

Termination in Grammatical Evolution: Grammar Design, Wrapping, and Tails

Miguel Nicolau, Michael O'Neill and Anthony Brabazon
 Natural Computing Research & Applications Group
 University College Dublin
 Ireland

Email: Miguel.Nicolau@ucd.ie, M.ONeill@ucd.ie, Anthony.Brabazon@ucd.ie

Abstract—This paper explores the issues with mapping termination in Grammatical Evolution, and examines approaches that can be used to minimise them. It analyses the traditional approach of reusing the same genetic material, known as wrapping, and shows why this is inefficient with some grammars used in the literature. It suggests the appending of non-coding genetic material to genotype strings, at the start of the run, and shows the benefits of this approach: higher probability of creating terminated individuals, better or similar experimental performance, and a tendency to generate smaller solutions, when compared to the use of wrapping.

I. INTRODUCTION

Grammatical Evolution (GE) [1] is a very popular form of Evolutionary Computation, with a large amount of articles published every year. Its simplicity is remarkable: using any search algorithm, a population of numerical strings is evolved, which can be transformed into syntactically correct solutions for a specific problem, through the use of a grammar. The usage of this grammar allows the system to be applied to a variety of problem domains, hence its popularity.

There is more than meets the eye when it comes to applying GE to a new problem domain, however. One of the main issues concerning its mapping process is that of termination: if a numerical string is too short, it might not complete the mapping of a syntactically correct solution, and the individual becomes invalid. Typically, to minimise this problem, a process called Wrapping [2] is employed, which consists in reusing the same numerical string, until the mapping has terminated.

In this paper, the use of wrapping is analysed, using three benchmark problems, and an alternative technique is proposed. It is shown that wrapping is of limited use, particularly with simpler grammars. More complex grammars do not suffer from this issue, but the usage of wrapping results in a substantial increase in the size of the solutions generated.

An analysis of the non-coding regions of the evolved numerical strings identifies the presence of terminating sequences, subject to proper grammar design. The solution proposed in this paper therefore involves inserting a sequence of random values at the end of the numerical strings used, such that those terminating sequences can proliferate faster.

The results obtained show how effective this simple technique is at maintaining most of the initial populations' validity. As the runs progress, the advantage in performance becomes smaller but still significant, and the size of solutions remains

close to that of GE without wrapping, and substantially smaller than that of GE using wrapping.

This paper is structured as follows. Section II presents GE and its mapping process. Section III analyses the wrapping process, and Section IV analyses the contents of non-coding regions in standard GE. The appending of extra non-coding genetic material is then analysed in Section V, and its effect is compared with standard GE, with and without wrapping. Finally, Section VI runs all three approaches on standard benchmarks, and Section VII concludes this work and draws future work directions.

II. GRAMMATICAL EVOLUTION

Grammatical Evolution (GE) [2], [1] is an evolutionary computation algorithm, similar to Genetic Programming (GP) [3], which is used to evolve functional programs as solutions to given problems.

Unlike GP, which generates populations of syntax trees, GE is based on the biological principle of a genotype-to-phenotype mapping process [4]. It uses an evolutionary algorithm to evolve populations of numerical strings (typically a variable-length Genetic Algorithm (GA) [5]), and then uses a context-free grammar to map them to syntactically correct solutions.

GE performs on par with GP for symbolic regression problems [1], while its grammar provides extra control of the syntax of evolved programs, both in terms of biases [6], [7] and data-structures used. This allows GE to be applied to a variety of problem domains, including Financial Modelling [8], horse gait optimisation [9], wall shear stress analysis in grafted arteries [10], and optimisation of controllers for video-games [11], to name a few.

A. Mapping Process

To illustrate the mapping process employed in GE, consider the following grammar:

```
<e>      ::= <e> <o> <e>
          | <v>
<o>      ::= + | - | * | /
<v>      ::= x | 1.0
```

Grammar 1. Simple arithmetical expressions grammar.

This grammar is composed of three *non-terminal* symbols (<e>, <o> and <v>) and six *terminal* symbols (+, -, *, /, x

and 1.0). Given a genotype string composed of the sequence of integers (also known as codons) (4, 5, 8, 4, 3, 1, 9, 7), a program (phenotype) can be constructed, which respects the syntax specified in the grammar.

This process works by using each codon to choose productions from the grammar, mapping a given start symbol (typically, the first non-terminal symbol defined in the grammar) to a sequence of terminal symbols. In this example, the first codon (4) is used to choose one of the two productions of the start symbol $\langle e \rangle$, through the formula $4\%2 = 0$, i.e. the remainder of the division (modulus) of the codon value by the number of productions of the symbol $\langle e \rangle$. This means that the first (0^{th}) production is chosen, transforming $\langle e \rangle$ into $\langle e \rangle \langle o \rangle \langle e \rangle$, which becomes the mapping string under construction.

The following codon (5) is then used with the leftmost unmapped symbol in the mapping string, so through the formula $5\%2 = 1$ the first symbol $\langle e \rangle$ is replaced by $\langle v \rangle$, and thus the mapping string becomes $\langle v \rangle \langle o \rangle \langle e \rangle$.

The mapping process continues in this fashion, so in the next step the mapping string becomes $x \langle o \rangle \langle e \rangle$ through the formula $8\%2 = 0$, then $x + \langle e \rangle$ through $4\%4 = 0$, and $x + \langle v \rangle$ through $3\%2 = 1$. Finally, the remaining non-terminal symbol is mapped with $1\%2 = 1$, and the final expression becomes $x + 1.0$, which can then be evaluated.

III. WRAPPING

In the mapping example given, not all codons were used to map the phenotype structure. One can therefore divide the genotype string into two parts: an **Effective Part** (4, 5, 8, 4, 3, 1) and a **Non-Coding Tail** (9, 7).

Assuming an initialisation method was used to create the initial population [12], all initial genotype strings will create valid phenotype strings, and in fact will be composed of an effective part only, that is, the whole length of the genotype is used during mapping. However, depending on the search operators used (such as crossover and mutation, when using a GA), the evolved strings may not have enough codons to fully map syntactically valid programs; in the example given, if the codon 5 were mutated to 6, for example, the mapping process would not terminate.

To address this issue, the typical approach originally suggested for GE is to make use of a process called *Wrapping* [2] (although other alternative repairing mechanisms have been suggested [13], wrapping remains the most common). This consists in reusing the same genotype string, up to a pre-specified maximum number of wrapping events, in an attempt to finish the mapping process.

Wrapping does not always work, however. This is because the mapping process can become stuck in a loop, never fully mapping all non-terminal symbols remaining. Ryan et. al [14] have shown this, and suggested an heuristic to help decide when to stop wrapping.

Wrapping is also dependent on grammar design. This section looks at three popular benchmark problems, and highlights how most grammars used in the literature do not work well

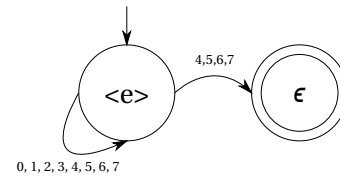


Fig. 1. Simplified NFA representing Grammar 2.

with wrapping; furthermore, designing grammars that do work well with wrapping is not a straightforward process, and quite often only more complex grammars will take advantage of it.

A. Symbolic Regression

Symbolic Regression (SR) is one of the most common applications of GP. The tree representation of GP individuals lends itself to good functional representation and manipulation of sub-expressions, providing solutions that are often very precise, also analysable, hand-tunable, and potentially provable.

Symbolic regression benchmark problems typically involve (re-)discovering a pre-defined function, by trying to match input-output pairs of values. For example, the “Quartic Polynomial” problem (using the function $f(x) = x^4 + x^3 + x^2 + x$) is probably the most used benchmark in GP [15].

Typically, a grammar designed for simple instances of SR combines the four arithmetic operators with the input variable and a constant [7], [16]. Grammar 1 from Section II is an example; this grammar can be reduced to a single non-terminal symbol, while keeping the same production biases (this allows crossover to retain the functionality of exchanged material, and leads to better performance [6]):

```

<e>      ::= <e> + <e>
          | <e> - <e>
          | <e> * <e>
          | <e> / <e>
          | x | x
          | 1.0 | 1.0

```

Grammar 2. Single non-terminal SR grammar.

When using this grammar, the use of wrapping is meaningless. This is because either the expression is fully mapped before using up all the integers, or there are one or more $\langle e \rangle$ symbols left to map, in which case the mapping never terminates.

To help visualise what happens, Fig. 1 shows a simplified Nondeterministic Finite Automaton (NFA), illustrating the mapping process; states are labelled with the next non-terminal symbol to map in the mapping string (or ϵ when there are none left), and transitions are labelled with the production chosen for that symbol (the result of the modulus operation, see Section II-A). Some transitions are similarly labelled, yet leading to different states; this represents the fact that a production can change the current non-terminal symbol to map, causing a transition to a different state, depending on the mapping string.

As can be seen, there are only two states: either there are one or more $\langle e \rangle$ symbols left to map, or the mapping has

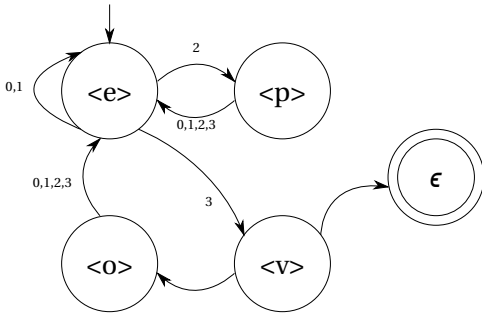


Fig. 2. Simplified NFA representing Grammar 3.

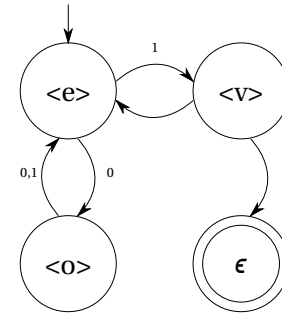


Fig. 3. Simplified NFA representing Grammar 4.

finished. Therefore, a sequence of integers from the genotype will either lead to the termination state, or forever lead back to the $\langle e \rangle$ state. One can therefore conclude that, **with single non-terminal grammars, wrapping never works** (this has been shown previously [14]).

The reduction of the grammar to a single non-terminal symbol has caused wrapping not to work. In the first applications of GE to SR [2], a more complex grammar was used¹ based on the function and terminal sets originally used by Koza [3]:

```

<e> ::= <e> <o> <e>
      | (<e> <o> <e>)
      | <p> (<e>)
      | <v>
<o> ::= + | - | * | /
<p> ::= sin | cos | tan | log
<v> ::= x

```

Grammar 3. Original SR grammar.

As before, a simplified NFA was constructed to illustrate the mapping process; this is shown in Fig. 2. The mapping of the symbol $\langle v \rangle$ to x requires no choice, and as such, no integer from the genotype string is used (as per the original implementation of GE [1]); this is represented by unlabelled transitions leading out of the $\langle v \rangle$ node (an automatic transition to either $\langle o \rangle$ or ϵ).

Once again, one can analyse the effect of wrapping with this NFA. If the mapping does not terminate after all the integers from the genotype string have been read, then the automaton will be in the $\langle e \rangle$, $\langle p \rangle$ or $\langle o \rangle$ states; if the final state is $\langle e \rangle$, then wrapping will not succeed in mapping the individual. If the automaton finished in the $\langle p \rangle$ or $\langle o \rangle$ states, then wrapping can help to terminate the individual. However, if the individual still is not mapped after the first wrapping event, then it will never map through wrapping, as the automaton will become stuck in an endless transition loop.

B. Max Problem

The Max problem [17] is another popular benchmark used with both GP and GE. The objective is to find a program-tree which returns the largest value for a given terminal and

¹Note that this grammar is explosive [7], in that the probability of adding more non-terminal symbols to the mapping string is higher than that of removing them (a balanced grammar should have a 50% probability).

function set with a depth limit D , where the root node counts as depth 0; individuals with a depth larger than D are assigned the worse fitness. Typically, the function set $+, \times$ is used, and there is only one constant, 0.5. This is an important benchmark, as it helps to analyse the behaviour of genetic operators, such as crossover in GP [18] and mutation in GE [19].

A grammar typically used for the Max problem [19] is:

```

<e> ::= <o> <e> <e>
      | <v>
<o> ::= + | *
<v> ::= 0.5

```

Grammar 4. Typical MAX grammar.

This grammar is quite similar to Grammar 3, and suffers the same problem with respect to wrapping; its simplified NFA is shown in Fig. 3. This diagram shows that, if the mapping does not terminate after all the integers have been read, then the automaton is either on the $\langle e \rangle$ or $\langle o \rangle$ states. As such, only if the $\langle o \rangle$ state is reached, can wrapping help with termination. If the mapping does not terminate after a first wrapping event, however, it will never terminate, as the automaton will be either back to the $\langle e \rangle$ state, or stuck in a transition loop always leading back to the $\langle o \rangle$ state.

C. Santa Fe Ant Trail

The Santa Fe Ant Trail (SF) is a typical benchmark [3] used in GP. The problem can be described as follows: in a toroidal grid of 32×32 squares, a trail of food pellets is placed, with certain gaps. The objective is to evolve a computer program that controls an artificial ant throughout the grid; the ant can turn left or right, check if there is food in the direction it is facing, and move in that direction.

The original grammar used with GE is as follows [1]:

```

<code> ::= <line> | <code> <line>
<line> ::= <if> | <op>
<if> ::= if_food() {<line>} else {<line>}
<op> ::= left(); | right(); | move();

```

Grammar 5. Original SF grammar.

A simplified NFA is no longer a sufficient tool to analyse this grammar in terms of wrapping; experimentation (not shown)

TABLE I
EXPERIMENTAL SETUP

Population Size	500
Max Generations	50
Derivation-tree Min/Max Depth (for initialisation)	5/10
Selection Tournament Size	1%
Elitism (for generational replacement)	1%
Crossover Ratio	50%
Average Mutation Events per Individual	1
Max Wrapping events	0

has concluded that it can occur a maximum of six times. It has however been shown [20] that this grammar defines a substantially different search space of the original Santa Fe problem for GP, and commands such as `if_food(){ move(); left(); } else { right(); }` cannot be generated. Harper [7] suggested using the following grammar:

```
<code> ::= <expr> | <code> <line>
<expr> ::= <line> | <line> <expr>
<line> ::= if_food(){<expr>} else {<expr>}
         | <op>
<op>   ::= left(); | right(); | move();
```

Grammar 6. Corrected SF grammar.

This grammar defines a language generating the same expressions as Koza’s initial experiments with GP, and is also wrapping-friendly (experimentation measured up to 30 wrapping events in a single genotype string, ending up in a successful mapping process).

IV. NON-CODING TAILS

This section examines the non-coding regions of individuals in standard GE. More specifically, it analyses the contents of the non-coding tails of individuals after evolution, using the same problems discussed in the previous section, but with wrapping turned off.

A. Symbolic Regression

To analyse the contents of non-coding tails, a full SR experiment was performed, using Grammar 2 (see Section III-A). The experimental setup used is shown in Table I, and the problem used was the quartic polynomial, with 20 samples over the interval $[-1.0, +1.0]$.

50 runs were performed, which were all successful. After a solution was found, each run was stopped, and each codon from each individual was identified as a *Producer* or *Consumer* (using the definition originally suggested by Ryan et. al[14]), depending on whether the formula `codon % 8` was lower than 4 (meaning the codon will choose a recursive rule from the grammar) or not (opposite case), respectively.

The results obtained show that 55.375% of the effective part was made up of consumers, while for the tails that proportion went up to 70.177% (std. dev. of 0.036 and 0.114 respectively). This means that the non-coding section of individuals is not random; it contains **terminating sequences**, which contribute towards their termination.

B. Max Problem

While single non-terminal grammars allow individuals to make use of tails as terminating sequences, with more complex grammars that is not necessarily the case. This is because the functionality of a codon is now dependent on the context upon which it is interpreted (with single non-terminal grammars, a codon will always choose the same production, regardless of the context upon which it is interpreted).

To exemplify this, an experiment similar to the previous section was performed, this time using the Max problem, with depth 4 (see Section III-A), and Grammar 4 was used. The formula `codon % 2` was used to identify producers and consumers, with producers being codons with odd values (and vice-versa for consumers), as in the `<e>` rule only the first production is recursive.

The results obtained show only $43.729\% \pm 0.070$ consumers in the effective parts, and $51.999\% \pm 0.076$ in the tails. This is because the function of a codon depends on the context upon which it is interpreted; with the grammar used, this means that even codon values can choose the transformations `<e> -> <o> <e> <e>` or `<o> -> +`, depending on whether they are used to map the symbol `<e>` or `<o>`. As perfect solutions for this problem consist of one multiplication and 14 additions [18], this explains the proliferation of (potential) producers, especially in the effective part.

To address this problem, the same experiment was performed, but this time using the following grammar, which specifies the same syntax, but decouples the choices of the `<e>` expansion from those of the `<o>` symbol:

```
<e>   ::= <o> <e> <e> | <o> <e> <e>
         | <v> | <v>
<o>   ::= + | *
<v>   ::= 0.5
```

Grammar 7. Corrected MAX grammar.

The results obtained show a rather different $53.591\% \pm 0.0622$ proportion of consumers in the effective part, and $58.661\% \pm 0.0551$ in the tails. This is because a codon can now be both a consumer for the `<e>` symbol and a selector of the `<o> -> +` transformation. This allows individuals to choose more additions, while maintaining the ability to terminate mapping, and has a substantial impact on performance: with Grammar 4, 22/50 runs were successful, whereas with Grammar 7 there were 35/50 successful runs. This shows the **importance of grammar design on the termination of individuals**, even when wrapping is not used.

C. Santa Fe Ant Trail

When more complex grammars are used, the exchange of terminating sequences in non-coding tails becomes less evident. That is the case with the Santa Fe problem; using the same experimental setup as in the previous sections, and Grammar 5 (with the symbol `<code>` representing the consumer/producer ratio), there was a proportion of $43.372\% \pm 0.047$ consumers in the effective regions, and a very similar

43.827% \pm 0.048 in the tails. The higher rate of consumers is linked once again to grammar design: a codon that transforms the `<code>` symbol into `<line>` (a consumer) will also transform `<line>` into `<if>`, which is a vital transformation for a fit solution to this problem.

Grammar 6 is equally unable to keep terminating sequences in the tails, with ratios of 54.309% \pm 0.037 consumers in the effective part, and 53.332% \pm 0.046 in tails. This is because odd codon values are consumers for both the `<code>` and `<expr>` symbols, but they act as producers for `<line>`. In fact, both Grammar 5 and Grammar 6 are explosive [7], and tails are not as effective at controlling this growth.

V. ARTIFICIAL TAILS

The previous section highlighted the importance of non-coding tails in GE, as a means to help termination of individuals. One can envisage increasing their influence, by artificially appending non-coding codons at the end of genotype strings. This cannot be done throughout the whole run, however, as the constant insertion of genetic material could negatively impact the size of individuals. Furthermore, as seen in the previous section, the contents of non-coding tails evolve, leading to the creation of terminating sequences; a constant adding and pruning of tails would have a negative impact on this process.

It was therefore decided to add tails to individuals only after initialisation. This can be done in many ways; in this study, after all individuals have been (validly) initialised, the average individual size is calculated, and then a percentage of that size is added to each individual, as a tail of random codons.

A. Mapping Performance

To analyse and compare the performance of this approach with wrapping, an experiment was conducted, to measure how many individuals remain valid (i.e. still map to a valid phenotype) after applying standard genetic operators (that is, 1-point crossover and integer mutation, as they are typically applied to GE [1]), and to study how this scenario changes when wrapping or artificial tails are employed.

To this end, populations of 10000 valid individuals were generated using a ramped half-and-half initialisation [3] adapted for GE [12], with minimum and maximum derivation tree depths of 5 and 10 respectively. The fitness of individuals was set to the same value, and the number of valid individuals after applying one generational round of search operators was recorded, when using wrapping or artificial tails (or none).

To measure the effect of genetic operators on the mapping probability of GE, three setups were used. To test the crossover operator, its probability was set to 1.0, and no mutation was used. To test integer mutation, crossover was turned off, and the mutation operator was slightly changed, such that each individual will have one (and only one) codon changed (replaced by a new random integer): the choice of the mutated codon was random, but it always occurred within the mapping section of the genome, even when artificial tails were used; neutral mutations were discarded. Finally, to test

TABLE II
PERCENTAGE OF MAPPABLE INDIVIDUALS CREATED ON THE FIRST GENERATION, USING CROSSOVER ONLY, MUTATION ONLY, OR BOTH COMBINED, USING AN INITIAL POPULATION OF 10000 INDIVIDUALS; RESULTS AVERAGED OVER 50 RUNS

Prob.	Grammar	Wraps	Tail	Valid phenotypes generated		
				Cross.	Mut.	Both
SR	2	0	0%	60.36%	72.99%	60.65%
		100	0%	60.36%	72.99%	60.65%
		0	100%	91.32%	93.82%	90.01%
MAX	7	0	0%	37.97%	12.22%	41.58%
		100	0%	57.28%	85.74%	57.79%
		0	100%	86.75%	92.61%	85.25%
SF	6	0	0%	40.08%	67.92%	33.60%
		100	0%	64.47%	82.32%	57.22%
		0	100%	51.01%	74.68%	44.37%

both operators, one crossover event and one mutation event were allowed per individual.

Table II shows the results obtained for the three benchmark problems examined previously. For each problem, the best performing grammar was chosen (see Section III). Column *Wraps* indicates the maximum allowed number of wrapping events when mapping, and *Tail* indicates the percentage of the average population size added as a random artificial tail, after initialisation. All values are averaged across 50 runs (standard deviations, not reported, were all under 0.001).

1) *Symbolic Regression*: The results show that standard GE can generate up to 40% illegal individuals, after application of genetic operators on the first generation. This is unaltered with the use of wrapping, as the grammar used is composed of a single non-terminal symbol. Artificial tails, although randomly created, aid in the termination of individuals, reducing the generation of illegal individuals down to less than 10%.

2) *Max Problem*: The performance of standard GE with the grammar used for this problem is even worse, particularly when using mutation: over 87% of generated individuals are invalid. This is because most of the mapping derivation choices when using this grammar are applied to the `<e>` symbol (see Grammar 7), leading very frequently to longer derivations, which have no codons left to terminate.

The use of wrapping helps substantially to reduce this effect, particularly when mutation is used. However, artificial tails prove to be the best choice once again, with less than 15% illegal individuals generated after the application of any genetic operator (or combination of both).

3) *Santa Fe Ant Trail*: The problem with explosive grammars can be clearly seen in the SF results, where standard GE also struggles to terminate individuals after the application of search operators. As the grammar used is wrapping-friendly (see Section III-C), successive wrapping events (up to 100) can try to terminate the mapping of individuals (albeit at the expense of generating much larger individuals).

Artificial tails are not as beneficial in this case. Given that the grammar is explosive, and the contents of tails are random,

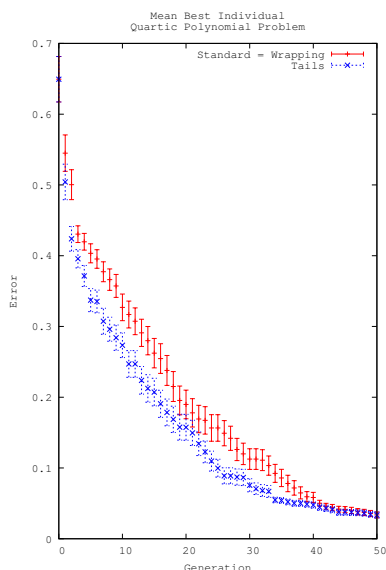


Fig. 4. Mean best individual per generation for the Quartic Polynomial problem. All results are averaged across 50 runs; error bars plot standard deviation.

there is a higher probability of continuing to choose recursive rules from the grammar, rather than consuming productions.

VI. RUN PERFORMANCE

In order to observe the impact of wrapping and artificial tails on the performance of GE, a series of experiments were performed, using the same problems used throughout this paper. The experimental setup is the same as in Table I, but a population size of just 100 individuals was used, to render the problems harder, and highlight the issue of termination.

A. Symbolic Regression

The Quartic Polynomial was used as the target function for the SR problem, using Grammar 2. Fig. 4 shows the average best individual results, averaged over 50 independent runs. Note that the results with and without wrapping are equal, because the grammar used does not allow wrapping events to terminate the mapping of individuals (see Section III-A).

The results show a small improvement in performance when using artificial tails, particularly in the first 40 generations; this is when more illegal individuals are generated with standard GE, and the appending of non-coding tails at the start of the run provides extra buffer genetic material, allowing more individuals to be valid at the earlier generations, and thus improve performance. Towards the end of the run, standard GE evolves its own terminating sequences in tails (see Section IV), and the difference in performance becomes smaller.

Fig. 5 shows the average genotype size of individuals in the population, along with the average effective size. It shows that despite the appending of extra genetic material, the effective size of solutions with and without artificial tails remains

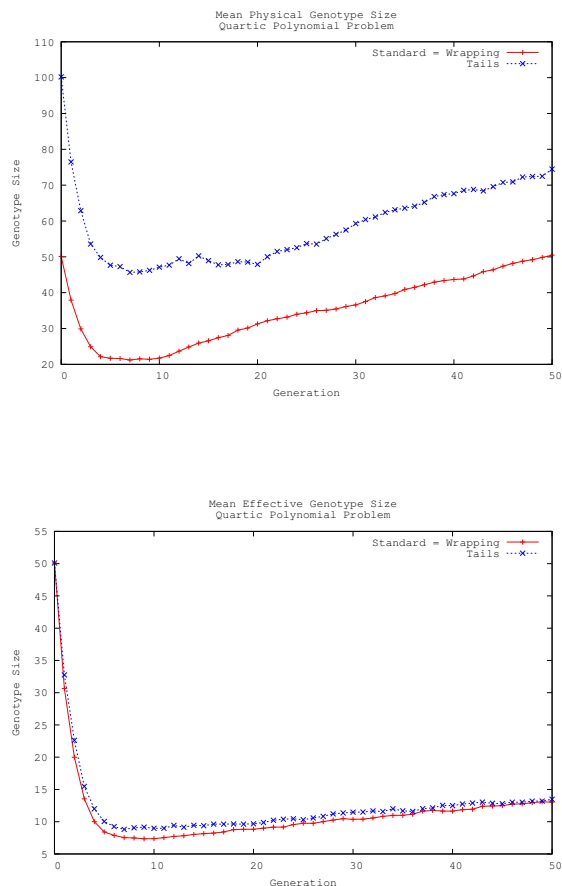


Fig. 5. Average physical and effective size of genotype strings in the whole population, for the Quartic Polynomial problem (averaged across 50 runs).

very similar, and the full physical lengths of genotype strings progress at a similar rate of growth.

B. Max Problem

Grammar 7 was used to evolve individuals for the Max problem, with depth of 4; Fig. 6 shows the results obtained. These show a substantial increase in performance over standard GE, when either wrapping or artificial tails are employed. The performance between the latter two is similar.

The analysis of the raw genotype sizes shown in Fig. 7 shows that wrapping generates a similar amount of genetic material as standard GE, while the extra genetic material used with the tails approach can be observed; the ratio of growth of raw material is very similar across all approaches.

The effective genotype sizes tell a much different story. Wrapping is allowed to generate much larger solutions, which only across time slowly reduce their size. The tails approach generates very similar solution sizes to standard GE.

C. Santa Fe Ant Trail

Finally, the results for the SF problem (using Grammar 6) are shown in Fig. 8. The use of a more complex (and

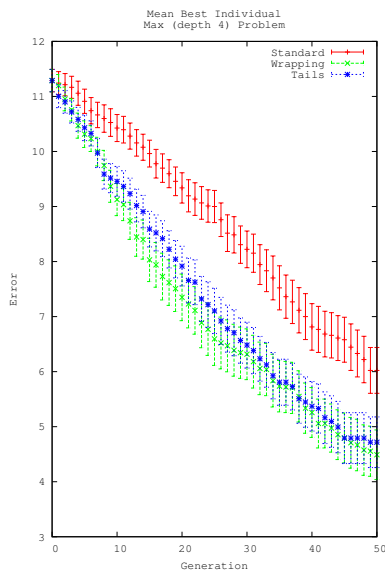


Fig. 6. Mean best individual per generation for the Max (depth 4) problem. All results are averaged across 50 runs; error bars plot standard deviation.

explosive) grammar has rendered the results similar across all approaches, with very small gaps of generations without overlapping std. dev. bars.

It is quite interesting to observe the evolution of genotype sizes, however. Once again, the standard GE and tails approaches are quite similar: the extra genetic material is clearly seen, but their ratio of growth is the same. The use of wrapping with a wrapping-friendly grammar, however, means that solutions can grow without the need of physical genotype growth.

This leads to unbounded solution growth, as can be seen in the effective size plot. With solution sizes of 1000 codons and larger, this means that the wrapping approach is actually evolving the full Santa Fe trail, rather than a program to iterate over a for loop, which is the objective of the benchmark. This occurs with no observable gain in fitness.

VII. CONCLUSION

This paper analysed the problem of termination in GE, and approaches to minimise it. The traditional approach of wrapping, allowing a genotype string to reuse its genetic material, is of limited benefit when using specific grammars from the literature; this highlights the importance of grammar design on the termination of individuals. With better designed grammars, it is better able at minimising the termination problem, but at the expense of generating much larger solutions.

The appending of extra non-coding tails to individuals after initialisation seems to offer a better approach. Although also dependent on good grammar design, it is more forgiving, and the results obtained show that it provides as good or better results than wrapping, while keeping the size of generated solutions much smaller.

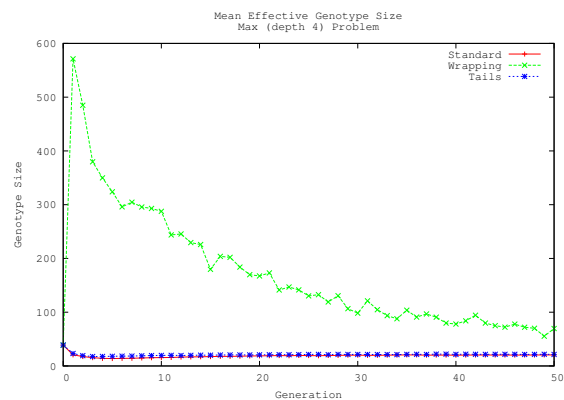
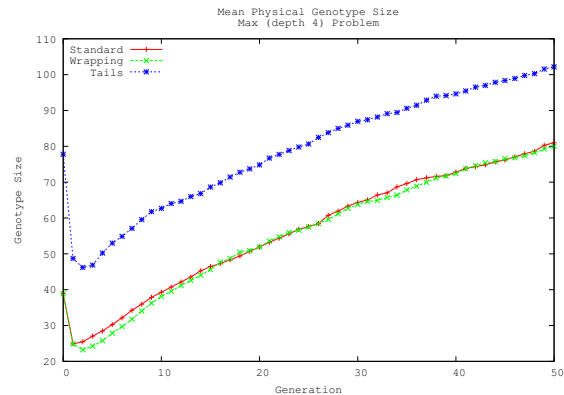


Fig. 7. Average physical and effective size of genotype strings in the whole population, for the Max (depth 4) problem (averaged across 50 runs).

Artificial tails are also able to reduce the generation of illegal individuals, i.e. genotype strings that are unable to terminate mapping, particularly at the start of a run. This is quite important, in order not to lose diversity.

Future work will investigate this effect on harder problems, where diversity in the early populations is more of a determining factor. In need of a deeper study is also which size to use for the artificial tails. Finally, the use of a different initialisation method such as PTC2 [21] substantially improves the performance of GE [7]; with the use of artificial tails and carefully designed grammars, the quality of initial solutions from the initialisation process should propagate better through the evolutionary process.

REFERENCES

- [1] M. O'Neill and C. Ryan, *Grammatical Evolution - Evolutionary Automatic Programming in an Arbitrary Language*, ser. Genetic Programming. Kluwer Academic, 2003, vol. 4.
- [2] C. Ryan, J. Collins, and M. O'Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming, European Workshop, EuroGP 1998*, ser. LNCS, Wolfgang Banzhaf et al., Ed., vol. 1391. Springer, 1998, pp. 83–95.

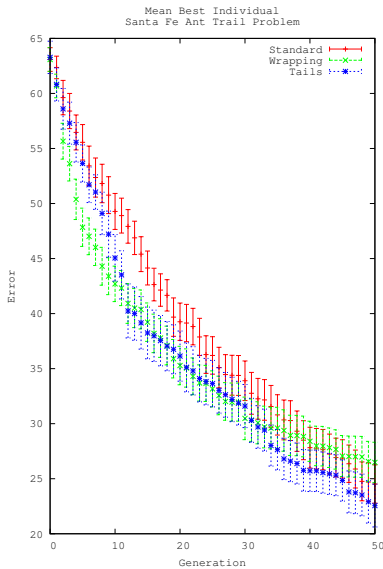


Fig. 8. Mean best individual per generation for the Santa Fe problem. All results are averaged across 50 runs; error bars plot standard deviation.

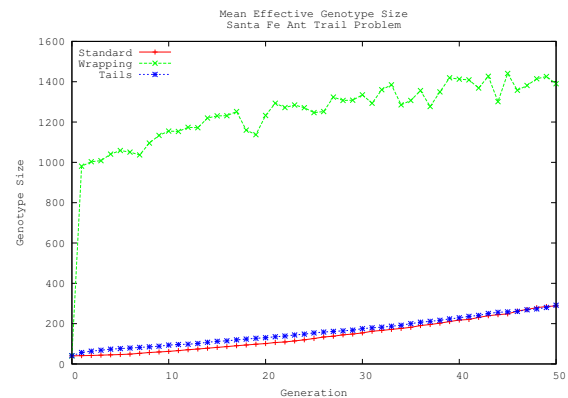
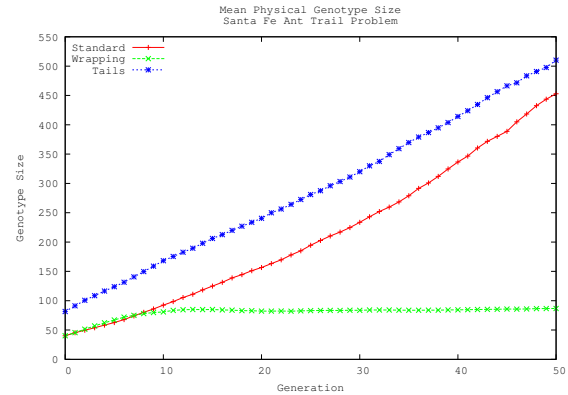


Fig. 9. Average physical and effective size of genotype strings in the whole population, for the Santa Fe problem (averaged across 50 runs).

- [3] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [4] W. Banzhaf, "Genotype-phenotype-mapping and neutral variation – A case study in genetic programming," in *Parallel Problem Solving from Nature - PPSN III, 3rd International Conference, Jerusalem, October 9-14, 1994, Proceedings*, ser. Lecture Notes in Computer Science, Yuval Davidor et al., Ed., vol. 866. Springer-Verlag, 1994, pp. 322–332.
- [5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [6] M. Nicolau, "Automatic grammar complexity reduction in grammatical evolution," in *Genetic and Evolutionary Computation Conference (GECCO) Workshops*, Riccardo Poli et al., Ed. AAAI, 2004.
- [7] R. Harper, "GE, explosive grammars and the lasting legacy of bad initialisation," in *IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, July 18-23, 2010, Proceedings*. IEEE Press, 2010, pp. 2602–2609.
- [8] A. Brabazon and M. O'Neill, *Biologically Inspired Algorithms for Financial Modelling*. Springer, 2006.
- [9] J. E. Murphy, M. O'Neill, and H. Carr, "Exploring grammatical evolution for horse gait optimisation," in *Genetic Programming, European Conference, EuroGP 2009*, ser. LNCS, Mario Giacobini et al., Ed., vol. 5484. Springer, 2009, pp. 579–584.
- [10] R. M. A. Azad, A. R. Ansari, C. Ryan, M. Walsh, and T. McGloughlin, "An evolutionary approach to wall shear stress prediction in a grafted artery," *Applied Soft Computing*, vol. 4, no. 2, pp. 139–148, 2004.
- [11] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving behaviour trees for the mario ai competition using grammatical evolution," in *EvoApplications 2011, Proceedings, Part I*, ser. LNCS, Cecilia Di Chio et al., Ed., vol. 6624. Springer, 2011, pp. 123–132.
- [12] C. Ryan and A. Azad, "Sensible initialisation in grammatical evolution," in *Genetic and Evolutionary Computation Conference (GECCO) Workshops*, Erick Cantú-Paz et al., Ed. AAAI, 2003.
- [13] M. Hemberg and U.-M. O'Reilly, "Extending grammatical evolution to evolve digital surfaces with genr8," in *Genetic Programming, European Conference, EuroGP 2004*, ser. Lecture Notes in Computer Science, Maarten Keijzer et al., Ed., vol. 3003. Springer, 2004, pp. 299–308.
- [14] C. Ryan, M. Keijzer, and M. Nicolau, "On the avoidance of fruitless wraps in grammatical evolution," in *Genetic and Evolutionary Computation Conference (GECCO)*, ser. Lecture Notes in Computer Science, E. C.-P. et al., Ed., vol. 2724. Springer, 2003, pp. 1752–1763.
- [15] D. Costelloe and C. Ryan, "On improving generalisation in genetic programming," in *Genetic Programming, European Conference, EuroGP*

- 2009, ser. LNCS, Mario Giacobini et al., Ed., vol. 5484. Springer, 2009, pp. 61–72.
- [16] E. Murphy, M. O'Neill, and A. Brabazon, "Examining mutation landscapes in grammar based genetic programming," in *Genetic Programming, European Conference, EuroGP 2011*, ser. Lecture Notes in Computer Science, S. S. et al., Ed., vol. 6621. Springer, 2011, pp. 130–141.
- [17] C. Gathercole and P. Ross, "An adverse interaction between crossover and restricted tree depth in genetic programming," in *Genetic Programming 1996: First Annual Conference, Stanford, USA, July 28-31, 1996, Proceedings*, J. R. K. et al., Ed. MIT Press, 1996, pp. 291–296.
- [18] W. B. Langdon and R. Poli, "An analysis of the max problem in genetic programming," in *Genetic Programming 1997: Second Annual Conference, Stanford, USA, July 13-16, 1997, Proceedings*, J. R. K. et al., Ed. Morgan Kaufmann, 1997, pp. 222–230.
- [19] J. Byrne, M. O'Neill, J. McDermott, and A. Brabazon, "An analysis of the behaviour of mutation in grammatical evolution," in *Genetic Programming, European Conference, EuroGP 2010*, ser. Lecture Notes in Computer Science, A. I. Esparcia-Alcázar and A. Ekárt, Eds., vol. 6021. Springer, 2010, pp. 14–25.
- [20] D. Robilliard, S. Mahler, D. Verhaghe, and C. Fonlupt, "Santa fe trail hazards," in *7th International Conference, Evolution Artificielle, EA 2005, Lille, France, October 26-28, 2005, Proceedings*, ser. Lecture Notes in Computer Science, E.-G. T. et al., Ed., vol. 3871. Springer, 2005, pp. 1–12.
- [21] S. Luke, "Two fast tree-creation algorithms for genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 274–283, 2000.