A novel approach to Machine Discovery: Genetic Programming and Stochastic Grammars

Alain Ratle[1] and Michèle Sebag[2]

[1] LRMA- Institut Supérieur de l'Automobile et des Transports 58027 Nevers France
Alain.Ratle_isat@u-bourgogne.fr
[2] LRI CNRS UMR 8623, Université Paris Sud, 91405 Orsay France
Michele.Sebag@lri.fr

Abstract. The application of Genetic Programming (GP) to the discovery of empirical laws most often suffers from two limitations. The first one is the size of the search space; the second one is the growth of non-coding segments, the introns, which exhausts the memory resources as GP evolution proceeds.
These limitations are addressed by combining Genetic Programming and Stochastic Grammars. On one hand, grammars are used to represent prior knowledge; for instance, context-free grammars can be used to enforce the discovery of dimensionally consistent laws, thereby significantly restricting GP search space. On the other hand, in the spirit of distribution estimation algorithms, the grammar is enriched with derivation probabilities. By exploiting such probabilities, GP avoids the intron phenomenon.
The approach is illustrated on a real-world like problem, the identification of behavioral laws in Mechanics.

1 Introduction

Genetic Programming (GP) extends the principles of Genetic Algorithms and Evolutionary Computation [3] to optimization in tree-structured spaces [8, 2]. This paper is more particularly concerned with the use of GP for the automatic discovery of empirical laws. Although GP is widely used for symbolic regression [12, 5], it suffers from two main limitations. One first limitation is that canonical GP offers no tractable way to incorporate significant domain knowledge, despite the fact that the knowledge-based issues of Evolutionary Computation have been widely acknowledged [17, 7].

Significant advances in this respect have been done by Gruau [6] and Whigham [26], combining context free grammars (CFG) and GP to express and enforce syntactic constraints on the GP solutions. In a previous work [18], grammar-based GP was used to enforce the discovery of dimensionally consistent laws. Indeed, in virtually all physical applications, the domain variables are labeled with their dimensions (units of measurement), and the solution law must be consistent with respect to these units (seconds and meters should not be added). Dimensional consistency constitutes a widely available background knowledge;

still it allows for massive contractions of the GP search space.
However, grammar-based GP faces specific difficulties when dealing with complex grammars [19]; and dimensional grammar is much more complex than previous grammars used in combination with GP in the literature [6, 13, 26, 15]. An ad-hoc GP initialization procedure, based on attribute-like grammars, was therefore devised [18].

A second limitation of GP is that it requires huge amounts of computational resources, even when the search space is properly constrained. This is blamed on the bloat phenomenon, resulting from the growth of non-coding branches (*introns*) in the GP individuals [8, 9]. While its causes are not yet clearly understood, the bloat phenomenon adversely affects GP in two ways; on one hand, it might cause the early termination of the GP runs due to the exhaustion of available memory; on the other hand, it significantly increases the fitness computation cost.

In this paper a novel GP scheme addressing the bloat phenomenon is presented, which combines grammar-based GP and Distribution Estimation Algorithms [1, 16, 11]. Distribution estimation algorithms maintain a distribution over the search space (as opposed to maintaining a population of individuals, which standard EC does); in each step, a number of individuals is generated from scratch from the current distribution; these individuals then are evaluated according to the optimization goal, and the distribution is biased toward the best individuals.

The coupling of grammar-based GP and Distribution Estimation Algorithms most naturally considers stochastic grammars: the distribution is encoded through probabilities on the grammar derivations. This scheme called stochastic grammar-based GP, significantly differs from related works [20, 22, 25].

Stochastic grammar-based GP is validated experimentally on problems inspired from real-world applications in Solid Mechanics, the discovery of behavioral laws from experimental data (section 5).

The main result of the paper is that stochastic-grammar based GP does prevent the uncontrolled growth of introns, while efficiently identifying or approximating the target laws. This result, addressing one major limitation of GP, leads to a novel understanding of the bloat phenomenon.

2 Symbolic regression with Evolutionary Computation

This section summarizes the bases of our framework: Genetic Programming [8, 2], grammar-based GP [6, 26], how to handle complex grammars within GP [18], and Distribution Estimation Algorithms [1, 11].

2.1 Genetic Programming

We assume the reader's familiarity with the principles of Evolutionary Computation [3]. The fitness function $\mathcal{F}$ is defined from the search space Ω onto $\mathbb{R}$, and artificial evolution maintains a set of points of Ω, also called *population*, in order to reach the possibly many optima of $\mathcal{F}$.

$$\mathcal{F} : \Omega \mapsto \mathbb{R}$$

The initial population is most often built by uniformly sampling the search space. Thereafter, at each step or *generation*, the population is modified using selection (biased toward the best fit points or *individuals*), mutation (operator on Ω) and crossover (operator on Ω^2). Evolution thus proceeds until the stop criterion is met.

Genetic Programming, a novel branch in Evolutionary Computation, extends EC to structured search spaces, more particularly tree-structured spaces [8, 2]. Ω denotes thereafter the tree-structured space built from the set of nodes $\mathcal{N}$ and the set of leaves or terminals $\mathcal{T}$.

For instance, the set of polynoms of variables $X_1, X_2..$ can be represented as a tree-structured search space, where the nodes are the addition, subtraction and multiplication operators ($\mathcal{N} = \{+, -, \times\}$), and the leaves are the variables and real-valued constants ($\mathcal{T} = \{X_1, X_2, \ldots \mathcal{R}\}$, where $\mathcal{R}$ stands for any real-valued constant).

If the structure of the target polynom is known before hand, its identification boils down to optimization in $\mathbb{R}^n$ (finding the coefficients most attuned to the task at hand). The particularity of Genetic Programming is to tackle both the identification of the structure, and the coefficients of the target solution.

The individual representation, also known as genotypic search space, is tightly related to the so-called genotypic operators, i.e. initialization, crossover and mutation. GP crossover proceeds by swapping two randomly selected subtrees in the current individuals, or parents; the resulting trees are called offspring. Mutation proceeds by replacing one subtree in the parent, with one randomly generated subtree.

2.2 Grammar-based GP

A context free grammar describes the admissible constructs of a language by a 4-tuple $\{S, N, T, P\}$, where S is the start symbol, N the set of non-terminal symbols, T the set of terminal symbols, and P the production rules. Any expression is iteratively built up from the start symbol by rewriting non-terminal symbols into one of their derivations, as given by the production rules, until the expression contains terminals only. Table 1 shows the CFG describing the polynoms of variables X_1 and X_2, with $\mathcal{R}$ again standing for any valued constant. Grammars enable the expression of problem-specific constraints on the GP search space. These constraints are enforced as follows. On one hand, the recursive application of derivation rules can be characterized as a tree, known as *derivation tree* (Fig. 1); the derivation trees are equivalent to the expression trees, or *parse trees*.

Grammar-based GP manipulates derivation trees instead of parse trees. Crossover is restricted to swapping subtrees built on the same non-terminal symbol;

Table 1. Context Free Grammar for polynoms of variables X_1 and X_2

$$N = \{< E >,\ < Op >,\ < V >\} \qquad \textit{expression, operator, variable}$$
$$T = \{+,\ -,\ \times,\ X_1,\ X_2,\ \mathcal{R}\ \}$$
$$P = \begin{cases} S & := \ < E > \ ; \\ < E > & := \ < Op > \ < E > \ < E > \ \mid \ < V > \ ; \\ < Op > & := \ + \mid - \mid \times \ ; \\ < V > & := \ X_1 \mid X_2 \mid \mathcal{R} \ ; \end{cases}$$

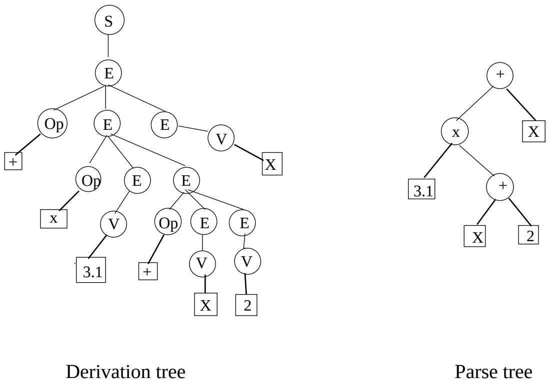


Fig. 1. Derivation tree and corresponding parse tree

mutation replaces a subtree by a new tree built on the same symbol [6, 26]. This ensures that CFG-compliant offspring are produced from CFG-compliant parents. These restrictions are quite similar to that of Strongly Typed GP [13].

2.3 Initialization in grammar-based GP

Our initial motivation for grammar-based GP is to enforce the discovery of laws dimensionally consistent wrt the application units, thereby both restricting the search space and increasing the expert's satisfaction.

We restricted ourselves to a finite number of compound units expressed wrt the elementary units; for instance the *Newton* unit is represented as vector $(1, 1, -2)$ wrt to elementary units (*kg, meter and second*). To each allowed unit is associated a non-terminal symbol. The corresponding production rule describes all ways of producing an expression of this type, e.g. a Newton-typed expression is obtained by adding two Newton-typed expressions, by multiplying a mass-typed expression with an acceleration-typed expression, and so forth. This grammar is automatically generated from the application units, and the range of allowed compound units; it describes all dimensionally consistent laws in the search space[3]. Although the CFG size is exponential, enforcing these restrictions linearly increases the crossover complexity in the worst case, and does not modify the mutation complexity.

[3] The production rule associated to the start symbol specifies the unit of the sought solution; it can also enforce the shape of the solution, according to the expert guess.

The dimensional grammar involves more than one hundred non-terminal symbols, and several thousands of derivations, which is considerably more than the grammars previously considered in the literature [6, 26, 15]. A severe limitation of grammar-based GP is then noticed [19]. The size of the GP trees normally is limited by setting a maximal tree depth D_{Max}. But expressions generated from the dimensional grammar through uniformly selecting derivation rules are *very* long (each non-terminal symbol is associated with many derivations, few of which are terminal). On one hand, these expressions are massively rejected because of their size, making the initialization step heavily computationally expensive. On the other hand, the expressions finally found in the initial population are poorly diversified, and evolution hardly ever recovers from a poor initial population [4].

A new initialization procedure was therefore designed for grammar-based GP [18], inspired from attribute-grammar. To each non-terminal symbol N is associated an integer value $\ell(N)$, giving the minimal depth of a parse tree rewriting this symbol. An integer value $\ell(d)$ is similarly associated to each derivation. These integers are recursively computed; $\ell(N)$ is set to the minimal $\ell(d)$ over all derivations d used to rewrite N; and $\ell(d)$ is computed from the $\ell(N)$s, over all symbols N occurring in the derivation.

If the minimal tree depth in the language is less than the maximal tree-depth allowed ($\ell(S) \leq D_{Max}$), then by recurrence for all non-terminal symbols in the current expression there exist admissible derivations, i.e. such that they lead to a tree within the tree-depth bound. By uniformly selecting one derivation among the admissible ones, parse trees complying with the grammar and the maximal tree-depth allowed, are produced. As the initial population so built is sufficiently diverse, evolution can work properly [18], until the intron growth is observed.

2.4 Distribution-based Evolution

Contrasting with genetic evolution, distribution-based evolution deals with a high-level (intentional) description of the best individuals encountered so far, as opposed to the (extensional) description given by the current population itself. This intentional description is a probability distribution $\mathcal{M}$ on the solution space. In each generation, the population is generated from $\mathcal{M}$ only, and the individuals are evaluated; thereafter, $\mathcal{M}$ is biased toward the best individual(s) in the current population, according to a given update mechanism.

As far as we know, the first algorithm resorting to distribution-based evolution is *Population-based Incremental Learning (PBIL)* [1], concerned with optimization in $\{0,1\}^n$. In this scheme, distribution $\mathcal{M}$ is represented as an element of $[0,1]^n$, initialized to $\mathcal{M}_0 = (.5, \ldots, .5)$.
At generation t, $\mathcal{M}_t$ is used to generate the population, the probability for any individual X to have its i-th bit set to 1 being the i-th coordinate of $\mathcal{M}_t$. The best individual X_{best} in the current population is used to update $\mathcal{M}_t$ by relaxation[4], with

$$\mathcal{M}_{t+1} = (1-\epsilon)\mathcal{M}_t + \epsilon X_{best}$$

[4] Other variants use the best two individuals, and possibly the worst one too, to update the distribution.

$\mathcal{M}_t$ is also randomly perturbed (mutated) to avoid premature convergence.

This scheme has been extended to accommodate different distribution models and non-binary search spaces (see [21, 11] among others).

3 Stochastic Grammar-based GP (SG-GP)

This section describes the extension of Distribution Estimation Algorithms to grammar-based GP, which involves the representation of the distribution, its exploitation in order to generate the current population, and its update from the current best (and worst) individuals.

3.1 Overview

Representation of the distribution. The distribution for grammar-based GP is most naturally represented as a stochastic grammar: each derivation rule d_i in the grammar is enriched with a weight w_i ($w_i \in \mathbb{R}^+$), and the selection probability of d_i depends on w_i as detailed below.

Generation of the population. The second component is how to exploit the current distribution in order to generate the population in each generation. It must be noted that this step is much similar to the initialization procedure in standard evolution. We accordingly modify the grammar-based initialization procedure (section 2.3), to accommodate for the derivation weights. Practically, all admissible derivations for the current non terminal symbol N are determined from the maximal tree-depth allowed and the position of N in the parse tree. One derivation among the admissible ones is selected according to their (normalized) weights, using a standard roulette mechanism. More formally, the probability p_i of selecting derivation d_i is defined as:

$$p_i = \begin{cases} \dfrac{w_i}{\sum\limits_{k \in \text{admissible derivs.}} w_k} & \text{if } d_i \text{ is an admissible derivation} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Updating the distribution. After all individuals in the current population have been evaluated, the probability distribution is updated from the N_b best and N_w worst individuals according to the following mechanism: for each derivation d_i,

- Let b denotes the number of individuals among the N_b best individuals that carry derivation d_i; weight w_i is multiplied by $(1+\epsilon)^b$;
- Let w denotes the number of individuals among the N_w worst individuals that carry derivation d_i; weight w_i is divided by $(1+\epsilon)^w$;
- Last, weight w_i is mutated with probability p_m; the mutation either multiplies or divides w_i by factor $(1+\epsilon_m)$.

All w_i are initialized to 1. Note that it does not make sense to have them normalized; they must be locally renormalized before use, depending on the current set of admissible derivations.

In summary, stochastic grammar based GP involves five parameters besides the three standard GP parameters (Table 2).

Table 2. Parameters of Stochastic Grammar-based Genetic Programming

Parameter	Definition
	Parameters specific to SG-GP
N_b	Number of best individuals for probability update
N_w	Number of worst individuals for probability update
ϵ	Learning rate
p_m	Probability of mutation
ϵ_m	Amplitude of mutation
	Canonical GP parameters
P	Population size
G	Maximum number of generations
D_{max}	Maximum derivation depth

3.2 Scalar and Vectorial SG-GP

In the above scheme, the genetic pool is represented by a vector $\mathcal{W}$, coding all derivation weights for all production rules. $\mathcal{W}$ belongs to $\mathbb{R}^{|d|}$, if $|d|$ denotes the total number of derivations in the grammar.
This way, the storage of a variable length population is replaced by the storage of a single fixed size vector; this is in sharp contrast with canonical GP, and more generally, with all evolutionary schemes dealing with variable size individuals.

One side effect of this representation is that it induces a total order on the derivations in a given production rule. However, it might happen that derivation d_i is more appropriate than d_j in upper levels of the GP trees, whereas d_j is more appropriate at the bottom of the trees.

To address this limitation, to each level of the GP trees (i ranging from 1 to D_{max}) is attached a distribution vector $\mathcal{W}_i$. This latter scheme is referred to as *Vectorial SG-GP*, as opposed to the former scheme of *Scalar SG-GP*.

The distribution update in *Vectorial SG-GP* is modified in a straightforward manner; the update of distribution $\mathcal{W}_i$ is only based on the derivations actually occurring at the i-th level among the best and worst individuals in the current population.

4 Experiment goal

Besides the finding accurate laws, our experiment goal is to gain some insights into the intron growth phenomenon.

It has often been observed experimentally that the proportion of introns in the GP material grows exponentially along evolution [14]. As already mentioned, the intron growth is undesirable as it drains out the memory resources, and increases the total fitness computation cost.

However, it was also observed that pruning the introns in each generation significantly decreases the overall GP performances [8]. Supposedly, introns protect good building blocks from the destructive effects of crossover; as the useful part of the genome is condensed into a small part of the individual, the probability for a crossover to break down useful sections is reduced by the apparition of introns.

Intron growth might also be explained from the structure of the search space [9]. Consider all genotypes (GP trees) coding a given phenotype (program). There exists a lower bound on the genotype size (the size of the shortest tree coding the program); but there exists no upper bound on the genotype size (a long genotype can always be made longer by the addition of introns). Since there are many more long genotypes than short ones, longer genotypes will be selected more often than shorter genotypes (everything else being equal, i.e. assuming that the genotypes are equally fit)[5].
Last, intron growth might also be a mechanical effect of evolution. GP crossover facilitates the production of larger and larger trees: on one hand, the offspring average size is equal to the parent average size; on the other hand, short size offspring usually are poorly fit; these remarks together explain why the individual size increases along evolution.

Since the information transmission in SG-GP radically differs from that in GP, as there exists no crossover in SG-GP, there should be no occasion for building longer individuals, and no necessity for protecting the individuals against destructive crossover.

Experiments with SG-GP are intended to assess the utility of introns. If the intron growth is beneficial *per se*, then either SG-GP will show able to produce introns — or the overall evolution results will be significantly degraded. If none of these eventualities is observed, this will suggest that the benefits of introns have been overestimated.

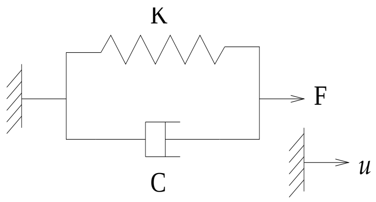
5 Experimental validation

This section presents the experimental validation of the SG-GP scheme on problems inspired from real-world applications, related to the identification of rheological models. These problems have important applications in the development of new materials, especially for polymers and composite materials [24].

[5] To prevent or reduce this intron growth, a parsimony pressure (regularization term) might be added to the fitness function [27]. In other words, the machine discovery objective is turned into a multi-objective optimization problem, finding an accurate law, and finding a simple law. As widely acknowledged, adjusting the relative weights of the two objectives is far from easy.

5.1 Test problem

The target empirical law corresponds to the Kelvin-Voigt model, which consists of a spring and a dashpot in parallel. When a constant force is applied, the response (displacement-time relation) follows the equation below is given below. Fitness cases (examples) are generated using random values of the material pa-



$$u(t) = \frac{F}{K}\left(1 - e^{\frac{-Kt}{C}}\right)$$

Fig. 2. The Kelvin-Voigt Model

rameters K and C and loading F. The physical units for the domain variables and for the solution are expressed with respect to the elementary *mass*, *time* and *length* units (Table 3). Compound units are restricted as the exponent for each elementary unit ranges in $\{-2, -1, 0, 1, 2\}$. The dimensional grammar is generated as described in section 2.3, with 125 non-terminal symbols and four operators (addition, multiplication, protected division and exponentiation). The grammar size is about 515 k.

Table 3. Physical Units in the Kelvin-Voigt Model

Physical units			
Quantity	mass	length	time
Variables			
F (Force)	+1	+1	−2
K (Elastic elements)	+1	0	−2
C (Viscous elements)	+1	0	−1
t (time)	0	0	+1
Solution			
u (displacement)	0	+1	0

5.2 Experimental setting

SG-GP is compared with standard elitist GP. The efficiency of SG-GP is assessed with respect to the quality of solutions, and in terms of memory use. All results are averaged on 20 independent runs.

GP and SG-GP parameters are set according to a few preliminary experiments (Table 4). Canonical GP is known to work better with large populations and small number of generations [8]. Quite the contrary, SG-GP works better with a small population size and many generations. In both cases, evolution is stopped after 2,000,000 fitness evaluations.

Table 4. Optimization parameters

SG-GP		GP	
Parameter	Value	Parameter	Value
Population size	500	Population size	2000
Max. number of generations	4000	Max. number of generations	1000
N_b	2	P(crossover)	0.9
N_w	2	P(mutation)	0.5
ϵ	0.001	Tournament size	3
P_m	0.001		
ϵ_m	0.01		

5.3 Experimental results

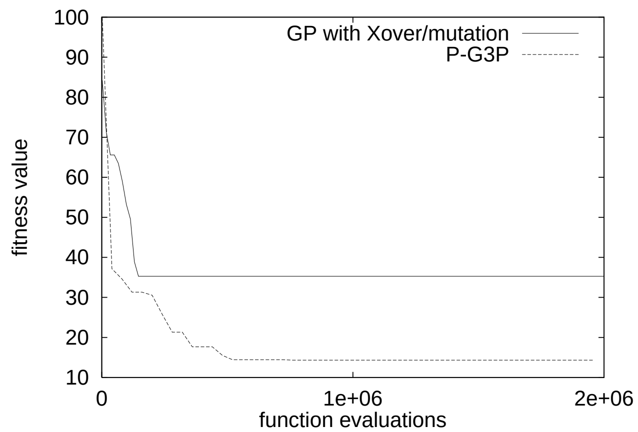


Fig. 3. Comparative performances of GP and Scalar stochastic grammar based GP (averaged on 20 runs)

Fig. 3 shows the comparative behaviors of canonical GP and scalar stochastic grammar-based GP on the test identification problems. The performance measures the average best fitness (mean squared distance over all 20 time steps, and 20 mechanical experiments), over 20 independent runs. The performances are significantly improved by using Vectorial SG-GP instead of Scalar SG-GP (Fig. 4). More precisely, Vectorial SG-GP finds the target law (up to algebraic simplifications) after 2,000,000 fitness evaluations on 5 out of 20 runs, while no perfect match could be obtained with scalar SG-GP (Table 5).

Grammar-based GP results, not shown here for space limitations, demonstrate that even scalar SG-GP is more efficient than grammar-based GP (see [18] for more details).

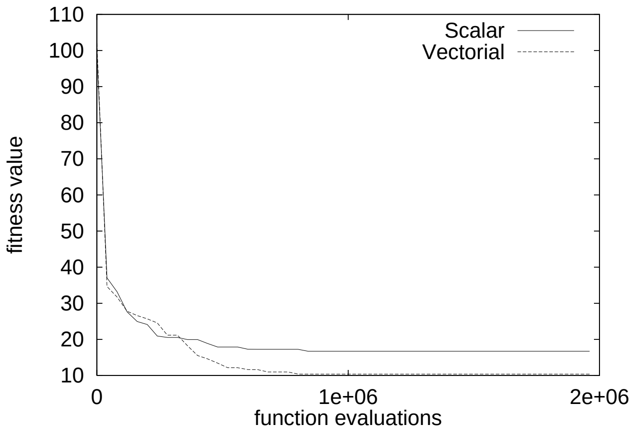


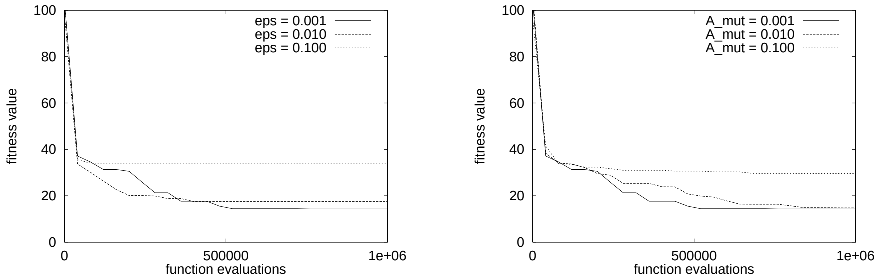
Fig. 4. Comparative performances of Scalar and Vectorial SG-GP (averaged on 20 runs)

Table 5. Exact Identification of the target solution by Scalar and Vectorial SG-GP

	Scalar	Vectorial
Number of hits	0 / 20	5 / 20
Av. best fitness	16.7 $\pm$ 5	10.3 $\pm$ 5

5.4 Sensitivity analysis

The sensitivity analysis studies the influence of the learning rate ϵ and the mutation amplitude ϵ_m on the overall performances of SG-GP (Fig 5). Overall, better results are obtained with a low learning rate and a sufficiently large mutation amplitude; this can be interpreted as a pressure toward the preservation of diversity.



(a) Learning rate (b) Mutation amplitude

Fig. 5. Parametric study of Stochastic Grammar based GP

The maximum derivation depth allowed D_{max} is also a critical parameter. Too short, and the solution will be missed, too large, the search will take a prohibitively long time.

5.5 Resisting the bloat

The most important experimental result is that SG-GP *does resist* the bloat, as it maintains an almost constant number of nodes. The average results over all individuals and 20 runs is depicted on Fig. 6.a.

In comparison is shown the number of nodes in GP (averaged on all individuals, but plotted for three typical runs for the sake of clarity). The individual size first drops in the first few generations; and after a while, it suddenly rises exponentially until the end of the run. The drop is due to the fact that many trees created by crossover in the first generations are either trivial solutions (very simple trees) or infeasible solutions which are rejected. The rise occurs as soon as large feasible trees emerge in the population.

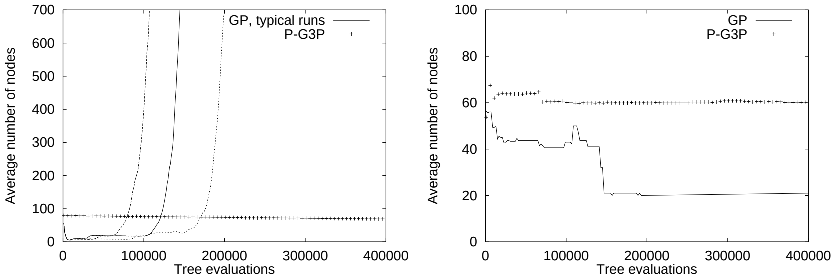
As noted by [2], the size of the best individual is not correlated with the average size. Figure 6.b shows the average size of the best-so-far individual. Interestingly, SG-GP maintains an almost constant size for the best individual, which is slightly less than the average size. On the opposite, GP converges to a very small solution, despite the fact that most solutions are very large.

5.6 Identification and Generalization

As already mentioned, SG-GP found the target law in 5 out of 20 runs (up to algebraic simplifications). In most other runs, SG-GP converges toward a local optimum, a simplified expression of which is:

$$u(t) = \frac{2 * t^2 * \frac{F}{K}}{\frac{C}{K} * \frac{C}{K} + 2 * t^2} \qquad (2)$$

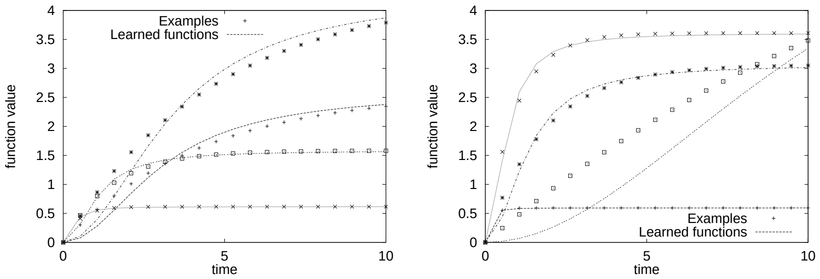
This law is not on the path to the global optimum since the exponential is missing. However, the law closely fits the training examples, at least from an



(a) Average individual size (b) Best individual size

Fig. 6. Solution size with GP and SG-GP, averaged on 20 runs

engineering point of view (Fig. 7.a).
Even more important is the fact that SG-GP finds solutions which behaves well on test examples, i.e. examples generated after the target law, which have not been considered during evolution.



(a) On training examples (b) On test examples

Fig. 7. Correct Identification and Generalization with SG-GP

By construction, the target law perfectly fits the test examples. But the non-optimal law (Eq 2) also fits the test examples; the fit is quite perfect is three out of four cases, and quite acceptable, again from an engineer's point of view, in the last case (Fig. 7.b).

6 Related works

Since the pioneering work by Langley et al. [10], many approaches have been proposed for Machine Discovery (see among others [23, 25, 22]). Rooted in the Machine Learning framework, these approaches rely on restricted assumptions (the data generator being a controlled process) [10]; or they demand the expert strong support, either interactively [23] or through background knowledge [22, 25].

The cited approaches and more generally, the Machine Discovery approaches develop clever heuristics. However, these heuristics either assume that exhaustive exploration is feasible, or that the target solution can be found through a greedy optimization process.

Schematically, stochastic-grammars based GP should be viewed as complementary to Machine Discovery; as usual, deterministic heuristics are much more efficient than stochastic ones for small or medium size problems. Still, their use sets strong requirements on the size of the search space, or equivalently on the available background knowledge. This contrasts with the wide usability of EC; even more so as SG-GP offers ways of incorporating powerful, still widely available background knowledge.

Our approach is also related to the *Probabilistic Incremental Program Evolution (PIPE)* system [20], which is also concerned with extending distribution-based evolution to GP. In PIPE, the distribution on the GP search space is represented as a Probabilistic Prototype Tree (PPT); each PPT node stores the probabilities for selecting any variable and operator in this node. One main difference is that the PPT size grows exponentially with the average tree depth, whereas the distribution size remains linear in SG-GP. This is visible as PIPE does not avoid the bloat phenomenon, contrasting with SG-GP.

7 Conclusion

In this paper was presented a novel Genetic Programming scheme, combining grammar-based GP [6, 26, 18] and distribution-based evolution [1], termed SG-GP for Stochastic Grammar-based Genetic Programming. SG-GP shows good identification abilities on the problem considered, as the target law was discovered in 5 out of 20 runs, while it was never discovered by canonical GP. What is equally important, SG-GP shows good generalization abilities; even in the cases where the target law was missed, the solutions found by SG-GP have good predictive accuracy on test experimental data. Last, SG-GP successfully completed these tasks by exploring individuals with constant size. No intron growth was observed; the overall memory requirements were lower by several orders of magnitude, than for canonical GP.

Fundamentally, SG-GP remedies two main limitations of GP. On one hand, it allows for exploiting powerful background knowledge, such as dimensional consistency. On the other hand, it successfully resists the bloat phenomenon, avoiding the intron growth.

The latter fact is surprising, for intron growth was considered to be unavoidable for program induction methods with fitness-based selection [9]. This conjecture is infirmed by SG-GP results on a real-world like problem, suggesting that intron growth is not necessary to achieve efficient non parametric learning in a fitness-based context, but might rather be a side effect of crossover-based evolution.

On-going work is concerned with examining the actual limitations of SG-GP through more intensive experimental validation.

One such limitation to be addressed regards the problem of adjusting the constants of the target laws (the parametric optimization problem within the non-parametric one).

In the long run, a key aspect for Machine Discovery with SG-GP is to determine an adequate grammar, i.e. to find prior restrictions on the search space. A perspective for further research is to use grammatical inference from the body of knowledge in the application domain (e.g. laws reported in textbooks) to this aim.

Acknowledgments

The authors are partially supported by ILPNet2 Network of Excellence (INCO 977102).

References

1. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Prieditis and S. Russel, eds, *Proc. of the 12^{th} Int Conf. on Machine Learning*, pages 38–46. Morgan Kaufmann, 1995.
2. W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.
3. T. Bäck. *Evolutionary Algorithms in theory and practice*. New-York:Oxford University Press, 1995.
4. J.M. Daida. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In *Proc. of the Genetic and Evolutionary Conf. 99*, pages 1069–1076. Morgan Kaufmann, 1999.
5. J. Duffy and J. Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In *Evolutionary Computation in Economics and Finance*. Springer Verlag, 1999.
6. F. Gruau. On using syntactic constraints with genetic programming. In P.J. Angeline and K.E. Kinnear Jr., eds, *Advances in Genetic Programming II*, pages 377–394. MIT Press, 1996.
7. C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.
8. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.
9. W. B. Langdon and R. Poli. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer Verlag, 1997.
10. P. Langley, H.A. Simon, and G.L. Bradshaw. Rediscovering chemistry with the BACON system. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, eds, *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

11. P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.
12. B. McKay, M.J. Willis, and G.W. Barton. Using a tree structures genetic algorithm to perform symbolic regression. In *IEEE Conf. publications, n. 414*, pages 487–492, 1995.
13. David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
14. P. Nordin, W. Banzhaf, and F.D. Francone. Introns in nature and in simulated structure evolution. In D. Lundh, B. Olsson, and A. Narayanan, eds, *Biocomputing and Emergent Computation*, pages 22–35. World Scientific, 1997.
15. M. O'Neill and C. Ryan, eds. *Grammatical Evolution.* Genetic and Evolutionary Conf. Workshop, GECCO 2002, 2002.
16. M. Pelikan, D.E. Goldberg and E. Cantu-Paz. BOA: the bayesian optimization algorithm. In *Genetic and Evolutionary Conf., GECCO 1999*, pages 525–532. Morgan Kaufmann, 1999.
17. N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–20, 1991.
18. A. Ratle and M. Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In M. Schoenauer et al., editor, *Proc. of the* 6^{th} *Conf. on Parallel Problems Solving from Nature*, pages 211–220. Springer-Verlag, LNCS 1917, 2000.
19. C. Ryan, J.J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T.C. Fogarty, eds, *Genetic Programming, First Eur. Workshop, EuroGP98*, pages 83–96. Springer Verlag LNCS 1391, 1998.
20. R. Salustowicz and J. Schmidhuber. Evolving structured programs with hierarchical instructions and skip nodes. In J. Shavlik, editor, *Proc. of the* 15^{th} *Int Conf. on Machine Learning*, pages 488–496. Morgan Kaufmann, 1998.
21. M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuous search spaces. In Th. Bäck, G. Eiben, M. Schoenauer, and H.-P. Schwefel, eds, *Proc. of the* 5^{th} *Conf. on Parallel Problems Solving from Nature*, pages 418–427. Springer Verlag, 1998.
22. L. Todorovski and S. Dzeroski. Using domain knowledge on population dynamics modeling for equation discovery. In P. Flach and L. De Raedt, eds, *Proc. of the* 12^{th} *Eur. Conf. on Machine Learning*, pages 478–490. Springer Verlag LNAI 2167, 2001.
23. R. Valdes-Perez. Machine discovery in chemistry: New results. *Artificial Intelligence*, 4:191–201, 1995.
24. I.M. Ward. *Mechanical Properties of Solid Polymers.* Wiley, Chichester, 1985.
25. T. Washio, H. Motoda, and Y. Niwa. Discovering admissible simultaneous equation models from observed data. In L. de Raedt and P. Flach, eds, *Proc. of the* 12^{th} *Eur. Conf. on Machine Learning*, pages 539–551. Springer Verlag LNAI 2167, 2001.
26. P.A. Whigham. Inductive bias and genetic programming. In *IEEE Conf. publications, n. 414*, pages 461–466, 1995.
27. Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.