artificial problems described in the introduction, and then is tested on the real application for which the solution is unknown. The goal is to identify the behavior laws of new materials, which has important applications especially for polymers and composite materials [36,37]. In the two test problems, a target model is analytically defined, and numerical examples are generated with random values of the materials parameters. Depending on the class of materials and the properties under study, the sought model either is a force–time relation, or a force–displacement relation.

In the test application 20 pseudo-materials have been numerically generated from random values of the material properties. The goal of the learning tool is to minimize the quadratic distance between these 20 simulated experiments and the proposed model. Therefore, a fitness value of zero is expected if the exact solution is found.

### *7.1. Grammar generation*

Three grammars are considered for the rheological models. For the second application (real problem),

[4] With regards to formal grammars, the presented heuristics corresponds to the use of an attribute grammar [35] with constraints. A depth attribute is attached to each symbol occurrence and each derivation rule is associated a constraint on the attributes. A simple constraint solver is used to check whether a given derivation path is currently allowed.

only the universal and dimensional grammars are considered.

UNIVERSAL grammar again specifies the same search space as canonical GP with operator set $\mathcal{N}$ and terminal set $\mathcal{T}$:

$$S := \langle E \rangle$$

$$\langle E \rangle := \langle O \rangle \langle E \rangle \langle E \rangle | \langle V \rangle$$

$$\langle O \rangle := + | - | \times | \div | \exp \equiv \mathcal{N}$$

$$\langle V \rangle := F \mid K \mid C \mid t \mid 1 \mid 2 \mid 3 \mid 4 \equiv \mathcal{T}$$

UNIVERSAL-EXN grammar, which expresses the fact that the exponent is necessarily negative (material parameters are always positive, and negative time is not considered). Within the universal grammar, evolution must thus create a combination of constants equal to $-1$, and multiply the result by another combination of variables. Instead, the search space can be biased toward physically plausible solutions by introducing the negative exponential operator `exn`, which returns the exponentiation of the negation of its argument. Universal-`exn` grammar differs from the universal one as operator `exn` is added to non-terminal $\langle O \rangle$.

DIMENSIONAL-EXN grammar, which takes advantage of the variable dimensions described on Table 1. This table allows the automatic generation of a dimensional grammar as described in Section 5. Operator `exn` is applicable on dimensionless expressions.

### *7.2. Experiment settings*

The empirical validation is concerned with evaluating how domain knowledge, provided through various grammars, can facilitate the evolution task. After describing the experiment setting, we present and discuss the results obtained by canonical GP and G$^3$P on the test applications.

Table 1
Physical units of measurement for the two rheological models

| Quantity | Mass | Length | Time |
|---|---|---|---|
| Domain variables | | | |
| Force, $F$ | $+1$ | $+1$ | $-2$ |
| Elastic element, $K$ | $+1$ | 0 | $-2$ |
| Viscous element, $C$ | $+1$ | 0 | $-1$ |
| Time, $t$ | 0 | 0 | $+1$ |
| Solution | | | |
| Displacement, $u$ | 0 | $+1$ | 0 |

Table 2
GP and G$^3$P experimental setting

| Parameter | Value |
|---|---|
| Population size | 4000 |
| Maximum number of generations | 500 |
| Probability of crossover | 0.8 |
| Probability of tree mutation | 0.2 |
| Probability of point mutation | 0.8 |
| Tournament size | 2 |
| Number of fitness cases | 320 |
| Number of independent runs | 20 |

Canonical GP and G$^3$P both follow a generational scheme with crossover, tree mutation and point mutation. Crossover swaps a randomly selected tree with another randomly selected tree (respectively derived from the same non-terminal symbol) in GP (respectively in G$^3$P). Tree mutation replaces a randomly selected subtree with a newly generated subtree (respectively derived from the same non-terminal symbol) in GP (respectively in G$^3$P). Point mutation replaces a randomly selected variable/constant with another one (respectively derived from the same non-terminal symbol) in GP (respectively in G$^3$P). The relevant parameters are summarized in Table 2.

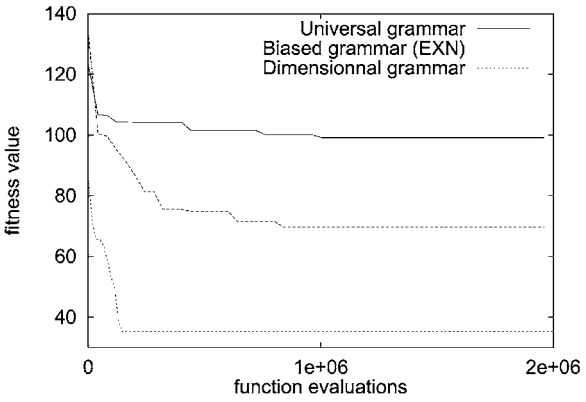


Fig. 9. Solution quality obtained with the three grammars for the Kelvin–Voigt problem.

Table 3
Comparative results on the Kelvin–Voigt problem (best result for 2,000,000 fitness evaluations, averaged on 20 runs)

| Grammar | Average fitness | S.D. |
|---|---|---|
| Universal grammar | 100.8 | 9.2 |
| Universal-exn grammar | 69.6 | 1.1 |
| Dimensional-exn grammar | 39.4 | 5.9 |

### *7.3. Results*

For both the Kelvin–Voigt and the four-element model, the exact solution was found in no run, whatever the grammar used. Fig. 9 shows the average over 20 runs of the best fitness against the total number of evaluations for all three grammars considered, for the most simple case, the Kelvin–Voigt problem. The baseline corresponds to the universal grammar (canonical GP); the second grammar differs from the universal one by the only addition of operator `exn`; the third one involves besides all dimensional constraints.

As could have been expected, the use of `exn` significantly improves the overall result. The use of dimensional constraints improves results even more significantly, as shown in Table 3.

In spite of the fact that the exact solution was never found, some of the approximate solutions were, from an engineering point of view, very similar to the data. Fig. 10 presents four random materials extracted from the training set (the dotted curves), together with the corresponding solutions found by the algorithm (the solid curves). Although these curves does not correspond exactly to the training examples, they are useful for material selection.

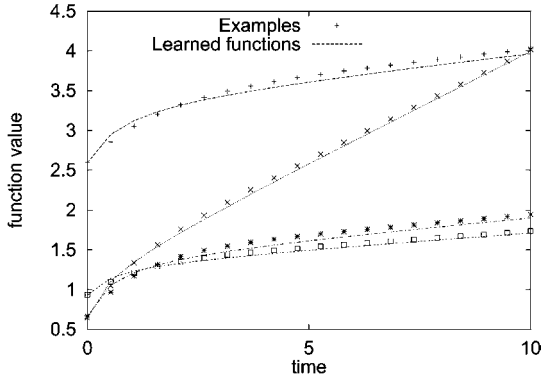


Fig. 10. Correlation between the training examples and the learned model for the four-element problem.

Finally, for the real application of unknown solution, similar results were found. No exact solution (zero fitness) was found, regardless of the parameters of the algorithm. Fig. 11 shows, however, that using a universal grammar, the best solution consists of a compromise lying in between all of the training samples. Using a dimensional grammar, the algorithm is able to find out a solution that matches more closely each of the samples.

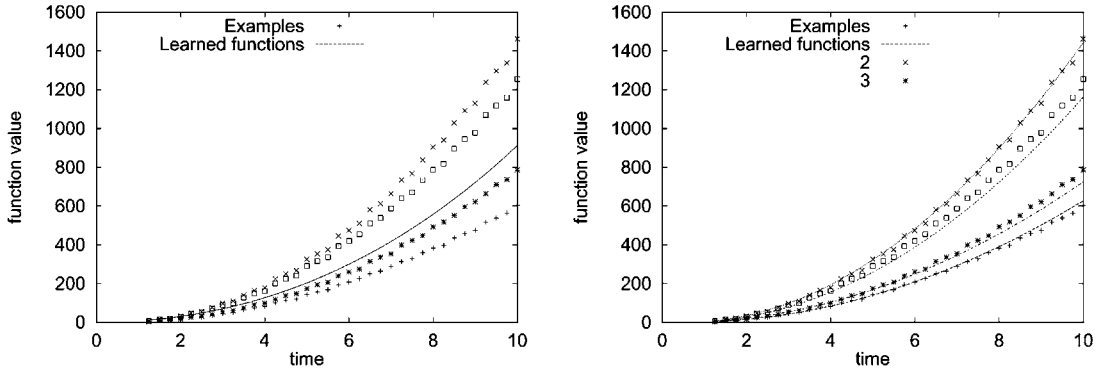


Fig. 11. Correlation between four samples of the real application and the best model found by the learning algorithm: universal grammar (left); dimensional grammar (right).

## 8. Conclusion and perspectives

This paper has investigated how to enhance genetic programming by taking into account prior knowledge related to the application domain.

After Gruau [24], the formalism of context-free grammars appears to be both efficient and manageable to express a wide range of syntactic constraints about the shape or properties of sought solutions. Grammar-guided GP, generalizing STGP [22], improves on canonical GP by restricting the search to admissible solutions, which represent a negligible fraction of the total search space.

The power of CF grammars has been demonstrated by showing how dimensional consistency can be encoded within such a grammar. Though the size of dimensional grammar is exponential in the number of elementary units in the domain, grammar-guided GP suffers from no computational overhead compared to typed GP. Compared to fitness-based ways of favoring dimensionally consistent solutions [20], the advantage of $G^3P$ is a drastic reduction of the GP search space. Besides, grammars also allow the expert to encode his/her priors regarding the shape of the target model.

One barrier to the wide use of $G^3P$ was identified as the initialization mechanism, for the maximum tree depth allowed for the trees usually conflicts with a great many constraints. This limitation has been addressed by a new heuristics, based on the dynamic masking of non-admissible constructs and derivations. This way, a sufficiently diversified initial population within the allowed maximal tree size might be created.

The advantage of the approach has been experimentally demonstrated on two test problems inspired from real-world applications. Indeed, the use of expert grammars is show to significantly improve the overall identification results.

The main limitation of $G^3P$ for dimensionally-aware GP is that a limited range of units is considered so far. Further research will investigate the use of fractional units together with a broader range of operators (e.g. square or cubic roots). This could be made possible by considering twice as many basic units (using rational instead of integer exponents).

Another difficulty for genetic programming regards the adjustment of real-valued parameters, which was not considered in our test problem. A perspective of research is to investigate how $G^3P$ could be used to restrict the range of values explored, according to the expert's indications.


## Acknowledgements

The authors acknowledge Helmut Hörner, who developed an efficient freeware for $G^3P$; this freeware, extended with the CFG compliant initialization procedure and the automatic generation of the dimensional CFG, served as a basis for our experiments. Numerical simulations data were provided by Nicolas Tardieu. We last acknowledge Andrei Constantinescu, Nicolas Tardieu, and Marc Schoenauer at Ecole Polytechnique, for many lively discussions.


## References


[1] P. Langley, H. Simon, G. Bradshaw, Rediscovering chemistry with the BACON system, in: R. Michalski, J. Carbonell, T. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, Los Altos, CA, 1983.

[2] B. Falkenhainer, R. Michalski, Integrating quantitative and qualitative discovery: the ABACUS system, Machine Learning 1 (4) (1986) 367–401.

[3] R. Valdes-Perez, Discovery tools for science applications, Communications of the ACM 42 (11) (1999) 37–41.

[4] S. Dzeroski, L. Todorovski, Discovering dynamics, in: Proceedings of the 10th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, CA, 1993, pp. 97–103.

[5] T. Washio, H. Motoda, Discovering admissible models of complex systems based on scale-types and identity constraints, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1997, pp. 810–817.

[6] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression by Tree, Wadsworth, Belmont, CA, 1984.

[7] V.N. Vapnik, The Nature of Statistical Learning, Springer, Berlin, 1995.

[8] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Evolution, MIT Press, Cambridge, MA, 1992.

[9] J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, Cambridge, MA, 1994.

[10] W. Banzhaf, P. Nordin, R. Keller, F. Francone, Genetic Programming — An Introduction on the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann, Los Altos, CA, 1998.

[11] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[12] T. Baäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1995.

[13] B. McKay, M. Willis, G. Barton, Using a tree structures genetic algorithm to perform symbolic regression, in: Proceedings of the IEEE Conference, Vol. 414, 1995, pp. 487–492.

[14] L. Spector, S. Luke, Cultural transmission of information in genetic programming, in: Proceedings of the Second International Conference on Genetic Programming (GP97), Morgan Kaufmann, Los Altos, CA, 1997, pp. 240–248.

[15] J. Duffy, J. Engle-Warnick, Using symbolic regression to infer strategies from experimental data, in: Evolutionary Computation in Economics and Finance, Springer, Berlin, 1999.

[16] T. Fernandez, M. Evett, Numeric mutation as an improvement to symbolic regression in genetic programming, in: Evolutionary Programming VII, Vol. LNCS 1447, Springer, Berlin, 1998, pp. 251–260.

[17] A. Salhi, H. Glaser, D. DeRoure, Parallel implementation of a genetic programming-based tool for symbolic regression, Information Processing Letters 66 (1998) 299-307.

[18] N.J. Radcliffe, Equivalence class analysis of genetic algorithms, Complex Systems 5 (1991) 183-920.

[19] C.Z. Janikow, A knowledge-intensive genetic algorithm for supervised learning, Machine Learning 13 (1993) 189-228.

[20] M. Keijzer, V. Babovic, Dimensionally aware genetic programming, in: Proceedings of the Genetic and Evolutionary Conference'99, Morgan Kaufmann, Los Altos, CA, 1999, pp. 1069–1076.

[21] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation 4 (1) (1996) 1–32.

[22] D.J. Montana, Strongly typed genetic programming, Evolutionary Computation 3 (2) (1995) 199–230.

[23] T. Haynes, D. Schoenefeld, R. Wainwright, Type inheritance in strongly typed genetic programming, in: P. Angeline, K.E. Kinnear Jr. (Eds.), Advances in Genetic Programming II, MIT Press, Cambridge, MA, 1996, pp. 359–375.

[24] F. Gruau, On using syntactic constraints with genetic programming, in: P. Angeline, K.E. Kinnear Jr. (Eds.), Advances in Genetic Programming II, MIT Press, Cambridge, MA, 1996, pp. 377–394.

[25] H. Hörner, A C++ class library for genetic programming, Technical Report, The Vienna University of Economics, 1996.

[26] C. Ryan, J. Collins, M. O'Neill, Grammatical evolution: evolving programs for an arbitrary language, in: W. Banzhaf, R. Poli, M. Schoenauer, T. Fogarty (Eds.), Proceedings of the First European Workshop on Genetic Programming (EuroGP98), Vol. LNCS 1391, Springer, Berlin, 1998, pp. 83–96.

[27] L. Martin, F. Moal, C. Vrain, Declarative expression of biases in genetic programming, in: Proceedings of the Genetic and Evolutionary Conference'99, Morgan Kaufmann, Los Altos, CA, 1999, pp. 401–408.

[28] A. Ratle, M. Sebag, Genetic programming and domain knowledge: beyond the limitations of grammar-guided machine discovery, in: M. Schoenauer et al. (Eds.), Proceedings of the Sixth Conference on Parallel Problems Solving from Nature, LNCS, Springer, Berlin, 2000, pp. 211–220.

[29] J. Daida, Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic, in: Proceedings of the Genetic and Evolutionary Conference'99, Morgan Kaufmann, Los Altos, CA, 1999, pp. 1069–1076.

[30] L. Kallel, M. Schoenauer, Alternative random initialization in genetic algorithms, in: T. Baäck (Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1997, pp. 268–275.

[31] P. Whigham, Inductive bias and genetic programming, in: Proceedings of the IEEE Conference, Vol. 414, 1995, pp. 461–466.

[32] C. Clack, T. Yu, Performance enhanced genetic programming, in: P. Angeline, R. Reynolds, J. McDonnell, R. Eberhart (Eds.), Evolutionary Programming VI, Springer, Berlin, 1997, pp. 87–100.

[33] R. Salustowicz, J. Schmidhuber, Evolving structured programs with hierarchical instructions and skip nodes, in: Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, CA, 1998, pp. 488–496.

[34] D.E. Knuth, The semantics of context-free languages, Mathematical System Theory 2 (2) (1968) 127–145.

[35] D. Schmidt, Denotational Semantics, Allyn & Bacon, Newton, MA, 1986.

[36] I. Ward, Mechanical Properties of Solid Polymers, Wiley, Chichester, 1985.

[37] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, H. Maitournam, Evolutionary identification of macro-mechanical models, in: P.J. Angeline, J.K.E. Kinnear (Eds.), Advances in Genetic Programming II, MIT Press, Cambridge, MA, 1996, pp. 467–488.