Pre-, In- and Postfix grammars for Symbolic Regression in Grammatical Evolution

Erik Hemberg
NCRA Group
University College Dublin
Ireland
erik.hemberg@ucd.ie

Nicholas McPhee
Div. of Science & Mathematics
University of Minnesota, Morris
USA
mcphee@morris.umn.edu

Michael O'Neill
NCRA Group
University College Dublin
Ireland
m.oneill@ucd.ie

Anthony Brabazon
NCRA Group
University College Dublin
Ireland
anthony.brabazon@ucd.ie

***Abstract*—Recent research has indicated that grammar design is an important consideration when using grammar-based Genetic Programming, particularly with respect to unintended biases that may arise through rule ordering or duplication. In this study we examine how the ordering of the elements during mapping can impact performance. Here we use to the standard GE depth-first mapper and compare the performance of postfix, prefix and infix grammars on a selection of symbolic regression problem instances. We show that postfix can confer a performance advantage on the harder problems examined.**

I. Introduction

A Substantial literature has emerged on the grammar-based form of Genetic Programming, Grammatical Evolution (GE), and its applications (e.g., [1], [2], [3]). Here we investigate the importance of the ordering of the mapping process that occurs during the generation of a solution. Traditional GE constructs derivation trees in a depth-first manner. In πGE, however, individuals can evolve the order in which non-terminals are expanded, leading to performance gains [4]. This indicated that the order in which non-terminals are expanded can affect search efficiency. Other studies also indicate that grammar design itself can impact an algorithm's performance [5], [6], [7], [15]

Here we use the standard depth-first mapper, with three grammars which differ only in their expression syntax. The first grammar is infix (typical in most previous GE work), the second is prefix, and the third is postfix. We then compare the performance of these grammars on a suite of symbolic regression problem instances. If the order in which non-terminals are mapped is truly important we would expect performance differences between the starkly contrasting prefix and postfix grammars. With prefix grammars, for example, operators are determined earlier in the input sequence than the operands, where the opposite is true for postfix. As a result the root of a syntax tree is the last component of a program that is determined in postfix, as opposed to the first with prefix. See Fig. I where the grammars from Fig. I produce the derivation trees.

The structure of the paper is first an introduction to GE in Sec. II, then a description of the experimental setup and results in Sec. III, and finally Conclusions & Future Work in Sec. IV.

II. Background

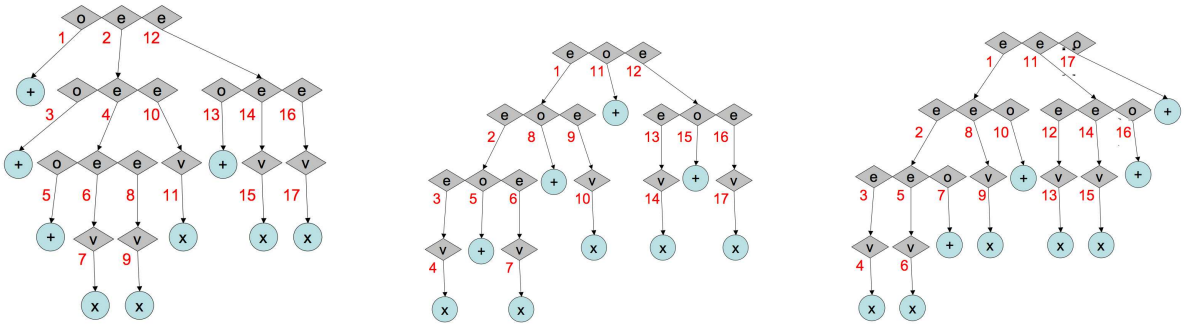
GE [1] is a form of grammar-based genetic programming(GP) [8]. For more background on grammar-based GP see [13], [14] Rather than representing the programs as parse trees, as in GP, a variable length linear genome representation is used in GE. A genotype-phenotype mapping is employed where an individual's binary string is interpreted as a sequence of integer values (called *codons*), which are then used to select production rules from a Backus-Naur Form (BNF) grammar, see Fig. 3. A context free grammar (CFG) is a four tuple (N, Σ, R, S). Where N is a finite set of non-terminal symbols. Σ is a finite set of terminal symbols, $N \cap \Sigma = \emptyset$. R is a finite set of production rules, $A \rightarrow \alpha$, $A \in N$ and $\alpha \in (\Sigma \cup N)^*$, and S is the start symbol, $S \in N$. An example grammar is shown in Fig. 2.

```
---- Prefix ----
<e> ::= ( <o> <e> <e> ) | <v>
---- Infix ----
<e> ::= ( <e> <o> <e> ) | <v>
---- Postfix ----
<e> ::= ( <e> <e> <o> ) | <v>
---- Common for all grammars----
<o> ::= +|-|*|/
<v> ::= x0 | x1 | <c>
<c> ::= 1|2|3|4|5|6|7|8|9
```

Fig. 2. The CFGs that were used for the experiments, with the different rules for each grammar design as well as the common rules. `e` is an expression, `o` is an operator, `v` is a variable and `c` a constant.

In GE the genetic operators such as crossover and mutation are applied to the linear genotype in a typical genetic algorithm (GA) [9] manner, unlike in a GP approach where they are applied directly to the phenotypic parse trees. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct. The user can design the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar. The mapping process creates a clear distinction between the search and solution space.

By using different grammars the search space can be mapped and explored in different ways. This is illustrated by examining the derivation trees that are created when mapping the genotype to the phenotype. Fig. I shows how different grammars can lead to different derivation trees that in fact



(a) Prefix mapping (b) Infix mapping (c) Postfix mapping

Fig. 1. Derivation trees mapped from the different grammars from Fig. 2. The grammars generate equivalent expressions from different inputs of length 17 and the input number is indicated in the figure. Diamonds denote non-terminal symbols and circles denote terminal symbols.

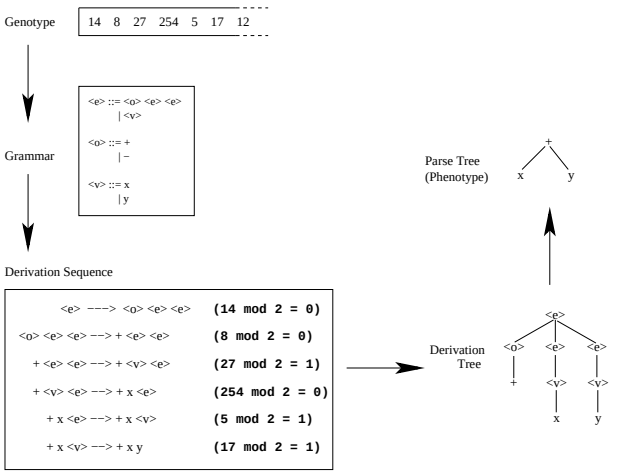


Fig. 3. GE genotype-phenotype mapping, the genotype selects production rules from a grammar to produce a derivation sequence.

represent the same phenotype. (The input sequences used to generate the trees, however, are different in each case.)

III. Experiments & Results

To test the effects of different grammar designs a small but expressive grammar of Symbolic regression was used.

1) Symbolic regression: The goal is to find a function that matches a target function on a set of observed points. In this paper the following target functions were used:

$$8/(2+x^2+y^2) \tag{1}$$

$$x^3(x-1)+y(y/2-1) \tag{2}$$

$$x^3/5+y^3/2-y-x \tag{3}$$

$$\frac{30*x^2}{(10-x)y^2}+x^4-x^3+\frac{y^2}{2}-y+\frac{8}{2+x^2+y^2}+x \tag{4}$$

$$\frac{30*x^2}{(10-x)y^2}+x^4-\frac{4}{5}x^3+\frac{y^2}{2}-2y+\frac{8}{2+x^2+y^2}+\frac{y^3}{2}-x \tag{5}$$

Some of these target functions were adopted from [10], while others where created to encourage the evolution of larger expression trees. From the range [-3,3] for both x and y 20 random sample points where chosen. Fig.4 shows the target functions plotted over this range, together with diagrams showing the structure of the target expressions, with the structural complexity increasing with each target.

TABLE I
Parameters for the GE algorithm

Fitness function	See III-1
Initialisation	Ramped Half and Half
Grow Derivation tree depth	12
Selection operation	Tournament
Tournament size	3
Replacement	Generational
Elites	2
Population size	500
Max wraps	1
Generations	50
Crossover probability	0.9
Mutation probability	0.01

2) Grammar: The grammars used are shown in Fig.2. The only variation is between prefix, infix and postfix representation of the function expression. This means that the grammars have different sites that determine the order of the expansion of the grammar in relation to the root, see Fig. I.

A. Experiment

The experiments are designed to test whether there is a difference in the performance between the different grammars. The performance is measured as the average best fitness μ after 50 generations over 1000 runs. The false discovery rate(FDR) [12] is calculated and the p-values are derived from t-tests. The FDR value tells how many of the p-values from the multiple hypotheses that were significant given the α of the FDR-test.

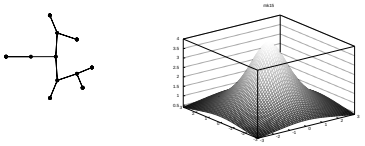
1) Hypothesis:

H_0: No difference in best fitness between the grammars. $\mu_{Pre}=\mu_{In}$, $\mu_{In}=\mu_{Post}$ and $\mu_{Pre}=\mu_{Post}$

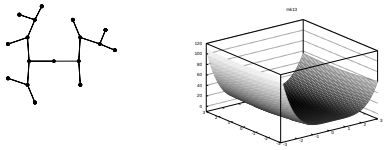
H_1: A difference in best fitness between the grammars. $\mu_{Pre}\neq\mu_{In}$, $\mu_{In}\neq\mu_{Post}$ or $\mu_{Pre}\neq\mu_{Post}$

α: The significance level of the test is 0.05.

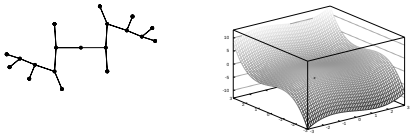
2) Setup: Parameter settings for the GE algorithm are listed in Table I. The input (called *chromosomes*) were variable-length vectors of integers (4 byte integers). Our fitness measure is the sum of the squared error over the



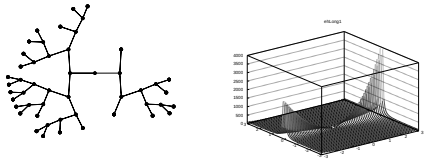
(a) Target (1), n=11, l=6



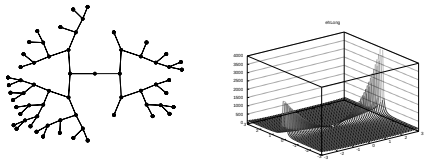
(b) Target (2), n=17, l=9



(c) Target (3), n=19, l=10



(d) Target (4), n=49, l=25



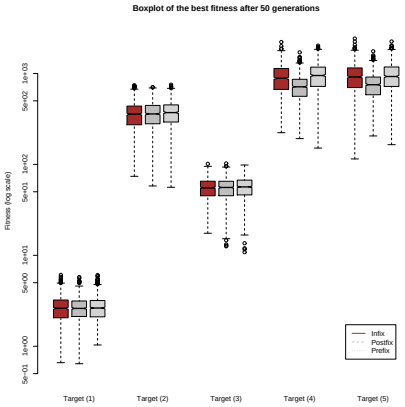
(e) Target (5), n=63, l=32

Fig. 4. Expression trees and a plot of the function over the range. Tree generating code from [11]. n = total number of nodes, l = number of leaves.

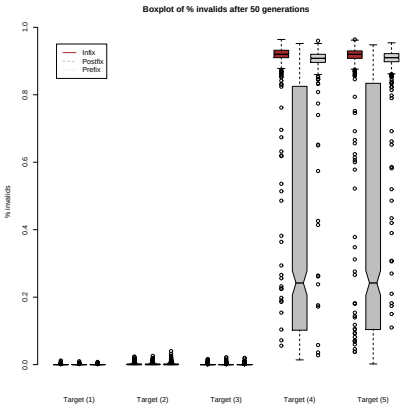
20 test cases. One-point variable length crossover was used, along with an integer mutation operator where a new value was randomly chosen. For division a naive protection was implemented, 0.0 was returned if the divisor equalled 0. An individual is invalid if it cannot produce a valid phenotype after it is mapped. Invalids are given the worst possible fitness.

B. Results

The best fitness over time is shown in Fig. 6 and in Fig. 5(a) boxplots of the runs are shown. Due to space restrictions values from the t-tests are left out. Looking at the last generation of pairwise comparisons between the different grammars. The FDR value is 3 for Target (4) for postfix and infix compared to prefix, and infix compared to postfix. For Target (5) the FDR value is 2, here for infix compared to prefix and infix compared to postfix. Also postfix compared to prefix for Target (2) has an FDR value of 1.



(a) Best fitness, y=axis is log scale



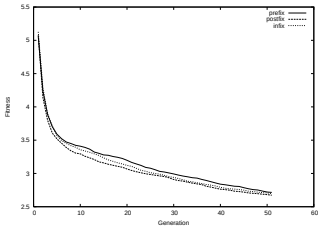
(b) % invalids in population

Fig. 5. Boxplots of Best Fitness and % invalids at the final generation.

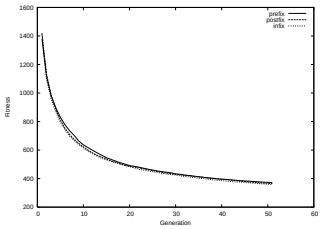
On the two larger problem instances, Targets (4 and 5), a performance advantage was observed for postfix when compared to both infix and prefix. Additionally, on Target (5) infix outperformed prefix. No statistically significant differences in performance were observed on the smaller Targets (1, 2 & 3). When studying the results from Fig. 5(b) one can notice that postfix grammars always have more valid individuals when compared to prefix, the FDR value was always 2, except for Target (1), although for Target (2,3) the number of invalids in all grammars is very low but for Targets (5,4) the difference is quit high.

C. Discussion

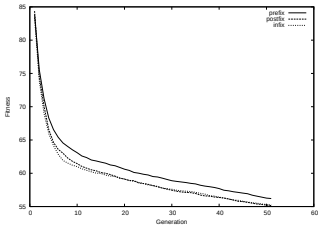
All grammars show a similar behaviour when it comes to fitness. An inspection of the log data recorded for each run revealed that for the prefix grammar, a significantly large number of invalid individuals were generated after the



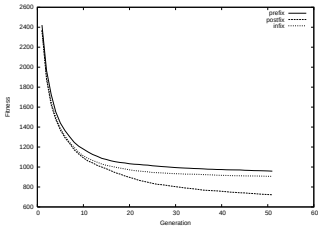
(a) Target (1), best fit.



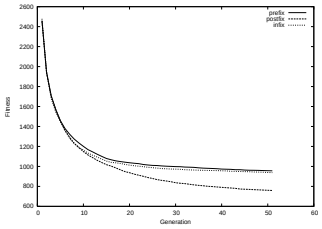
(b) Target (2), best fit.



(c) Target (3), best fit.



(d) Target (4), best fit.



(e) Target (5), best fit.

Fig. 6. Best fitness results averaged over 1000 runs.

initial population. Clearly, this might account for some of the differences in performance observed, but it is interesting to ask why are so many invalids being generated? One explanation could be the different locations of the grammar expansions in the input string.

IV. Conclusion & Future work

We wished to see if the order of symbols within a grammar can impact on performance of GE by comparing infix, postfix and prefix syntactical variants. The results suggest that the choice of grammar can produce performance advantage on the different problems examined.

In order to further understand the impacts of grammar design and GE more problem types need to be tried. A discrete problem like 6-MUX should be investigated to see if this problem type exhibits the same behaviour. Further it might be interesting to try a problem that is solved by GE, as well as compare the findings to other grammar-GP systems.

Future studies will examine the number of invalids and focus on how the search operators are manipulating the solutions with different syntactical representations.

Acknowledgement

This research is based upon works supported by the Science Foundation Ireland under Grant No. 06/RFP/CMS042.

References

[1] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[2] F. Rothlauf and M. Oetzel, "On the locality of grammatical evolution." in *EuroGP*, LNCS, P. Collet, et al, Eds., vol. 3905. Springer, 2006, pp. 320–330.

[3] U.-M. O'Reilly and M. Hemberg, "Integrating generative growth and evolutionary computation for form exploration," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 163–186, 2007.

[4] A. Brabazon and M. O'Neill, "Credit rating with pi grammatical evolution," in *Proceedings of Computer Methods and Systems Conference*, R. Tadeusiewicz, et al, Eds., vol. 1. Krakow, Poland: Oprogramowanie Naukowo-Techniczne Tadeusiewicz, 14-16 Nov. 2005, pp. 253–260.

[5] E. Hemberg, M. O'Neill and A. Brabazon, "Grammatical bias and building blocks in meta-grammar grammatical evolution," in *2008 IEEE World Congress on Computational Intelligence*, J. Wang, Eds, IEEE Press 1-6 June 2008.

[6] M. O'Neill and C. Ryan, "Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code," in *EuroGP 2004, Proc.*, LNCS, M. Keijzer, et al, Eds., vol. 3003. Coimbra, Portugal: Springer-Verlag, 5-7 Apr. 2004, pp. 138–149.

[7] M. Nicolau, "Automatic grammar complexity reduction in grammatical evolution," in *GECCO 2004 Workshop Proc.*, R. Poli, et al, Eds., Seattle, Washington, USA, 26-30 Jun. 2004.

[8] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. MIT Press, Dec. 1992.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Jan. 1989.

[10] M. Keijzer, "Improving symbolic regression with interval arithmetic and linear scaling," in *EuroGP 2003, Proc.*, LNCS, C. Ryan, et al, Eds., vol. 2610. Essex: Springer-Verlag, 14-16 Apr. 2003, pp. 70–82.

[11] S. Gustafson, http://www.gustafsonresearch.com/research/vis/.

[12] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society B*, vol. 57, no. 1, pp. 289-300, 1995.

[13] P. A. Whigham, "Grammatically-based genetic programming," in *Proc. of the Workshop on Genetic Programming: From Theory to Real-World Applications*, J. P. Rosca, Ed., Tahoe City, California, USA, 9 1995, pp. 33–41.

[14] R. I. (Bob) McKay, X. H. Nguyen, P. A. Whigham, and Y. Shan, "Grammars in genetic programming: A brief review." in *Progress in Intelligence Computation and Intelligence: Proceedings of the International Symposium on Intelligence, Computation and Applications*, L. Kang, et al, Eds, ppp 3–18, Wuhan, PRC, April 2005. China University of Geosciences Press.

[15] Marco A. Montes de Oca. "Exposing a bias toward short-length numbers in grammatical evolution.", in *LNCS 4971. EuroGP 2008, Proc*, S. Gustafson, et al, Eds, pp 278–288, Berlin, 2008. Springer.