Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery

Alain Ratle and Michèle Sebag

LMS - CNRS UMR 7649, Ecole Polytechnique, 91128 Palaiseau Cedex

Abstract. Application of Genetic Programming to the discovery of empirical laws is often impaired by the huge size of the domains involved. In physical applications, dimensional analysis is a powerful way to trim out the size of these spaces This paper presents a way of enforcing dimensional constraints through formal grammars in the GP framework. As one major limitation for grammar-guided GP comes from the initialization procedure (how to find admissible *and sufficiently diverse* trees with a limited depth), an initialization procedure based on dynamic grammar pruning is proposed. The approach is validated on the problem of identification of a materials response to a mechanical test.

1 Introduction

This paper investigates the use of Genetic Programming [Koz92] for *Machine Discovery* (MD), the automatic discovery of empirical laws. In the classical Machine Learning framework introduced in the seminal work of Langley [LSB83], MD systems are based on inductive heuristics combined with systematic exploration of the search space. This approach suffers from severe limitations with real-world problems, due to ill-conditioned data and huge search spaces.

Such limitations are avoided in Genetic Programming (GP) due to its stochastic search principle. The price to pay is that GP offers no direct way to incorporate expert knowledge, although the knowledge-based issues of Evolutionary Computation are now recognized [Jan93]. In this paper, the emphasis is put on a particular, albeit rather general, expertise: in all application domains, variables most often have physical dimensions that cannot be ignored, for example, mass and length can not be added together. The restriction of the search space to dimensionally admissible laws has been tackled by [WM97] in a Machine Learning framework and by [KB99], using dedicated GP operators. On the other hand, an elegant and promising way to encode domain-knowledge is by formal grammars [RCO98,Hör96]. One major difficulty with Grammar-Guided Genetic Programming (G_3P) lies in the initialization step, the importance of which cannot be overestimated [Dai99]. Finding admissible trees *within a maximum depth* might be challenging enough to result in poorly diversified populations.

This paper investigates the use of grammars to restrict the GP search space to dimensionally admissible laws. The next section briefly presents context-free grammars and Sect. 3 discusses some related works. Sect. 4 describes a class of grammars for dimensionally admissible expressions, Sect. 5 presents its use for generating the population, and finally, Sect. 6 reports on numerical experiments with G_3P for the identification of phenomenological laws in materials science.

2 Context-Free Grammars

A Backus-Naur form (BNF) grammar describes the admissible constructs of a language through a 4-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 the production rules. Any expression is built up from the start symbol. Production rules specifie how should the non-terminal symbols, e.g. $< expr >$, be rewritten into one of their derivations (e.g. $(< oper >< expr >< expr >)$ or $< var >$) until the expression contains terminal symbols only.

Example:
$$\boxed{\begin{array}{l} N = \{ < expr >, < oper >, < var > \} \\ T = \{ \ x \ , \ \mathcal{R} \ , \ + \ , \ * \ , \ (\ , \ ) \ \} \\ P = \{ \ S := < expr > \\ \qquad < expr > \ := \ (< oper >< expr >< expr >) \mid \ < var > \\ \qquad < oper > \ := \ + \mid * \\ \qquad < var > \ := \ x \mid \mathcal{R} \qquad\qquad\qquad\qquad \} \end{array}}$$

The above grammar describes all polynomials of the variable x ($\mathcal{R}$ is interpreted as any real-valued constant); hence it is equivalent to the GP search space with the node set $\mathcal{N} = \{+, *\}$ and terminal set [1] $\mathcal{T} = \{x, \mathcal{R}\}$. One advantage of grammars is to allow fine-grained constraints to be imposed on the search space. Assume for instance that for one particular application, the parent node of an additive node must be a multiplicative node only, and *vice versa*. This is enforced via grammars by describing two non-terminals, $< add - expr >$ and $< mult - expr >$, with the following production rules:

$$\begin{array}{rcl} < add - expr > & := & (\ + \ < mult - expr >< mult - expr >) \mid \ < var >; \\ < mult - expr > & := & (\ * \ < add - expr >< add - expr >) \mid \ < var >; \end{array}$$

In canonical GP, satisfying this constraint would require either to design a specific initialization procedure and evolution operators, or to filter out any non-complying individual.

3 GP and Grammars: Previous Works

Canonical GP relies on the hypothesis of closure of the search space [Koz92], which assumes that the return value of any subtree is a valid argument for any

[1] It should be clear that *non-terminals* and *terminals* have different meanings in canonical GP and in BNF grammars. In the former, terminals are the leaves (variables and constants), and non-terminals are the nodes (operators), while in the latter, terminals comprise both variables and operators, and non-terminals are expressions that must be resolved into terminals.

function. This ensures that simple crossover and mutation (respectively swapping sub-trees and replacing an arbitrary subtree by a random one) shall produce admissible offsprings. What is gained in procedural overhead is lost in expressiveness: neither syntactic nor semantic restrictions are accounted for, and prior knowledge can dictate nothing but the node set. This implies several limitations:

- The size of the search space is huge, even for problems of moderate difficulty [Whi95]: it is typically exponential with respect to the number of terminals and nodes and to the maximum depth.
- The general shape of the trees is arbitrary.
- No consideration is given to the types (integers, reals, complexes,...).
- Variables are assumed to be dimensionless.

Consequently, the use of canonical GP with typed or dimensioned variables implies the useless generation of a vast majority of irrelevant trees [CY97]. Several authors have addressed this problem using various kinds of bias. A first kind is provided by the expert through domain knowledge. The importance of taking this knowledge into account is now generally admitted [Jan93]. In an MD context, prior knowledge might concern the shape of the solution[2]. 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 that are a priori judged interesting. This can be enforced by syntactic constraints; their beneficial effect have been illustrated by Whigham [Whi95] for the 6-multiplexer problem.

The use of syntactic constraints with genetic programming have been suggested as a potential form of bias by Koza [Koz92] in 1992. More formally, Gruau [Gru96] has shown that syntactic constraints can be used for reducing the size of the search space by allowing only type-consistent parse trees. However, a major limitation of Gruau's approach is that no limitation is put on the depth of the trees. This usually results in a severe growth in tree size.

A second kind of bias consists of constraining the types of the variables manipulated by the tree expression. These constraints might be related to the adequacy of the variables and operators (e.g. don't take the square root of a negative value), or to the physical dimensionality of the variables. A first step toward *dimensionally aware GP* was proposed by Keijzer and Babovic [KB99]. The dimensionality of each expression is encoded by a label which consists of a vector of the exponents of the basic units. For example, a variable expressed in Newtons ($kg \times m/s^2$) is labeled $[1, 1, -2]$. The requirements on the label of a subtree is defined from its parent and sibling nodes. This implies that the initialization procedure may have to construct a subtree with any label (compound unit). A `DimTransform` function is defined and produces a terminal of the required units. Since no terminal exists for each possible unit, `DimTransform` might introduce non-physically meaningful constructs, precluding the physical relevance of the final tree. Therefore, an auxiliary fitness measure is introduced in order to favor trees with few calls to `DimTransform`.

[2] For instance, in the example presented below, the force recorded during an experiment with the parameters $\mathbf{x}$ has the form $\mathcal{F}(\mathbf{x}, t) = f_1(\mathbf{x}) \times e^{f_2(\mathbf{x})t}$

Type constraints are closely related to the strongly typed GP (STGP) proposed by Montana [Mon95] and extended by Haynes et al. [HSW96]. In STGP, a type label is associated to every terminal, argument, and return value. The initial population is created by restricting the random choices to terminals or functions having the appropriate type label. 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. STGP does not address, however, the problem of the dimensional consistency of the expressions.

Formal grammars have been implemented in a GP system by Hörner [Hör96], with crossover and mutation using procedures that are similar to those of the STGP. However, Hörner's system suffered limitations from the difficulty of initializing valid parse trees, as was pointed out by Ryan [RCO98].

4 Dimensionalization through formal grammars

The new concepts presented in this paper are twofold. The first part consists of using grammar rules for incorporating dimensionality constraints into a GP framework. Second, the limitations of Grammar-Guided GP are broken down by a new initialization procedure based on a dynamic pruning of the grammar, in order to generate only feasible trees of prescribed derivation depth.

This approach is illustrated by a problem of mechanical behavior law identification. The elementary units involved are *mass, length* and *time*. The characterization of any compound unit as an $n-$tuple giving its exponent with respect to the elementary units is borrowed from [KB99]. The allowed compound units are specified by the user. The present study is restricted to integer powers of the basic units in the range $\{-2 \dots 2\}$. This excludes operators that returns fractional units (*e.g.*, the square root). The domain of allowed units therefore contains $5^3 = 125$ possible combinations. A non-terminal symbol is defined for each allowed compound units, together with the corresponding derivation rules to express all the admissible ways of resolving this symbol. Such a large number of combinations makes necessary the use of an automatic grammar generator. It might be objected that the size of this grammar makes it unpractical for real-world applications. Indeed, its memory complexity is exponential with respect to the number of elementary units, but no extra housekeeping is devoted to the GP kernel for units management. Therefore, the computational cost of this approach is no larger than other grammar-guided GP systems, and the use of a standard GP engine is allowed with no internal modifications. For instance, the results presented in this paper use Hörner's GP kernel as a basic engine [Hör96].

The grammar generator builds up each production rule with all the dimensionally coherent derivations. For example, a non-terminal with units $[i, j, k]$ can be replaced by the multiplication of two non-terminals with units $[a, b, c]$ and $[d, e, f]$ if and only if $[a, b, c] + [d, e, f] = [i, j, k]$. A similar rule applies to division, and addition/subtraction require both arguments to be of units $[i, j, k]$. Experts have to provide the derivation rule associated to the start symbol S, thereby encoding its expected units, and possibly the shape of the sought solution. The set of available variables and their units should also be provided. The procedure is described as follows:

Begin Grammar Generation
  For each combination of units $[i, j, k]$ **do**
    Create the production rule `<NTijk> :=` $expr_{ijk}$
    with $expr_{ijk} = +$ `<NTijk><NTijk>` | $-$ `<NTijk><NTijk>`
    $expr_{ijk} = expr_{ijk}$ | `<NTijk>` $\times \exp$ `<NT000>`
    For each variable/constant terminal $\mathcal{T}_\ell$ with units $[i, j, k]$
      $expr_{ijk} = expr_{ijk} \mid \mathcal{T}_\ell$
    For each pair of combinations of units $[a, b, c]$, $[d, e, f]$ **do**
      If $[a, b, c] + [d, e, f] = [i, j, k]$
        $expr_{ijk} = expr_{ijk} \mid \times$ `<NTabc><NTdef>`
      If $[a, b, c] - [d, e, f] = [i, j, k]$
        $expr_{ijk} = expr_{ijk} \mid \div$ `<NTabc><NTdef>`
    End for each pair $[a, b, c]$, $[d, e, f]$
  End for each $[i, j, k]$
End procedure

5 Initialization of Bounded Depth Trees

The initialization procedure has to build up trees based on the provided grammar. A major difficulty arises with the dimensioned grammar since most derivation rules can not be resolved directly into a terminal. The fraction of terminal derivations can be so small that there is almost no chance for a random process to select a terminal symbol. This implies, as noted by Ryan [RCO98], that the trees tend to be *very* deep. On the other hand, if the user specifies a maximum tree depth, the initialization proceeds by massively rejecting oversized trees. The problem is similar to what occurs in constrained optimization whenever the feasible region is very small.

Some mechanisms for controlling the derivation depth must therefore be incorporated in the initialization procedure. The proposed approach is intended to bound the initialization operator to the domain of dimensionally-feasible trees of depth equal or inferior to a prescribed value D_{max}. During grammar generation, to each non-terminal symbol `<NT>` is associated an integer $d($`<NT>`$)$, giving the depth of the smallest tree needed to rewrite `<NT>` into terminal symbols. The depth associated to each terminal symbol (operators, variables and constants) is set to 1. The depth of each `<NT>`, initially set to infinity, is recursively computed according to the following relations:

$$\begin{aligned} d(\texttt{<OP><NTa><NTb>}) &= 1 + \max\left(d(\texttt{<NTa>}), d(\texttt{<NTb>})\right) \\ d(\texttt{<NT}i\texttt{>}) &= \min_j \{d(deriv_j)\} \text{ for } \texttt{<NT}i\texttt{>} = deriv_1 | deriv_2 | \ldots | deriv_n; \end{aligned} \tag{1}$$

During the tree-generation phase, depth labels are employed in order to enforce the bound on tree size. Given a non-terminal node at a depth D in a tree, and assuming a maximum tree depth of D_{max}, the remaining allowed depth $D_{max} - D$ is computed. The chosen derivation is randomly drawn among the subset of the derivations for which $d(\texttt{<NT>}) \leq D_{max} - D$. This way, it is impossible for the algorithm to engage into a path that has no fully terminal solution in less than $D_{max} - D$ steps, and by the way, all the generated trees are feasible.

6 Numerical Experiments

The test-case presented herein is a simplified real-world application where an algebraic law is expected to be found for modeling experimental data corresponding to the constitutive law of a material during an indentation test. Figure 1 presents a schematic view of the experimental setup. A hard indenter of a prescribed shape (usually conical or tetrahedral) is pressed against the surface of the material to be tested out. The experimenter records the reaction force F along time t and displacement u.

For simple constitutive laws, the analytical relations between force, displacement, and materials properties are well known [Joh87]. For complex constitutive laws, finite elements models allow one to simulate the material reaction force. However, this simulation is rather expensive (3 hours on an HP350 workstation). For ill-known materials, only experimental data are available. This pinpoints the need for a simple analytical model in the two latter cases.

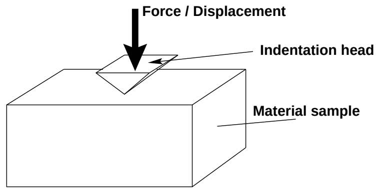

Fig. 1. Experimental setup of the indentation tests

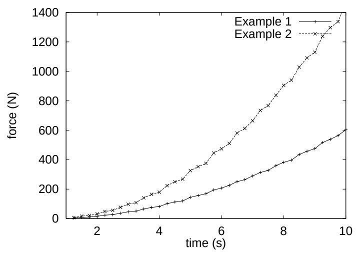

Fig. 2. Typical force vs time relations of numerical simulation

Examples have been generated according to random values of the material properties, and the material behavior has been computed with the finite element model. The examples are, by the way, noisy, due to the limitations of the numerical method: roundoff errors and modeling approximations. Typical results of simulated force–time relation are presented on Figure 2. From prior knowledge, this relation is expected, during the loading phase, to be of the form:

$$F(t) = Au^2 e^{(Pt)} \tag{2}$$

where A and P are unknown functions of the materials properties. The available physical quantities and their associated units are presented on Table 1. Due to the noisy nature of the examples, it is not expected that GP, nor any other machine discovery algorithm, will find out a solution that exactly fits the data.

Table 1. Physical units

Quantity	mass	length	time
E (Young's modulus)	+1	−1	−2
K (viscous modulus)	+1	−1	−1
n (plasticity factor)	0	0	0
S_y (yield strength)	+1	−1	−2
u (displacement)	0	+1	0
t (time)	0	0	+1
F (Indentation Force)	+1	+1	−2

Table 2. GP parameters

Parameter name	Value
Population size	4000
Max. number of generations	1000
Probability of Crossover	0.8
Probability of Tree Mutation	0.2
Probability of Point Mutation	0.8
Number of training examples	20
Number of independent runs	20

Machine Discovery experiments have been conducted with the GP parameters given on Table 2. The crossover consists of swapping two arbitrary subtrees from two parents, with a choice restricted to subtrees having a root node of the same type. Tree mutation consists of crossing over one individual with a random admissible tree. The point mutation replaces one terminal node by another terminal of the same type. This operator is analogous to a local improvement operator, and has been observed, for the present problem, to be less destructive than the tree mutation. Six grammars were devised and are described as follows:

1. **universal-non-dim:** The most general case, with no a priori knowledge or dimensional constraints. This grammar is equivalent to canonical GP:

   S := <NT>;
   <NT> := <OP> <TER> <TER> | <OP> <NT> <NT> |
   <OP> <TER> <NT> | <OP> <NT> <TER> ;
   <OP> := + | − | ∗ | ÷ | exp ; (=$\mathcal{N}$)
   <TER> := E| K | N | S_Y | u | t | 1 | 2 | 3 | 4; (=$\mathcal{T}$)

2. **[$\mathbf{Ae^{Pt}}$]-non-dim:** A partial constraint on the shape of the tree is introduced by this second grammar:

   S := exp <NT> <NT1> ;
   <NT1> := ∗ <NT> t ;
   <NT> := <OP> <TER> <TER> | <OP> <NT> <NT> |
   <OP> <TER> <NT> | <OP> <NT> <TER> ;
   <OP> := + | − | ∗ | ÷ ;
   <TER> := E| K | N | S_Y | u | t | 1.0 | 2.0 | 3.0 | 4.0 ;

   This grammar enforces two constraints on the search space: the highest-level (root) operator is necessarily an `exp` operator, and this `exp` operator is multiplied by an arbitrary expression (first argument) but exponentiates an expression multiplied by the time t.

3. $[\mathbf{Au^2e^{Pt}}]$**-non-dim:** The complete shape constraint $[Au^2 \exp(Pt)]$ is now enforced in a way similar to the previous case.
4. **universal-dim:** Dimensional constraints but no shape constraint. The solution is expressed in Newtons, so the start symbol is defined a priori as:

```
S := <NT+1+1-2>;
```

5. $[\mathbf{Ae^{Pt}}]$**-dim:** Dimensional constraints plus the partial shape constraint of the second grammar.
6. $[\mathbf{Au^2e^{Pt}}]$**-dim:** Dimensional constraints plus the complete shape constraint as in the third grammar.

Figure 3 presents the size of the search space computed as a function of the allowed derivation depth, with the universal grammar (case 1), and the dimensionally-constrained grammar (case 4). These curves show that in both cases, the number of solutions grows exponentially, but the search space can be reduced by several order of magnitude with the use of dimensional constraints.

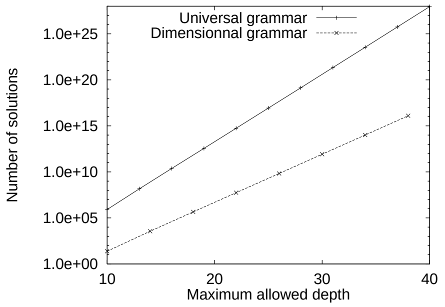

Fig. 3. Number of solutions in the search space with respect to the derivation depth, for non-dimensional (universal) and dimensional grammars.

Average best fitness value over 20 independent runs, and standard deviation are presented on Table 3, while the evolution of the average best fitness with respect to the number of evaluations is plotted on Fig. 4 for the non-dimensional grammars and on Fig. 5 for the dimensional grammars. Comparisons based on the number of evaluations are fair benchmarks since no significant variation in total computation time have been noticed between the grammars.

Figures 4 and 5 ask for two comments. First of all, giving the expected shape of the equation does not necessarily improve the results[3]. It partially does so in the case of non-dimensional grammars. But this might be due to the fact that the shape constraint prevents the search from being trapped in the same local

[3] Note that the problem at hand is based on real data where the solution is actually unknown.

optimum the universal grammar always falls in, which causes the null standard deviation observed for this case. This local optimum corresponds to the function $F = t^2 e^{2e^u}$. For the dimensional grammars, shape constraints are detrimental to the quality of the results in both cases. Second, the dimensional constraints appear to be clearly beneficial since the results obtained with dimensional grammars always supersede those obtained with untyped grammars, by an average of 6 standard deviations.

Table 3. Results

Grammar	Average fitness	Std. deviation
universal-untyped	6.2236E+4	0.0E+0
$[A \exp(Pt)]$-untyped	6.5762E+4	2.2E+3
$[Au^2 \exp(Pt)]$-untyped	5.1194E+4	1.9E+3
universal-dim	3.1009E+4	5.8E+3
$[A \exp(Pt)$-dim	4.0089E+4	2.7E+3
$[Au^2 \exp(Pt)$-dim	3.6357E+4	3.4E+3

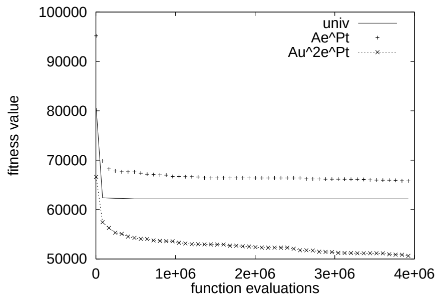

Fig. 4. Average best fitness for the three non-dimensional grammars

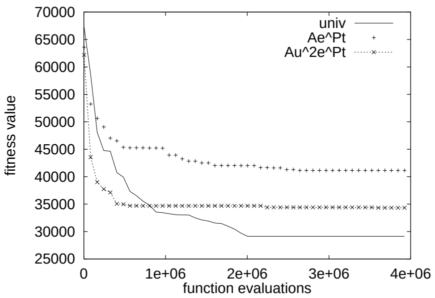

Fig. 5. Average best fitness for the three dimensional grammars

7 Conclusion

The innovations presented in this paper are twofold. First, a novel approach for the management of dimensionality constraints by the means of an automatic grammar has been presented. Second, the point of designing an admissible and still sufficiently diversified initial population has been addressed through dynamic pruning of the grammar, depending on the maximum tree depth allowed, and the current position in the tree. So far, the initialization step was a major limitation to the use of formal grammar for constraining a GP search space.

The main limitation of the presented approach is its dependence over a limited range of allowed units. Using fractional units can be made possible by the use of rational instead of integer numbers. This would allow the use of a broader range of operators (square root, powers,...), but would be equivalent to having twice as many basic units. Further research will be devoted to the simultaneous evolution of the grammar and the GP trees, in order to evolve grammars that facilitate the discovery of fitter individuals.

Acknowledgments

The authors acknowledge Helmut Hörner who developed an efficient basis for G_3P research. Finite elements simulations data were provided by Nicolas Tardieu.

References

[CY97] C. Clack and T. Yu. Performance enhanced genetic programming. In *Evolutionary Programming VI*, pages 87–100. Springer–Verlag, 1997.

[Dai99] J.M. Daida. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In *GECCO99*, pages 1069–1076. Morgan–Kaufmann, 1999.

[Gru96] F. Gruau. On using syntactic constraints with genetic programming. In *Advances in Genetic Programming II*, pages 377–394. MIT Press, 1996.

[Hör96] H. Hörner. A C++ class library for genetic programming. Technical report, The Vienna University of Economics, 1996.

[HSW96] T.D. Haynes, D.A. Schoenefeld, and R.L. Wainwright. Type inheritance in strongly typed genetic programming. In *Advances in Genetic Programming II*, pages 359–375. MIT Press, 1996.

[Jan93] C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.

[Joh87] K.L. Johnson. *Contact Mechanics.* Cambridge University Press, 1987.

[KB99] M. Keijzer and V. Babovic. Dimensionally aware genetic programming. In *GECCO99*, pages 1069–1076. Morgan–Kaufmann, 1999.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of natural Selection.* MIT Press, Cambridge MA, 1992.

[LSB83] P. Langley, H.A. Simon, and G.L. Bradshaw. Rediscovering chemistry with the Bacon system. In *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

[Mon95] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[RCO98] C. Ryan, J.J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *EuroGP98*, volume 1391 of *LNCS*, pages 83–96, 1998.

[Whi95] P.A. Whigham. Inductive bias and genetic programming. In *IEE Conf. publications, n. 414*, pages 461–466, 1995.

[WM97] T. Washio and H. Motoda. Discovering admissible models of complex systems based on scale-types and identity constraints. In *Intl. Joint Conf. on Artificial Intelligence 97*, pages 810–817, 1997.