THE ROLE OF KEEPING “SEMANTIC BLOCKS” INVARIANT
Effects in Linear Genetic Programming Performance

Marina de la Cruz Echeandía, Alba Martín Lázaro, Alfonso Ortega de la Puente
Departamento de Ingeniería Informática, Escuela Politéctnica Superior
Universidad Autónoma, Madrid, Spain

José Luis Montaña Arnáiz
Departamento de Matemáticas, Estadística y Computación, Facultad de Ciencias
Universidad de Cantabria, Santander, Spain

César L. Alonso
Centro de Inteligencia Artificial, Universidad de Oviedo, Gijón, Spain

Keywords: Grammar evolution, Attribute grammars, Christiansen grammars, Genetic programming, Straight-line programs, symbolic regression.

Abstract: This paper is focused on two different approaches (previously proposed by the authors) that perform better than Genetic Programming in typical symbolic regression problems: straight-line program genetic programming (SLP-GP) and evolution with attribute grammars (AGE). Both approaches have different characteristics. One of the most important is that SLP-GP keeps semantic blocks invariant (the crossover operator always exchanges *complete* subexpressions). In this paper we compare both methods and study the possible effect on their performance of keeping these blocks invariant.

1 MOTIVATION

The so called Holland *schema theorem* (Holland92, 1975) explains the reasons why genetic based methods guarantee to find quasi-optimal solutions. It shows how some relevant patterns (schema) that could be associated with higher fitness values, are kept after applying genetic operators by the fitter individuals. Keeping these patterns is a way to improve the fitness.

These algorithms are supposed to use binary genotypes to which the fitness function is applied. Nevertheless, genetic programming techniques usually code the final programs in a more complex way, because it is more difficult to translate genotypes into phenotypes if they have to meet the constrains of the programing language under consideration. Genetic programming frequently uses tree-like data structures, like the original GP proposal (Koza, 1992) or non-binary strings, like Grammatical Evolution and its derivatives (ONeill and Conor, 2003; de la Cruz et al., 2005; Ortega et al., 2007) to code their genotypes. When the mapping from genotype to phenotype is complex, it is more difficult to trace or even identify the schema.

In this paper we will use the name “semantic blocks” for those genotype fragments associated with some specific meaning in the phenotype. For example, given an assignment of some arithmetic subexpression to some variable, such as $z = x + 3 - y$, a semantic block could give a proper value to its variables, such as $x = 3; y = x - 2$.

This paper is focused on the possible improvement in the performance of some genetic programming algorithms that could be caused by genetic operators that keep these semantic blocks invariant.

The paper compares the performance of two different approaches previously proposed by the authors: attribute grammar evolution (AGE) (de la Cruz et al., 2005) and genetic programing based on straight line programs (SPL-GP) (Alonso et al., 2008) Both methods have proven to improve the efficiency of classic GP and incorporate complex formal represen-

de la Cruz Echeandía M., Martín Lázaro A., Ortega de la Puente A., Luis Montaña Arnáiz J. and L. Alonso C..
THE ROLE OF KEEPING “SEMANTIC BLOCKS” INVARIANT - Effects in Linear Genetic Programming Performance.
DOI: 10.5220/0003085403650368
In *Proceedings of the International Conference on Evolutionary Computation* (ICEC-2010), pages 365-368
ISBN: 978-989-8425-31-7

tations for the candidate solutions (respectively attribute grammars and straight line programs). In addition, SPL-GP uses genetic operators that preserve the structure of the "semantic blocks". Even although preserving semantic blocks is not the only difference between AGE and SPL-GP, we think that a first comparison of both methods as they are defined could provide some information about the effect on the performance of keeping semantic blocks invariant. One of our future goals is to define an AGE extension adding a mechanism able to keep the semantic blocks inspired by SPL-GP.

2 INTRODUCTION TO SLP-GP

It is easy to see that any program (algorithm) can be expressed as a sequence of predefined *atomic operations*. A standard way of representing each operation is by means of vectors with $n+2$ components. The first component usually holds the operation performed. Another element refers to the place where the result will be stored (it usually is a different and new variable for each operation). There are n additional components for a maximum of n operands.

Quadruples belong to this kind of representations ($n+2$ vectors with four elements, $n=2$).

For example, the expression $2x_2(x_1+1)^2-2x_2$ could be represented by the following sequence of quadruples: $\{\ u_1 := x_1+1\ ,\ u_2 := u_1 * u_1\ ,\ u_3 := x_2+x_2\ ,\ u_4 := u_2 * u_3\ ,\ u_5 := u_4 - u_3\ \}$

It is easy to see that the semantic block of a given element (quadruple) contains all the elements of the SLP needed to properly evaluate it. For example: the semantic block of the second quadruple ($u_2 := u_1 * u_1$) is the sequence $\{\ u_1 := x_1+1\ ,\ u_2 := u_1 * u_1\ \}$, while the semantic block of the third element ($u_3 := x_2+x_2$) contains only this quadruple and the semantic block of the last quadruple is the complete SLP.

Further details of SLP-GP and examples of its crossover operator can be found in (Alonso et al., 2008). The reader can find there a more formal approach and definitions of slp's.

3 INTRODUCTION TO AGE

Attribute grammars (Knuth, 1968) are one of the tools used to completely describe high level programming languages (both their syntax and their semantics).

AGE (de la Cruz et al., 2005) is an extension to Grammatical Evolution (ONeill and Conor, 2003). Both techniques are automatic programming evolutionary algorithms independent of the target programming language, and include a standard representation of genotypes as strings of integers (codons), and a formal grammar (respectively attribute and context free grammars) as inputs for the deterministic mapping of a genotype into a phenotype. This mapping minimizes the generation of syntactically (and in the case of AGE also semantically) invalid phenotypes. Genetic operators act at the genotype level, while the fitness function is evaluated on the phenotypes.

Further details, deeper descriptions and examples can be found in (de la Cruz et al., 2005).

4 SLP-GP VS. AGE

There are several differences that make it difficult to compare both methods. We can briefly summarize them as follows

- **Length of Genotypes.** SLP-GP uses slp's of "length" 12 (with 12 operations or instructions) while AGE uses genotypes with variable length. Like other variable length evolutionary algorithms, AGE suffers from bloating (the unbounded increase in the length of the genotypes while the search evolves). Pruning (removing some fragments of genotypes) is a genetic operator useful to control bloating. AGE removes the codons not used after the translation into phenotype.
- **Mapping from Genotype to Phenotype.** AGE includes a mapping from genotype to phenotype driven by the attribute grammar that generates the language of candidate solutions. SLP-GP uses slp's both as genotypes and as phenotypes. In AGE, the number of codons (length of the corresponding genotype) needed to generate a phenotype of a given length depends on the structure of the specific grammar and it is, consequently, more difficult to estimate.
- **The Way in which Computational Effort is Computed.** In (Alonso et al., 2008) the number of basic operations is used to measure the computational effort. Each slp has 12 operations and each generation contains 200 different genotypes. Therefore, SLP-GP could compute the computational effort either by means of the total number of basic operations, candidate solutions or generations needed to find the solution, because all of them are directly proportional. In (Alonso et al., 2008) the maximum number of basic operation is 10^7. This condition is equivalent to a maximum of 5000 generations. Nevertheless, as we have previously introduced for AGE, it is difficult to es-

timate how many codons are needed to generate a phenotype with a given number of basic operations. Instead of the total number of basic operations performed, we have used the cumulative success frequency taking into account the generation in which the successful experiments find the solution.

4.1 Experimental Setting

In order to compare SLP-GP and AGE we have reproduced the experiments performed in (Alonso et al., 2008) for the following real functions: $f_1(x) = x^4 + x^3 + x^2 + x$, $f_2(x) = e^{-sin3x+2x}$, $f_3(x) = 2.718x^2 + 3.1416x$, $f_4(x) = min\{\frac{2}{x}, sin(x)+1\}$.

We have used the same parameters when possible: 30 sample points, 20 individuals in the population, functions set $F = \{+,-,*,/^{(*)}\}$, constants set $\{0,1,2\}$, crossover rate $= 0.9$, 100 runs per function, 5000 generation at maximum, and error thresholds $3.716070e-07, 7.623910e-01, 5.265160e-01$ and $1.425200e-02$ for the solutions respectively of f_1, f_2, f_3 and f_4. Sample points have been respectively taken from $[-5,5]$, $[-\frac{\pi}{2},\frac{\pi}{2}]$, $[-\pi,\pi]$, and $[0,15]$. The following additional functions have been respectively added to F: $\{sqrt^{(*)}\}$, $\{sqrt^{(*)}, sin, cos, exp\}$, $\{sin, cos\}$, and $\{sin, cos\}$ ($^{(*)}$ used only by SLP-GP).

There is a significant difference between the set of functions used by both methods. $/^{(*)}$ and $sqrt^{(*)}$ represent *protected versions (against undefined results)* of, respectively, $/$ and $sqrt$. These versions return 1 when they are undefined ($x = 0$ and $x < 0$ respectively). AGE considers semantically incorrect those individuals with undefined subexpressions and, subsequently, AGE never generates them.

In addition, protected functions change the behavior of the final expressions and make them difficult to understand and describe.

The remaining parameters were tuned separately. For SLP-GP: mutation and reproduction rates $= 0.05$ and slp's length $= 12$. For AGE: mutation rate $= 0.9$ and variable length genotypes.

5 EXPERIMENTAL RESULTS

It is possible to compare the performance of both methods, if we do not take into account the number of basic operations performed to get the solutions.

We will show two kinds of graphics:

1. The cumulative success frequency with respect to the generation in which the quasi-optimal solution is found (figure 1). It shows how many runs successfully finish and with which speed. Some of the curves does not show 5000 generations because they are focused on the range in which most of the solutions are found
2. The empirical distribution of the best fitness displayed using the standard box plot notation with marks at best execution, 25, 50, 75 per cent, and worst execution (figure 2). We consider that it globally describes the *quality* of the populations.

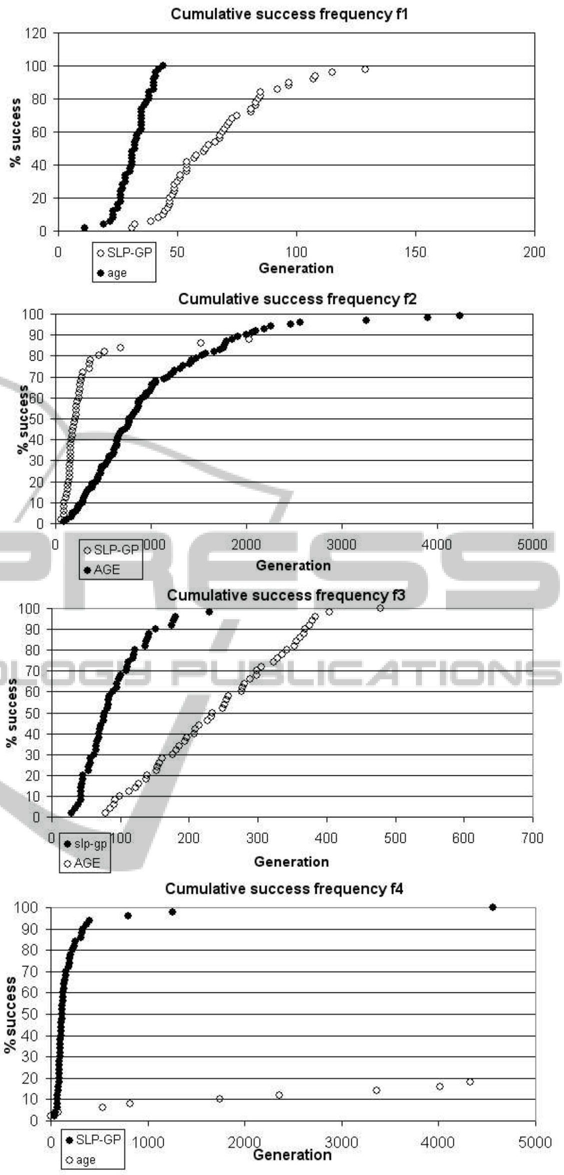

Figure 1: Cumulative success frequency for $f1$, $f2$, $f3$, and $f4$.

It can be easily seen that, except for $f1$, SPL-GP performs better than AGE, specially for function $f4$. The poor performance of AGE for more difficult cases could be caused by the capability of SPL-GP for pre-

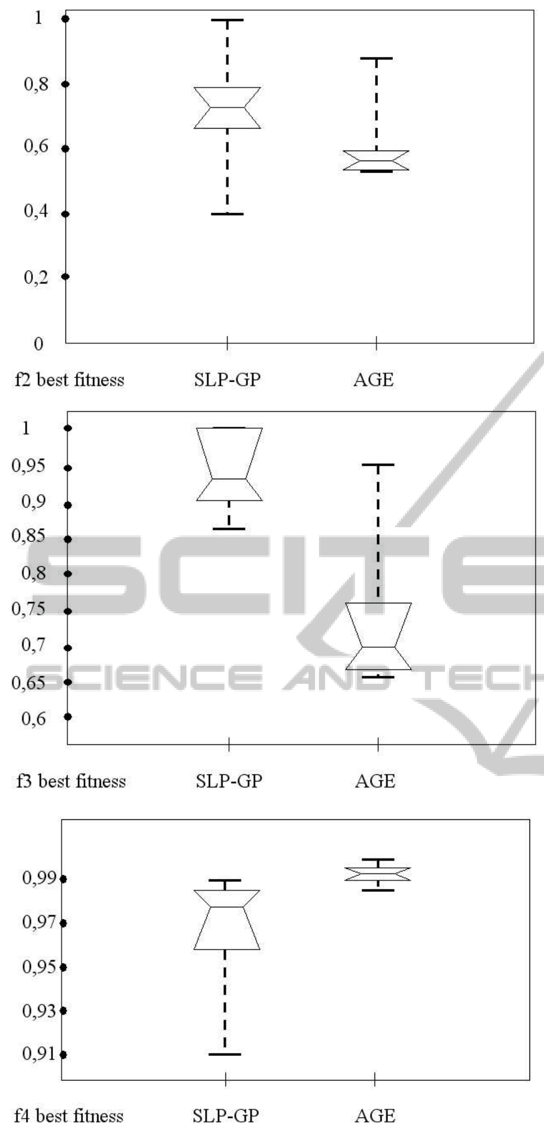

Figure 2: Empirical best fitness distribution for $f2$, $f3$, and $f4$.

serving semantics blocks. No figure shows the empirical distribution for $f1$ because both methods gets fitness values above 0.9999995 for all the runs. It is also worth mentioning that AGE is able to get the exact (zero error) target function in some runs for almost all the cases ($f1$, $f2$ and $f3$) while SLP-GP only approximates them.

6 CONCLUSIONS AND FURTHER RESEARCH LINES

We have compared two different genetic programming algorithms previously proposed by the authors. From our point of view, the main difference between these methods is that SLP-GP implements a crossover operator that keeps semantic blocks invariant.

We have designed a set of graphics to compare the performance of both methods, taking into account their differences. We conclude that SLP-GP performance is better. We think that this is a consequence of keeping semantic blocks invariant.

In the future we would like to gather a wider set of performance data using different target functions to get a deeper comparison between SPL-GP and AGE. We would like to propose a new version of AGE and CGE that extends crossover to keep semantic blocks inviriant, as in SPL-GP. We will then compare SPL-GP with the new algorithm.

ACKNOWLEDGEMENTS

This work was partially supported by the R&D program of the Community of Madrid (S2009/TIC-1650, project “e-Madrid”) as well as by the Spanish Ministry of Science and Innovation (TIN2007-67466-C02-02). The authors thank Dr. Manuel Alfonseca for his help to prepare this document.

REFERENCES

Alonso, C. L., Montaña, J. L., and Puente, J. (2008). Straight line programs: a new linear genetic programming approach. *Proc. 20th IEEE International Conference on Tools with Artificial Intelligence (IC-TAI)*, pages 517–524.

de la Cruz, M., Ortega, A., and Alfonseca, M. (2005). Attribute grammar evolution. In Mira, J. and ’Alvarez, J., editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 182–191, Berlin / Heidelberg. Springer.

Holland92 (1992 (originally published in 1975)). *Adaptation in Natural and Artificial Systems*. The MIT Press, London, 2nd edition.

Knuth, D. E. (1968). *Semantics of Context-Free Languages*. Mathematical Systems Theory, vol. 2, n 2, pp. 127-145.

Koza, J. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge.

ONeill, M. and Conor, R. (2003). *Grammatical Evolution, evolutionary automatic programming in an arbitrary language*. Kluwer Academic Phblishers.

Ortega, A., de la Cruz, M., and Alfonseca, M. (2007). Christiansen grammar evolution: Grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):77–90.