Exploring Position Independent Initialisation in Grammatical Evolution

Conference Paper · July 2016

CITATIONS
0

READS
2

3 authors, including:

David Fagan
University College Dublin
13 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Michael Fenton
University College Dublin
12 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Exploring Position Independent Initialisation in Grammatical Evolution

David Fagan*, Michael Fenton†, Michael O'Neill‡
Natural Computing Research and Applications Group
School of Business
University College Dublin
Dublin, Ireland
* Email:david.fagan@ucd.ie
† Email: michael.fenton@ucd.ie
‡ Email: m.oneill@ucd.ie

***Abstract*—Initialisation in Grammatical Evolution (GE) is a topic that remains open to debate on many fronts. The literature falls between two mainstay approaches: random and sensible initialisation. These methods are not without their drawbacks with the type of trees generated. This paper tackles this problem by extending these traditional operators to incorporate position independence in the initialisation process in GE. This new approach to initialisation is shown to provide a viable alternative to the commonly used approaches, whilst avoiding the common pitfalls of traditional approaches to initialisation.**

I. Introduction

Genetic Programming [1] has been shown to benefit from having an initial population that is initialised to individuals of varying shapes and sizes. The most prevalent approach, Ramped Half-and-Half (RHH), has been shown to have some drawbacks [2]. Grammatical Evolution (GE) has also been the beneficiary of adopting a similar approach to initialisation [3]. The issues seen in GP are further compounded in GE with the additional complexity of the grammar. These issues have been explored in depth [4], [5], however the random and RHH approaches still remain widely used in the literature.

Position Independence in GE [6] is a method for derivation tree construction whereby the order of constructing the derivation tree is encoded in the genome, and is therefore subject to evolution. This approach makes for trees being constructed in what appears to be a random order. Traditionally GE uses a recursive depth-first order for derivation tree building. RHH is implemented in a similarly recursive manner, with little knowledge of the surrounding tree structure, besides its parent and child nodes. This study looks to investigate if using some form of position independence in tree construction during initialisation can improve the structural diversity of the initial population over standard RHH initialisation [4], [5].

The paper proceeds as follows. Section II introduces GE. Section III outlines the new position independent initialiser. Section IV outlines the approach used to examine the tree shapes of the population. Section V outlines the experimental setup, before the results of the study are presented in Section VI. A discussion follows in Section VI-C, before conclusions and future work are stated in Section VII.

II. Grammatical Evolution

Grammatical Evolution (GE) [7], [8], is a grammar-based form of Genetic Programming (GP) [9]. GE takes inspiration from DNA-protein mapping in its approach to the generation of solutions. GE relies upon the use of a list of integers referred to as a chromosome, or genotype. This chromosome is mapped to a phenotype, or solution, through the application of a BNF grammar to the chromosome.

O'Neill [10] presented a series of advantages in the adoption of a genotype-phenotype map for GP. These include a generalized encoding that can represent a variety of structures allowing GP to generate structures in an arbitrary language, efficiency gains for evolutionary search (e.g. through neutral evolution), maintenance of genetic diversity through many-to-one maps, preservation of functionality while allowing continuation of search at a genotypic level, reuse of genetic material potentially allowing information compression, and positional independence of gene functionality.

GE has been extended to incorporate tree based variation operations such as those seen in CFG-GP [11], and more recently semantic methods [12], [13].

A. Initialisation in GE

Sensible initialisation [3] is a GE implementation of standard GP Ramped Half-and-Half initialisation [1], [2]. However, a notable difference between the two methods arises from the use of a formal grammar to define the function and terminal sets. Whereas GP builds trees by choosing from function and terminal sets until the required size is reached, grammar-based methods are bound by the qualities of production choices [3]. The grammar must first be traversed in order to ascertain which production rules are recursive. Furthermore, each non-terminal must carry knowledge of its minimum path (i.e. the shortest possible distance until all resultant non-terminals have been fully expanded). The initialisation method then selects from either recursive or non-recursive non-terminals in order to reach the requisite depth level.

Similar to how the GE chromosome is read from left to right, Sensible initialisation in GE builds trees in a left-first manner. This means that branches of the derivation tree are

978-1-5090-0623-6/16/$31.00

filled out from the left of the tree to the right, which can have a significant effect on the shape of the derivation tree.

III. Position Independent Initialisation

Position Independence in Grammatical Evolution is not new. πGE [6] presented an alternative approach to derivation tree construction in GE. In πGE the genotype consists of a list of integer pairs. The first integer of the pair dictates the expansion position in the derivation tree. The second integer dictates the production choice in the traditional GE manner. This approach means that the order of derivation tree expansion is under the control of evolution. This additional degree of freedom was shown to provide πGE with an increase in search space connectivity allowing for easier searching [14]. This mapping process forms the inspiration for the position independent initialisers that follow.

A. PI Grow

Traditional Grow in GP looks to construct a tree that is at most a certain depth. The trees are generally grown in a recursive manner, randomly picking production choices until only leaf nodes remain. If the tree approaches the depth limit then a terminating sequence is selected to make sure the depth limit is not breached [1], [2], [3]. Position Independent Grow (PI Grow) takes inspiration from the πGE mapping process. This study looks to investigate if there is a benefit to generating an initial population in a position independent way.

PI Grow is a non-recursive method of tree growth. The process begins by selecting the start symbol from the grammar and storing it in a queue. This symbol is removed from the queue and a valid production is randomly picked from the grammar. The resulting expansion is then put in the queue in the position the start symbol was removed from. PI Grow now randomly selects a new non-terminal from the queue and again a production is done and the resulting symbols are added to the queue in the position from which the parent node was removed. This process of selecting from the queue continues until no non-terminals remain in the queue. The process of maintaining the order of the queue means the queue represents the lowest current level of the tree. An example of this process is shown in Figure 1.

```
1.            [<e>]
2.            [<e>,<o>,<e>]
3.            [<e>,  -  ,<e>]
4.            [<e>,  -  ,<v>]
5.            [<e>,  -  , Y ]
6.    [<e>,<o>,<e>,  -  , Y ]
7.    [<v>,<o>,<e>,  -  , Y ]
8.    [<v>,<o>,<v>,  -  , Y ]
9.    [<v>,<o>, Y ,  -  , Y ]
10.   [<v>, - , Y ,  -  , Y ]
11.   [ X , - , Y ,  -  , Y ]
```

Fig. 1. Position Independent Derivation String Expansion.

As the tree is being expanded, the algorithm checks to see if the current symbol is the last recursive symbol remaining in the queue. If the depth limit hasn't been reached, and PI Grow currently has the last recursive symbol to expand, Pi Grow will only pick a production that results in recursive symbols. This process guarantees that at least one branch will reach the specified depth limit. This gives the initialiser the freedom to generate trees of both a Full and Grow nature.

IV. Determining Derivation Tree Shape in Grammatical Evolution

There are many ways to describe a derivation tree. The most common way of examining a population of trees seems to come from the easily quantified measures. Tree size, phenotype length, and root bias are all measures that have been used to determine the shape and size of trees in a population [4], [5]. In previous studies three common properties have been used to analyse the tree structures of each individual:

1) **Node count:** Node count is simply the number of nodes in the derivation tree. This gives a measure for the volume of the tree.
2) **Phenotype Length:** The phenotype length gives an indication of how wide the tree is.
3) **Tree Root bias:** Tree root bias is an expression of the percentage of nodes that are to the left or the right of the root node. This indicates if a tree presents a bias to a certain shape.

These measures can indicate the size of the tree, in terms of volume and width, as well as some indication of the shape via the root bias, but fall short of giving a complete picture. To this end the Tree Slope Measure is presented.

A. Tree Slope Measure

The Tree Slope Measure (TSM) takes inspiration from the depth-first order bias and breadth-first order bias measures introduced in [14]. These measures were used to determine if the order of a πGE tree expansion was tending towards either depth-first or breadth-first ordering. These approaches used the level of the leaf nodes of the tree to ascertain certain insights into the derivation order.

The TSM uses the same idea of recording the depths of the leaf nodes to determine a new measure indicating the shape of the derivation tree. TSM first executes a depth-first traversal of the derivation tree and at each leaf node appends the depth of that node to a list. This list is then transformed into an ordered set of [x,index(x)] co-ordinates. A line of best fit is then fitted to these points, and from this line a slope is extracted. This slope provides not only a bias but also a strong indication for the shape of the tree. A tree with a negative slope is said to be right biased and a positive slope indicates a left bias. The greater the slope value the more left-sided or right-sided the tree is.

V. Experimental Setup

The aim of the study is to assess whether the adoption of a position independent approach to initialisation within GE

can improve the structural diversity of the initial population over standard RHH initialisation. A developmental branch of PonyGE [15] was used for all experiments, the parameters of which are listed in Table I.

TABLE I
EXPERIMENTAL SETUP

Parameter	Value Used	
Initialisation Study		
Population	200 runs of 500 Population	
Max initial depth	10	
Ramping	True	
Duplicates	Prevented	
Performance Study	**CFG-GP**	**GE**
Population	500	
Generations	50	
Number of runs	100	
Mutation	Sub-Tree	Per codon
Mutation Rate	1 event per individual	1/length of genome
Crossover Rate	0.5	
Max initial depth	10	
Global depth limit	20	-
Ramping	True	
Duplicates	Prevented in initialised population	
Initialisation tails	-	50%
Tournament size	1%	
Wrapping	Off	
Elitism	1%	

In order to test these approaches, a number of initialisation methods were compared using GE. The six initialisation approaches are outlined below:

1) **Grow:** Grow randomly builds a tree up to a maximum specified depth. If a branch of the tree reaches the imposed depth limit the branch is finished by selecting only terminals. There is no guarantee of the tree reaching the depth limit.
2) **Full:** Full constructs a tree where every branch extends to the set maximum depth. This generally results in very bushy or full trees.
3) **GP RHH:** Ramped Half-and-Half or Sensible Initialisation. Half the population is constructed with Full and the other half with Grow. A ramping of depths is used to produce a varied population of tree sizes and shapes.
4) **PI Random:** PI Random is basically Grow but performed in a position independent way. There is no guarantee of the tree reaching the depth limit.
5) **PI Grow:** PI Grow is explained in Section III.
6) **PI RHH:** PI RHH is similar to GP RHH except it uses PI Grow and Full to construct the population.

A large number of initial populations were generated (i.e. no evolution was performed past generation 0) using two grammar variants. These individuals were then examined to determine the effect of the initialiser on their derivation tree structures.

A. Duplicate individuals and ramping depths

For each initialised population, duplicate individuals are prevented from being generated. An individual solution is considered to be unique if the combination of its phenotype (as a string) and the number and depths of its terminal nodes are unique. The number and depths of an individual's terminal nodes are obtained through a depth-first traversal over the entire tree. Every time a terminal node is encountered its depth is appended to a list. It is the combination of both the phenotype string and the depth list that must be unique for an individual to be considered unique.

In this manner it is possible for unique tree structures which produce the same phenotypic output to be included in the population. This is analogous to many-to-one mapping in GE [8] whereby distinct genotypes can generate the same phenotype. Though their phenotypes may be identical, such individuals can be considered unique as they contain unique genetic material. As such variation operators such as crossover and mutation could produce significantly different results when applied to two such individuals.

Since only unique individuals are permitted in initialised populations, ramping tree depths for large population sizes can cause problems. Consider the binary grammar described in Figure 2. The minimum depth of this grammar is 2, with 2 possible combinations at that depth (`[x, y]`. At a depth of 3, `<e><o><e>` maps to `<v><o><v>`, thus the total number of possible combinations is $2 \times 2 \times 2 = 8$. At a depth of 4 there are 192 combinations, and it is only once the depth passes 5 that the recursive element of the grammar begins to generate a large number of solutions at 81,408.

With a minimum depth of 2 given by this grammar, an initialiser with a population size of 250,000 ramping to a depth of 10 will require 11,111 unique solutions at each ramping depth. Clearly this is not possible at depths 2, 3, and 4, therefore the initialiser can only begin ramping at depth 5 (depths 2-4 are ignored as it is easily possible to enumerate the entire space in this range for this grammar). Thus, ramping between depths 5 and 10 there are 16,666 unique solutions which can be generated at any given depth[1].

B. Grammars

A basic binary grammar (shown in Figure 2) is used to illustrate the effects of initialisation methods. The positioning of operators within the grammar has a notable effect on the shape of the derived tree structures (discussed in Section VI-C). For that reason, all three options are explored in this paper: prefix (where the operator is placed *before* the expressions), infix (where the operator is placed *between* the expressions, as shown in Figure 2), and postfix (where the operator is placed *after* the expressions).

[1]An important point to note is that the grammar shown in Figure 2 is extremely basic and is unlikely to be used in any real application. It is used here purely for illustrative purposes. Even slightly more complex grammars will allow much greater permutations and combinations of unique solutions at lower depths.

```
<e> ::= <e><o><e> | <v>
<v> ::= x | y
<o> ::= + | -
```

Fig. 2. Basic infix binary grammar.

A more complex grammar, shown in Figure 3, was used based on a current real-world application (the grammar has been anonymised). This grammar contains a number of interesting attributes:

- Both infix and pre-fix elements,
- Varying properties at different depths (e.g. the minimum depth of the grammar is 4, but no solutions can be generated at depths 5 or 7),
- A mix of different arities for each production rule, and
- Combinations of recursive and non-recursive non-terminals.

This grammar is comparable to many other real-world problems such as code generation, etc. The combination of these attributes makes for an interesting case study.

C. Performance Comparison

Examining the initial populations provides insights into the starting point of the run. While it might appear that the new approach may deliver an initial population that samples the tree space in a desirable way, this may not translate into the fitness landscape. To ascertain if the changing of initialisation methods has an impact on search performance, GE was applied to three regression benchmark problems [16]:

- Keijzer-6 (K6) [17]: $\sum_i x\frac{1}{i}$;
- Vladislavleva-4 (V4) [18]: $\frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$;
- Dow Chemical Dataset (Dow) [19].

The runs were carried out using the general settings outlined in Table I. The grammar used for all three experiments is presented in Figure 4. Slight changes were made to this grammar to account for the differing number of input variables. The grammar was then balanced to reduce any selection bias.

VI. Results

The results for this study are presented below. Firstly the initialisers are examined in terms of size, shape, and bias (as outlined in Section IV). Results for two grammar variants are presented. Following this the results for the benchmarks are shown before the section finishes with the reporting of the affect of the initialisers on duplicate individuals.

A. Initialisation

1) Binary Grammar: Infix, prefix, and postfix versions of the grammar where used in this study. The results for prefix and post fix have been excluded as they provided 100% bias to each direction respectively. The two grammars where used as controls to test the measures functioned as expected. The results for the infix grammar are shown in the graphs that follow. The graphs are based on the initial population of 200 independent runs, each with a population of 500 unique individuals. All initialisation methods are ramped to the same levels and use the same random seeds.

Figure 5 shows the distribution of root bias in the initialisers. This measure indicates how much of the tree is to the left of the root. The graph clearly shows that Grow and PI Random generate a lot of heavily left-biased trees. This is to be expected as both initialisers are the same and differ only in a slight implementation change. GP RHH can be seen to have a left bias due to its use of Grow, as well as a strong central bias due to the Full initialiser that generates a lot of balanced trees. PI Grow can be seen to sample a wider variety of tree biases than the other methods and tends to a wider centrally biased distribution.

Figure 6 presents the distribution of fitted line slopes for each initialiser. The graph shows that RHH and Pi RHH have a massive bias towards very full trees, e.g. slope 0. This is due to the Full initialisation method that exclusively produces flat bushy trees with an infix binary grammar. Also of note is the two strong spikes on the left and right of the graph indicating a large amount of very left and right biased trees which are generated by the Grow and Pi Random methods.

Figure 7 shows the distribution of solution sizes for each initialiser. Both Grow and PI Random show a similar trend towards generating shorter solutions. This can be linked to both approaches not being required to reach a specified depth. The RHH approaches can be seen to generate clumps of very long individuals (due to Full) and a large amount of shorter individuals (due to Grow). PI Grow however produces tapered distribution of solution up to 200 symbols in length. This indicates a much broader sampling of the initial solution space than any of the other methods.

Finally, Figure 8 shows ramping in action, specifically for PI Grow. It can be seen that each level has very different characteristics in terms of what trees are being sampled. The combination of these ramps lead to a coverage of the space not possible by a single depth limit.

2) Real World Grammar: Following on from the initial study, a complex grammar was selected (Figure 3) in order to examine the resultant initial populations (no evolution was performed). Figure 9 shows the results of this, based on 200 initial populations of 500 trees each, providing a total of 100,000 individuals. The graphs suggest that there is very little observable difference when the grammar contains a mixture of arities and a combination of infix and prefix rules. It can be seen that PI Grow once again is slightly more varied in its population with maximum values being lower and minimum values being higher in terms of slope bias and root bias. PI Grow also displays a slightly smoother distribution of individual sizes.

3) Duplicate Individuals: Within an initial population the aim is that each members of the population represents a unique tree. This criteria being met means that every individual is positioned in its own unique place in the search space from which it can search. Table II displays the number of duplicate

```
<s>         ::= <basic> | <basic> | <basic> | <opt>
<basic>     ::= <expr><conds><expr>
<o>         ::= logical_and((<s>),(<s>)) | logical_or((<s>),(<s>))
<e>         ::= <reg> | <reg> | <reg> | <val><op><val> | <val>
<reg>       ::= (<e><o><e>) | (<e><o><e>) | (<e><o><e>) | (<e><o><e>) |
                <fn>(<e>) | <fn>(<e>) | <fn>(<e>) | <spfn> | <spfn>
<op>       ::= + | - | * | %
<plog>     ::= log(abs(<e>)+1)
<sqrt>     ::= sqrt(abs(<e>))
<spfn>     ::= <plog> | <sqrt>
<fn>       ::= sin | cos | tan
<val>      ::= A | B | C | D | E | F | G | H | I | J | K | <epsilon>
<epsilon> ::= <n><n><n>.<n><n><n>
<n>        ::= 0|1|2|3|4|5|6|7|8|9
<conds>    ::= < | > | ==
```

Fig. 3. Real-World Application Grammar

```
<e> ::= <e> + <e> | <e> - <e>
      | <e> * <e> | <e> / <e>
      | psqrt(<e>) | sin(<e>) | tanh(<e>)
      | exp(<e>) | plog(<e>)
      | x1 | x1
      | <c><c>.<c><c> | <c><c>.<c><c>
<c> ::= 0 | 1 | 2 | 3 | 4 |
        5 | 6 | 7 | 8 | 9
```

Fig. 4. Grammar used for K6 experiments. This grammar formed the basis for the grammar used in the other two problems.

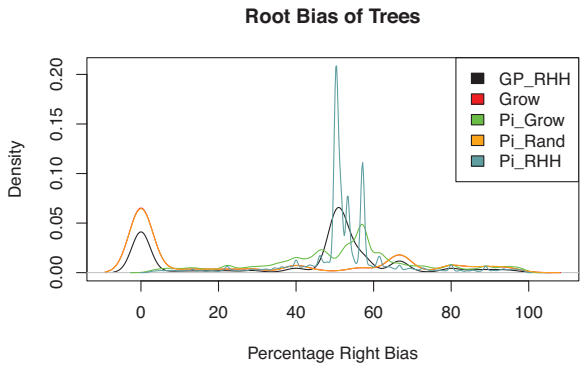

Fig. 5. The graph shows the percentage right bias on the different initialisers on the binary grammar in Figure 2. Zero percent indicates a full left bias in the tree.

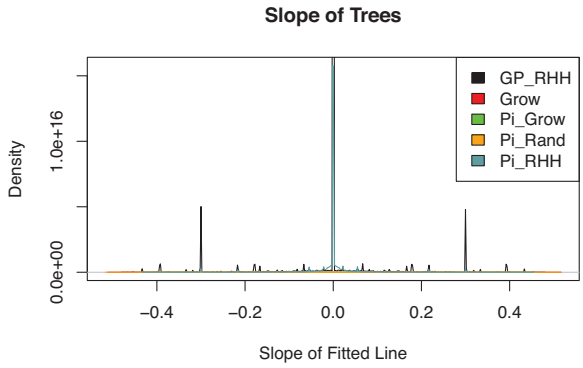

Fig. 6. The graph shows the fitted line slope on the different initialisers on the binary grammar in Figure 2.

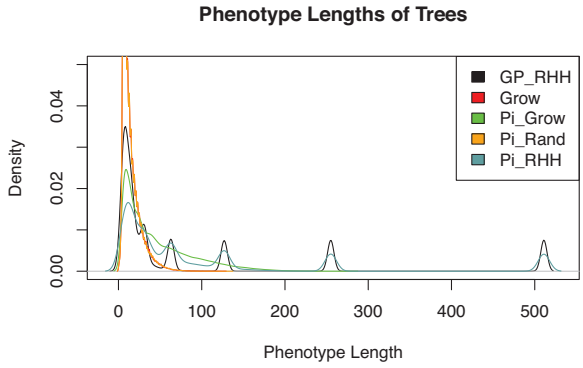

Fig. 7. The graph shows the length of solutions explored using different initialisers on the binary grammar in Figure 2.

individuals encountered at each ramp depth, for a population of 100,000 individuals.

It can be seen from Table II that both traditional Grow and traditional Ramped Half-and-Half generate excessive numbers of duplicate individuals. This is due to the propensity of the original Grow method towards generating small trees. With Full and PI-based methods, the creation of duplicate individuals reduces markedly as the ramping depth increases.

Grow and PI Random show the largest number of duplicates

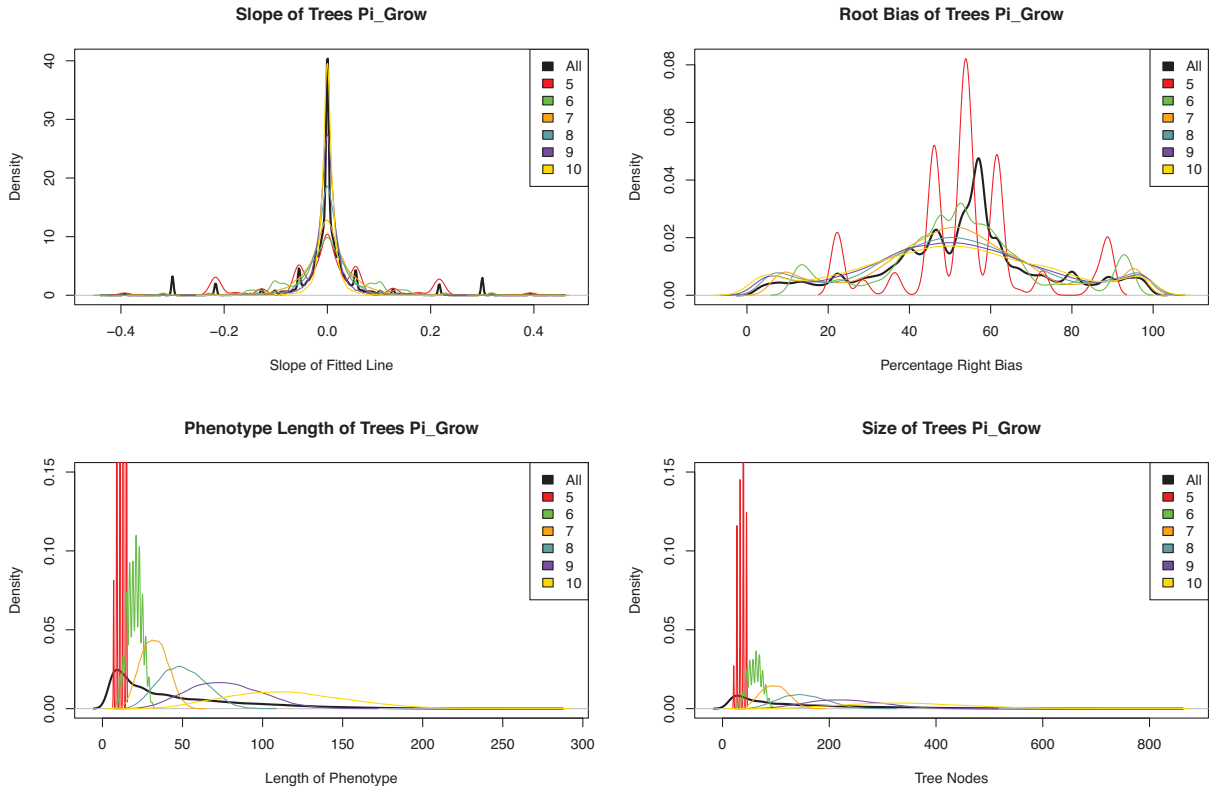

Fig. 8. The graphs show how each ramping depth contributes to the initial population. The graphs are based on the initial population of 200 independent runs (using PI Grow) with a population of 500, using the grammar in Figure 3.

TABLE II
NUMBER OF DUPLICATE INDIVIDUALS PREVENTED PER RAMPING DEPTH, BINARY GRAMMAR.

Ramp depth	RHH	PI Grow	FULL	PI Rand	PI RHH	Grow
4	12,676	3,815	6,472	29,629	4,000	29,543
5	12,434	18	81	29,044	0	28,631
6	13,249	0	5	29,894	0	29,328
7	13,942	0	2	29,499	0	30,070
8	13,558	0	0	30,430	0	30,240
9	13,941	0	0	30,779	0	30,638
10	14,529	0	0	30,863	0	30,988

as both methods are very similar (indeed the graphs for both methods overlap in Figures 5 to 7). The reason for this is that neither method *forces* the individual to a specified depth, but rather randomly builds a tree *up to* a given depth limit. Thus, the vast majority of Grow and PI Random individuals in GE are of very small depths, regardless of the depth limit. Recalling from Section V-A that a very small number of unique solutions exist at the lower depth limits, it is clear how a large number of duplicates can be generated. RHH has on average half the duplicates of Grow, which is to be expected as only half of the individuals are made with Grow. PI RHH on the other hand is seen to benefit from its use of PI Grow as it markedly reduces the number of duplicates seen when compared to traditional RHH.

The duplicate figures quoted do represent an additional computation time that can be saved by choosing different initialisation methods. These figures do see a reduction with the addition of the more complex grammar in Figure 3. The addition in arity in the rules in the grammar do produce a larger space to be sampled from at each depth, resulting in the decrease as shown in Table III.

TABLE III
NUMBER OF DUPLICATE INDIVIDUALS PREVENTED PER RAMPING DEPTH, REAL WORLD GRAMMAR.

Ramp depth	RHH	PI Grow	Grow	PI RHH
4	365	352	365	403
6	68	82	84	54
8	28	0	43	4
9	26	0	33	0
10	23	0	39	0

B. Performance Differential

The use of a new initialiser such as PI Grow may present desirable attributes for an initial population, but such traits

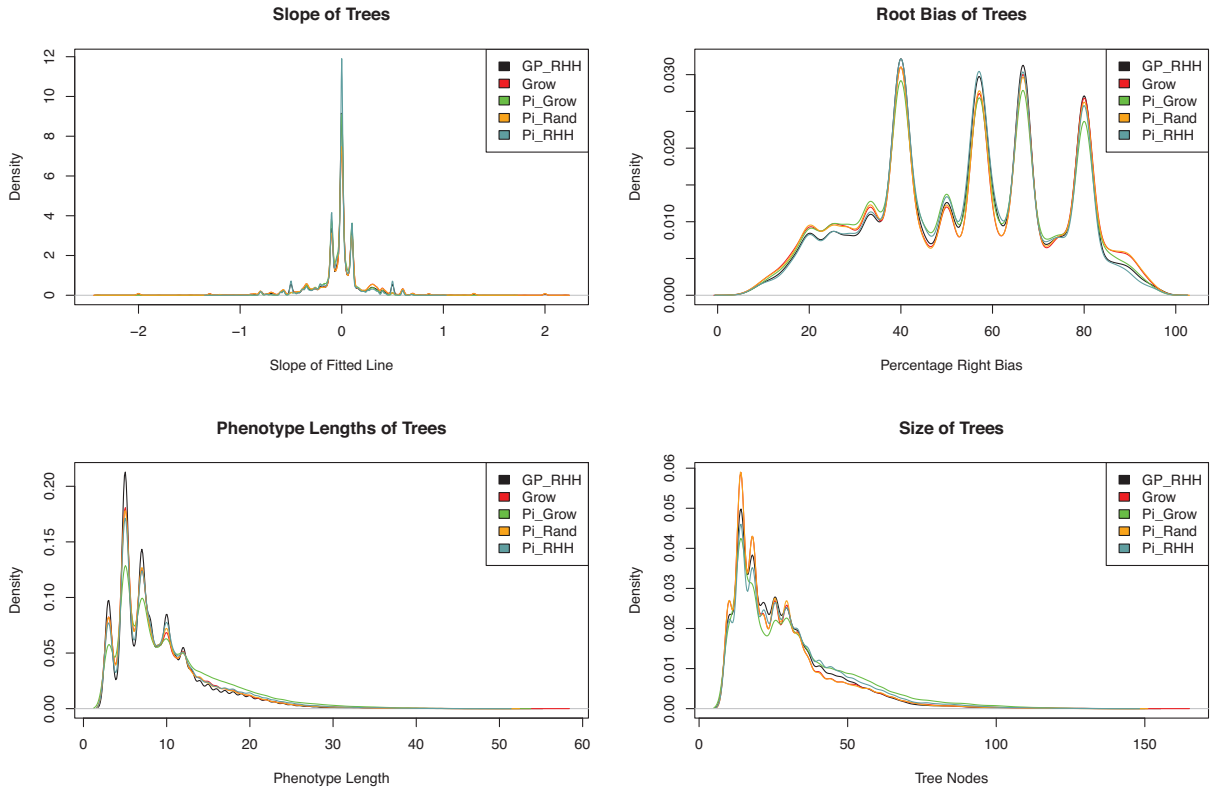

Fig. 9. Shown is a comparison of the initial population of several approaches to initialisation on a real world grammar shown in Figure 3. Displayed is the distribution for each method based on 200 independent populations of 500 individuals. The graphs display slope bias, tree root bias, phenotype length, and tree size.

must not impact the algorithm's performance in terms of finding solutions. The results of the full evolutionary runs using the benchmark experiments described in Section V-C are presented in Table IV.

The results in the table represent the average of the best fitness over 100 independent runs. The runs were performed using both traditional GE linear genomic operations and CFG-GP tree-based operations. From Table IV it can be seen that while there are some differences between the initialisation methods in terms of average performance, these are covered in the standard deviation. The results for K6 suggest that the algorithm is over-fitting for some of the approaches, as the training results (not shown) show a large improvement in performance over test results shown.

C. Discussion

Both traditional GP grow [1], [2] and GE "sensible" grow [3] generate trees *up to* a specified depth, but do not necessarily force any branches of those trees to the specified depth. This allows for trees of varying depths up to and including the ramping depth to be included in the population. This draws parallels with traditional GP sub-tree mutation, whereby a new sub-tree is generated up to a maximum global depth (i.e. the new randomly generated sub-tree can have any depth up to a specified limit). It is therefore interesting to see the differences between traditional GE "sensible" grow and PI Grow. It can be seen from figures 5 and 7 that GE grow favours small trees that are either heavily left or right biased. It can therefore be inferred that GE/CFG-GP sub-tree mutation will also favour heavily left or right biased small tree structures, since the same methods are used to randomly grow trees up to a maximum specified depth. This warrants further investigation.

It must be noted that these results are in some degree skewed due to the imposition of unique individuals on all initialisation methods. Since GE "sensible" grow (and tangentially GE/CFG-GP sub-tree mutation) is biased towards generating smaller trees, preventing the initialiser from generating duplicate individuals at higher ramping levels could *force* the creation of larger tree structures than the grammar would otherwise generate without such an imposition. Table II shows that similar numbers of duplicate individuals are generated regardless of the ramping depth for all methods which randomly grow tree structures (RHH, PI Random, Grow). Only if the method is forced to generate structures to the requisite depth (as seen with Full and PI Grow) will the level of duplicate solutions decrease.

TABLE IV
FITNESS RESULTS FOR INITIALISERS

	CFG-GP				GE			
Name	RHH	Random	PI-Grow	PI-RHH	RHH	Random	PI-Grow	PI-RHH
K6	0.0029 ±0.0037	0.0126 ±0.0785	0.0033 ±0.0087	0.0011 ±0.0020	0.0022 ±0.0032	0.0028 ±0.0095	0.0052 ±0.0101	0.0028 ±0.0036
V4	0.0426 ±0.0111	0.0451 ±0.0209	0.0426 ±0.0113	0.0421 ±0.0103	0.0420 ±0.0084	0.0416 ±0.0087	0.0411 ±0.0067	0.0411 ±0.0091
Dow	0.1096 ±0.0098	0.1108 ±0.0104	0.1085 ±0.0061	0.1100 ±0.0073	0.1124 ±0.0052	0.1129 ±0.0045	0.1120 ±0.0051	0.1120 ±0.0051

VII. CONCLUSION

Position Independent Initialisation was examined and found to present a viable alternative to traditional RHH initialisation methods. PI Grow appears to be the most promising of these approaches. Furthermore, forcing at least one branch to reach each ramp depth has provided an initialiser that explores deeper into the tree search space in terms of size than the traditional Grow method, while not producing very large trees such as with the Full method. The method also has been shown to remove some bias that may be present in GP RHH and Grow. PI Grow also reduced the number of duplicate individuals observed during initialisation. The results do suggest however that grammar design can have a big impact on the performance of these initialisers.

This study has indicated that position independent growth represents a promising method of tree creation. The next avenue of exploration is to investigate incorporating such methods into sub-tree mutation.

ACKNOWLEDGEMENTS

This research is based upon works supported by the Science Foundation Ireland under Grant No. 13/IA/1850.

REFERENCES

[1] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. [Online]. Available: http://mitpress.mit.edu/books/genetic-programming

[2] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008, (With contributions by J. R. Koza). [Online]. Available: http://www.gp-field-guide.org.uk

[3] C. Ryan and R. M. A. Azad, "Sensible Initialisation in Grammatical Evolution," in *Proceedings of the 5rd Annual conference on Genetic and Evolutionary Computation (GECCO 2003)*, 2003, pp. 142–145.

[4] R. Harper, "GE, Explosive Grammars and the Lasting Legacy of Bad Initialisation," in *IEEE Congress on Evolutionary Computation (CEC 2010)*. Barcelona, Spain: IEEE Press, 18-23 Jul. 2010.

[5] E. Murphy, E. Hemberg, M. Nicolau, M. O'Neill, and A. Brabazon, "Grammar Bias and Initialisation in Grammar Based Genetic Programming," in *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, ser. LNCS, A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, Eds., vol. 7244. Malaga, Spain: Springer Verlag, 11-13 Apr. 2012, pp. 85–96.

[6] M. O'Neill, A. Brabazon, M. Nicolau, S. M. Garraghy, and P. Keenan, "pi Grammatical Evolution," in *Genetic and Evolutionary Computation – GECCO-2004, Part II*, ser. Lecture Notes in Computer Science, K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, Eds., vol. 3103. Seattle, WA, USA: Springer-Verlag, 26-30 Jun. 2004, pp. 617–629.

[7] I. Dempsey, M. O'Neill, and A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*, ser. Studies in Computational Intelligence. Springer, 2009. [Online]. Available: http://www.springer.com/engineering/book/978-3-642-00313-4

[8] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, ser. Genetic programming. Kluwer, 2003. [Online]. Available: http://www.wkap.nl/prod/b/1-4020-7444-1

[9] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-Based Genetic Programming: A Survey," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3/4, pp. 365–396, Sep. 2010, tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[10] M. O'Neill, "Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution," Ph.D. dissertation, University Of Limerick, 2001.

[11] P. A. Whigham, "Grammatically-Based Genetic Programming," in *Proceedings of the Workshop on Genetic Programming: from Theory to Real-World Applications*, vol. 16, no. 3, 1995, pp. 33–41.

[12] A. Moraglio, J. McDermott, and M. O'Neill, "Geometric Semantic Grammatical Evolution," in *Semantic Methods in Genetic Programming*, C. Johnson, K. Krawiec, A. Moraglio, and M. O'Neill, Eds., Ljubljana, Slovenia, 13 Sep. 2014, workshop at Parallel Problem Solving from Nature 2014 conference. [Online]. Available: http://www.cs.put.poznan.pl/kkrawiec/smgp2014/uploads/Site/Moraglio.pdf

[13] S. Forstenlechner, M. Nicolau, D. Fagan, and M. O'Neill, "Introducing Semantic-Clustering Selection in Grammatical Evolution," in *GECCO 2015 Semantic Methods in Genetic Programming (SMGP'15) Workshop*, C. Johnson, K. Krawiec, A. Moraglio, and M. O'Neill, Eds. Madrid, Spain: ACM, 11-15 Jul. 2015, pp. 1277–1284. [Online]. Available: http://doi.acm.org/10.1145/2739482.2768502

[14] D. Fagan, "Analysing the Genotype-Phenotype Map in Grammatical Evolution," Ph.D. dissertation, University College Dublin, Ireland, 30 Oct. 2013. [Online]. Available: http://ncra.ucd.ie/papers/DavidFaganPhDThesis2014.pdf

[15] J. McDermott, E. Hemberg, and J. Byrne, "PonyGE," https://github.com/jmmcd/ponyge.git, accessed: 2015-01-12.

[16] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. JaÅŻkowski, U.-M. OâĂŹReilly, and S. Luke, "Better GP Benchmarks: Community Survey Results and Proposals," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 14, no. 1, pp. 3–29, 2013.

[17] M. Keijzer, "Improving Symbolic Regression with Interval Arithmetic and Linear Scaling," in *Genetic Programming, 6th European Conference, EuroGP 2003, Essex, UK, April 14-16, 2003, Proceedings*, ser. Lecture Notes in Computer Science, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610. Springer, 2003, pp. 70–82.

[18] E. J. Vladislavleva, G. F. Smits, and D. den Hertog, "Order of Non-Linearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 333–349, 2009.

[19] C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekart, A. I. Esparcia-AlcÃązar, C.-K. Goh, J. J. Merelo, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis, Eds., *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings*, ser. LNCS, vol. 6024, EvoStar. Springer, 2010.