meta-Grammar Constant Creation with Grammatical Evolution by Grammatical Evolution

Ian Dempsey
University of Limerick
Pipeline Trading Systems
New York
ian.dempsey@gmail.com

Michael O'Neill
Dept. of Computer Science
University of Limerick
Limerick, Ireland
michael.oneill@ul.ie

Anthony Brabazon
Faculty of Commerce
University College Dublin
Dublin, Ireland
anthony.brabazon@ucd.ie

ABSTRACT

This study examines the utility of meta-grammar constant generation on a series of benchmark problems. The performance of the meta-grammar approach is compared to a grammar which incorporates grammatical ephemeral random constants, digit concatenation, and an expression based approach. It is found that the meta-grammar approach to constant creation is particularly beneficial on the dynamic problem instances in terms of the best fitness values achieved.

Categories and Subject Descriptors

I.2.0 [**Computing Methodologies**]: Artificial Intelligence—*General*

General Terms

Algorithms, Theory

Keywords

Constant Creation, Digit Concatenation, Ephemeral Random Constants, Genetic Programming, Grammatical Evolution, meta-Grammars

1. INTRODUCTION

Many applications of Genetic Programming require the generation of constants, hence the discovery of efficient means of generating diverse constants is important. The current standard approach to constant generation is to use ephemeral random constants, whose values are created randomly within a pre-specified range at the initialisation of a run [4]. These values are then fixed throughout a run, and new constants can only be created through combinations of these values and other items from the function and terminal set, such as +,-, * and /.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

In earlier studies, a digit concatenation approach to constant creation in Grammatical Evolution has been adopted and some investigations into its utility have been conducted [1, 2, 3]. The findings of these studies provide evidence to support the superiority of the digit concatenation approach across a range of constant creation problems, when compared to an expression-based method in which arithmetic operators are required to generate new constants. More recent work has established that the introduction of grammatical ephemeral random constants can improve the expression based approach in some instances [11]. We extend these studies with the introduction of a meta-Grammar approach to constant creation based on Grammatical Evolution by Grammatical Evolution ($(GE)^2$) [10]. A meta-Grammar is employed in a diploid chromsomal structure where one chromosome describes the solution as usual, the second chromosome is the individuals own grammar which is used to map the solution chromosome and the meta-Grammar is used to map the grammar chromosome for each individual.

This contribution is organised as follows. Section 2 provides a short introduction to Grammatical Evolution and the meta-Grammar approach adopted in Grammatical Evolution by Grammatical Evolution. Section 3 describes the problem domains examined, and the experimental approach adopted in this study. Section 4 provides the results, and finally, conclusions and an outline of future work are provided in Section 5.

2. GRAMMATICAL EVOLUTION

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [5, 6, 7, 8, 9], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [4, 12, 13, 14, 15], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct, and as such it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences.

BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals (the primitive symbols that can be used to construct the output program or sentence(s)), according to the production rules. A simple example BNF grammar is given below, where <expr> is the start symbol from which all programs are generated. The grammar states that <expr> can be replaced with either one of <expr><op><expr> or <var>. An <op> can become either +, -, or *, and a <var> can become either x, or y.

```
<expr> ::= <expr><op><expr> | <var>
  <op> ::= + | - | *
 <var> ::= x | y
```

The grammar is used in a developmental process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar. In order to select a production rule in GE, the next codon value on the genome is read, interpreted, and placed in the following formula:

$$Rule = Codon\ Value\ \%\ Num.\ Rules$$

where % represents the modulus operator.

Beginning from the left hand side of the genome codon integer values are generated and used to select appropriate rules for the left-most non-terminal in the developing program from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar. (b) The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during the mapping process of this individual. (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value. A full description of GE can be found in [5].

2.1 Grammatical Evolution by Grammatical Evolution

In order to allow evolution of a grammar, grammatical evolution by grammatical evolution $(GE)^2$, we must provide a grammar (meta-Grammar) to specify the form a grammar can take. This is an example of the richness of the expressiveness of grammars that make the GE approach so powerful. See [5, 21] for further examples of what can be achieved with grammars. By allowing an evolutionary algorithm to adapt its representation (in this case through evolution a grammar) it provides the population with a mechanism to survive in dynamic environments, in particular, and also to automatically incorporate biases into the search process.

In this approach we therefore have two distinct grammars, the *universal grammar* (or grammars' grammar) and the *solution grammar*. The notion of a universal grammar is adopted from linguistics and refers to a universal set of syntactic rules that hold for spoken languages [22]. It is proposed that during a child's development the universal grammar undergoes modifications through learning that allows the development of communication in their parents native language(s) [23].

In the meta-grammar method, the universal grammar dictates the construction of the solution grammar. Given below are the examples of these grammars for solutions that generate expressions, which could be used for symbolic regression type problems.

Universal Grammar (Grammars' Grammar)

```
<g>::="<expr>::=<op><expr><expr> | <var>"
      "<op>::="<ops>
      "<var>::="<vars>
<ops>::=<opt> "|" <ops> | <opt>
<opt>::= + | - | * | /
<vars>::= <vart> "|" <vars> | <vart>
<vart>::= m | v | q | a
```

Solution Grammar

```
<expr>::=<opt><expr><expr> | <var>
<op>::= ?
<var>::= ?
```

In the universal grammar above, a grammar <g>, is specified such that it is possible for the non-terminals <var> and <op> to have one or more rules, with the potential of rule duplication. These are the rules that will be made available to an individual during mapping, and this effectively allows bias for symbols to be subjected to the processes of evolution. The productions <vars> and <ops> in the universal grammar are strictly non-terminals, and do not appear in the solution grammar. Instead they are interim values used when producing the solution grammar for an individual.

The hard-coded aspect of the solution grammar can be seen in the example above with the rules for <op> and <var> as yet unspecified. In this case we have restricted evolution to occur only on the number of productions for <var> and <op>, although it would be possible to evolve the rules for <expr> and even for the entire grammar itself.

In this study two separate, variable-length, genotypic chromosomes were used, the first chromosome to generate the solution grammar from the universal grammar and the second chromosome to generate the solution itself. Crossover operates between homologous chromosomes, that is, the solution grammar chromosome from the first parent recombines with the solution grammar chromosome from the second parent, with the same occurring for the solution chromosomes. In order for evolution to be successful it must co-evolve both the genetic code and the structure of solutions based on the evolved genetic code.

2.2 Constant Creation Grammars

Two grammars are examined in this study. The first is a combination grammar which includes three constant generation techniques, with the grammar adopted provided below.

```
<exp>::= <trad> | <catR> | <ephemeral>

<trad> ::= <trad> <op> <trad> | <tradT>
<tradT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<op> ::= + | - | / | *

<catR> ::= <cat> <dot> <cat> | <cat>
```

```
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .

<ephemeral> ::= <ephemeral> <op> <ephemeral>
                | <ephemeralT>
<ephemeralT> ::= ‘‘100 random real constants’’
<op> ::= + | - | / | *
```

The concatenation section (<cat>) only allows the creation of constants through the concatenation of digits. This is in contrast to the Traditional section (<trad>) that restricts constant creation to the generation of values from expressions using a fixed set of constants specified by the non-terminal <tradT>. The third section concerns grammatical ephemeral random constants. In this method, a set of 100 real-valued constants are generated randomly in the range 0 to 100 inclusive at the outset of a run and these are then directly incorporated as choices for the nonterminal <ephemeralT>. In a standard GP manner, these constants can then be utilised in arithmetic expressions to generate new constant values, however, unlike in GP these ephemeral random constants can be switched on or off by simply selecting their corresponding production rule thus overcoming potential deletion from the population.

The meta-Grammar for constant generation is provided below.

```
<g> ::= "<SlnCatR> ::=" <catRs>
        "<SlnCat> ::="
        <cats> "<SlnDigit> ::=" <digit>
<catRs> ::= <catRt> "|" <catRs> | <catRt>
<catRt> ::= "<SlnCat>" | "<SlnCat>"."<SlnCat>"
<cats> ::= <catT> "|" <cats> | <catT>
<catT> ::= "<SlnDigit>"<catT> | "<SlnDigit>"
        | <digit>
<digit> ::= <digitT> "|" <digit> | <digitT>
<digitT> ::= 0|1|2|3|4|5|6|7|8|9
```

A simple example of this meta-Grammar in action is seen where we aim to evolve the target 50. In one such experiment the meta-Grammar produced the solution grammar displayed below.

```
<SlnCatR> ::= <SlnCat>
<SlnCat> ::= <SlnDigit>0
<SlnDigit> ::= 5
```

This solution grammar then makes it very easy to produce the target as the only mapping available produces 50. This underlines the strength of the diploid structure and use of a meta-Grammar as it allows the grammar itself specialise towards the solution.

3. PROBLEM DOMAIN & EXPERIMENTAL APPROACH

In this study, we compare the utility of two different constant creation grammars (one which combines 3 methods) on a series of benchmark constant creation problems. The constant generation problems tackled are; Finding a Static Constant, Finding Dynamic Real Constants, and the Logistic Equation. A description of each problem follows.

3.1 Finding a Static Constant

The aim of this problem is to evolve a single integer constant. For these experiments a simple integer value within the range of the Ephemeral random constants was selected, 50. Fitness in these experiments is the absolute difference between the target and evolved values, the goal being to minimise this value.

3.2 Finding Dynamic Real Constants

This problem involves a dynamic fitness function that changes its target real constant value every 10th generation. Two instances of this problem are tackled, the first sets the successive target values to be 192.47, 71.84, 71.83, 173.59, 192.47, i.e. generation for 0 to 9 192.47 is the target 10 to 19 71.84, etc., and the second instance oscillates between the two values 192.47 and 71.84. The aim with these problems is to analyse the different constant representations in terms of their ability to adapt to a changing environment, and to investigate that behaviour in the event of both small and large changes. As in the static constant problem, fitness in this case is the absolute difference between the target and evolved values, with the goal being the minimisation of this error.

3.3 The Logistic Equation

In systems exhibiting chaos, long-term prediction is problematic as even a small error in estimating the current state of the system leads to divergent system paths over time. Short-term prediction however, may be feasible [16]. Because chaotic systems provide a challenging environment for prediction, they have regularly been used as a test-bed for comparative studies of different predictive methodologies [17, 18, 19]. In this study we use time-series information drawn from a simple quadratic equation, the logistic difference equation.[1] This equation has the form:

$$x_{t+1} = \alpha x_t(1 - x_t) \qquad x \in (0.0, 1.0)$$

The behaviour of this equation is crucially driven by the parameter α. The system has a single, stable fixed point (at $x = (\alpha - 1)/\alpha$)for $\alpha < 3.0$ [19]. For $\alpha \in (3.0, \approx 3.57)$ there is successive period doubling, leading to chaotic behaviour for $\alpha \in (\approx 3.57, 4.0)$. Within this region, the time-series generated by the equation displays a variety of periodicities, ranging from short to long [20]. In this study, three time-series are generated for differing values of α. The choice of these values is guided by [20], where it was shown that the behaviour of the logistic difference equation is qualitatively different in three regions of the range (3.57 to 4.0). To avoid any bias which could otherwise arise, parameter values drawn from each of these ranges are used to test the constant evolution grammars. The goal in this problem is to rediscover the original α value. As this equation exhibits chaotic behaviour, small errors in the predicted values for α will exhibit increasingly greater errors, from the target behaviour of this equation, with each subsequent time step. Fitness in this case is the mean squared error, which is to be minimised. 100 initial values for x_t were used in fitness evaluation, and for each x_t iterating 100 times (i.e. x_t to x_{t+100}).

[1]This is a special case of the general quadratic equation $y = ax^2 + bx + c$ where $c = 0$ and $a = -b$.

4. RESULTS

For every problem instance, 30 runs were conducted using population sizes of 500, running for 50 generations, adopting one-point crossover at a probability of 0.9, and bit mutation at 0.1, along with roulette selection and a replacement strategy where the worst performing 25% of the population is replaced each generation with newly generated individuals from crossover and mutation. In the case of the meta-Grammar runs population sizes of 250 were adopted due to the increased computational time required to process these runs introduced by the maintenance of diploid chromosomes. As such we would consider these results to reflect an understated performance of the meta-Grammar approach.

4.1 Finding a Static Constant

The results presented in Fig. 1 display a comparison of the average best fitness of each of the grammars over the 30 runs.

As can be seen in Fig. 1 the meta-grammar begins with a poorer fitness in comparison to the Combination grammar but quickly evolves a comparable fitness over the 50 generations. A t-test and a bootstrap t-test [25] reveal that there is no significant difference in the results by the final generation. The average best performance of the Combination grammar by the final generation was a difference of 0.373499 with 6 runs evolving the exact target. In comparison the meta-grammar produced an average best performance of 0.391333 and reached the target exactly in 14 of the 30 runs.

4.2 Finding Dynamic Real Constants

In Fig. 2 graphs are presented for the experiments where the set of numbers to be evolved over the course of a run are: 192.47, 71.84, 71.83, 173.59 and 192.47. Here the combination again begins with a good fitness and once generation 10 is reached and the target changes to 71.84, it quickly attains a very good fitness. This is due to the target being within the range of the Ephemeral random constants, however at generation 30, where the target changes to 173.59, this fitness deteriorates significantly. Analysis at this point, where the target is 173.59, shows that the Ephemeral random constants method has grown to occupy 45% of the population versus 36% and 17% for the Concatenation and Traditional methods respectively. This suggests that while the target was with in the range of the Ephemeral constants it was able to quickly attain a high fitness and a strong position in the population but following from this was unable to successfully evolve from this position once the target left its range as demonstrated by the lack of significant evolution towards a better fitness for the last two targets. In these experiments the meta-grammar method again starts off with a poorer fitness but attains a fitness similar to the Combination grammar by generation 10. When the target changes to 173.59 it too experiences a strong deterioration in fitness but unlike the combination grammar it takes large leaps in fitness as the generations progress on this target. It then goes on to continue improving fitness when the target again shifts at generation 40 to 192.47.

Results for the oscillating problem instance are presented in Fig. 3. Here, where the target oscillates from 192.47 to 71.84 every 10 generations, we notice a similar trend. In the combination grammar by generation 20 Ephemeral constants have reached a strong position within the population after a period with 71.84 as the target. The fitness drops drastically when the target changes to 192.47. When the target reaches the higher number for the third time the fitness is worse again due perhaps to a further loss of diversity in the population between the different methods in the Combination grammar. For the meta-grammar method it begins with a poorer fitness but quickly catches the Combination grammar by generation 10. Once the target changes to 192.47 we see a similar story to the dynamic experiments where meta-grammar begins the target with a poor fitness but quickly evolves more fit individuals over the 10 generations. Interestingly this trend is emphasised when the target hits 192.74 for the second time where meta-grammar begins with a fitness that is worse than at the start the last time but ends on a fitness that is better than at generation 30.

An interesting aside at this point is to examine the changes in the best performing solution grammar over the course of an oscillation experiment. In one such experiment the meta-Grammar was able to produce a solution grammar which helped in closely approximating the 71.84 target. This grammar is described below. A solution yielded by this grammar produced a phenotype of 72.32.

```
<SlnCatR> ::= <SlnCat> | <SlnCat>.<SlnCat>
<SlnCat> ::= <SlnDigit>2
<SlnDigit> ::= 3 | 7
```

The system was then able to maintain this individual in the population when the target swung to 192.47 and recall it again when the target returned to 71.84, providing a level of memory. For the target of 192.47 $(GE)^2$ consistently evolved improving solution grammars right down to the final generation when one best performing grammar transitioned from

```
<SlnCatR> ::= <SlnCat>
<SlnCat> ::= <SlnDigit>0
           | <SlnDigit><SlnDigit><SlnDigit>
<SlnDigit> ::= 0 | 1 | 0
```

and a solution of 101 to

```
<SlnCatR> ::= <SlnCat>.<SlnCat>
<SlnCat> ::= <SlnDigit><SlnDigit><SlnDigit>
<SlnDigit> ::= 0 | 2 | 9 | 1
```

and a solution of 190.021 achieving a good approximation of the target. This is in contrast to the Combination grammar method which generally saw little or no evolution towards the target when it swung to the largest number outside the range of the grammatical ephmeral constants range.

4.3 The Logistic Equation

Here both methods present good fitnesses. Table 1 shows average best fitnesses for the different values of α.

Table 1: Average best fitness for different values of α for each grammar.

α	Combo	$(GE)^2$
3.59	0.000061	0.00032
3.80	0.00045	0.00041
3.84	0.00024	0.0002468

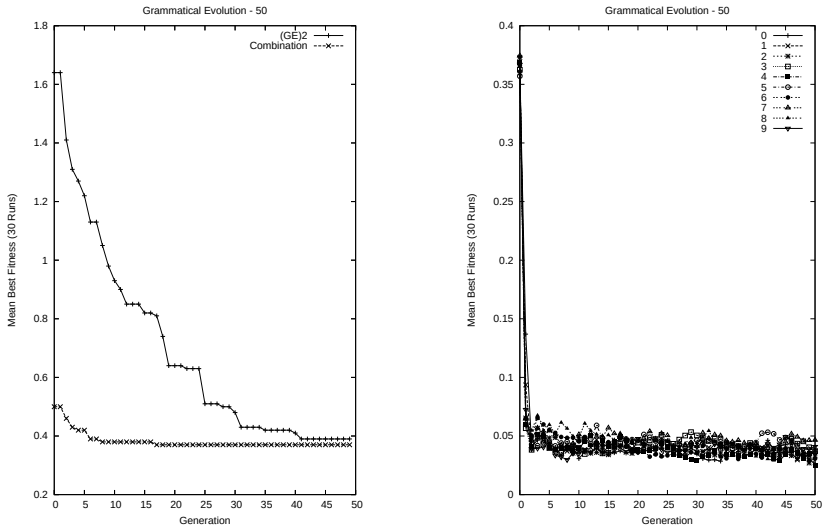

Figure 1: Plot of the mean best fitness values for each constant generation method (left) and the mean symbol usage at each generation (right) on the static constant problem instance.

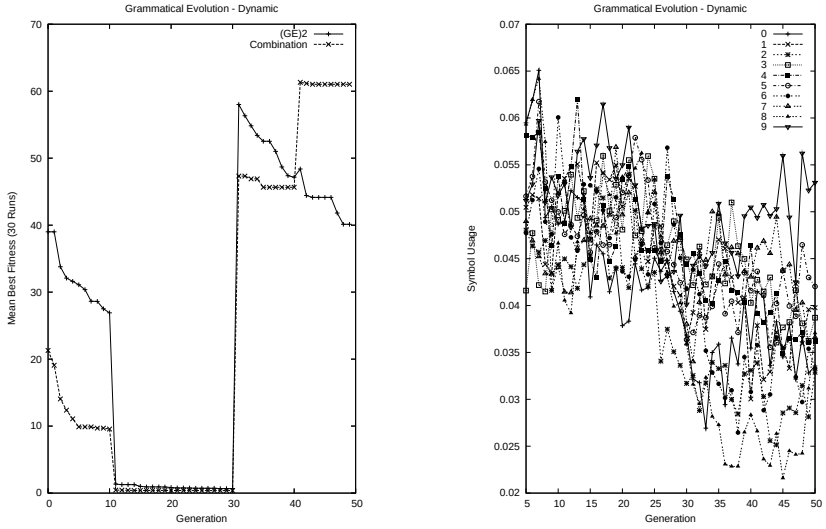

Figure 2: Plot of the mean best fitness values for each constant generation method (left) and the mean symbol usage at each generation (right) on the first dynamic problem instance.

As can be seen in the table meta-grammar performs well in comparison to the Combination grammar with close results in all but $\alpha = 3.59$. Fig 1 presents a sample of the results for $\alpha = 3.84$. It once more follows trends seen in the previous experiments where meta-grammar begins with a poorer fitness but rapidly takes the evolutionary steps to reach a fitness similar to the Combination grammar.

5. CONCLUSIONS & FUTURE WORK

This study demonstrates the utility of Grammatical Evolution by Grammatical Evolution in evolving constants. In [11] the benefits of the Combination grammar were highlighted. Here we compare the Combination grammar with the meta-grammar method and see that the meta-grammar method has advantages over the combination grammar. These advantages are largely seen in the dynamic experiments where $(GE)^2$ is able to quickly evolve a new target with large evolutionary steps. This is due to its diploid structure where both the grammar and solution are evolved simultaneously and favourable biases in a grammar are quickly built upon.

Among the static experiments the meta-grammar method is able to hold its own with t-tests highlighting that there was no significant difference between the methods for evolving 50 but at the same time evolved over twice the num-

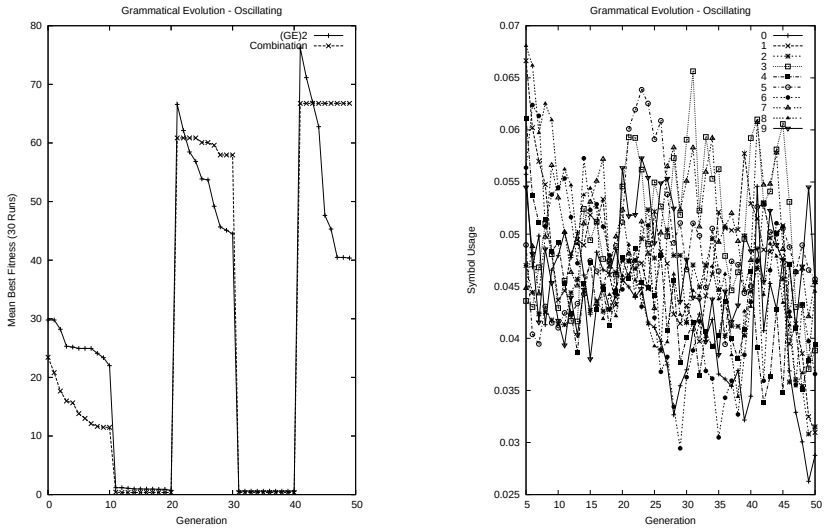

Figure 3: Plot of the mean best fitness values for each constant generation method (left) and the mean symbol usage at each generation (right) on the oscillating dynamic problem instance.

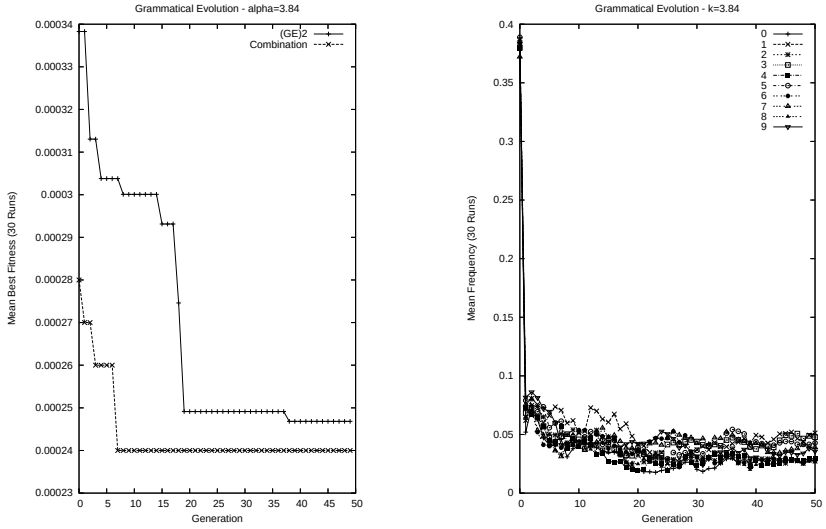

Figure 4: Plot of the mean best fitness on the logistic equation problem instance where α=3.84.

ber of exact solutions for the target. In the logistical equation meta-grammar achieved a better fitness in one out of three values for α, and only marginally losing at one setting. When considering these results it must also be borne in mind that the meta-grammar method had half the population size used by the Combination grammar and so potentially better results may be expected. Future work should explore this possibility.

One of the interesting features in this study is the high rate of evolution produced by the meta-grammar method. In all problem instances meta-grammar began the early generations with a far inferior fitness due to the larger search space presented by the diploid chromosome structure. However over a small number of generations this disadvantage is quickly over come and fitnesses are attained which are comparable to the Combination grammar and its smaller search space. This strength makes meta-grammar and $(GE)^2$ an ideal candidate in dynamic problems as in the results of Section 4.2 where the diploid structure begins each new target behind the haploid structure employed by the Combination grammar but over the course of 10 generations surpasses the fitness of the Combination method. Considering this further analysis of the meta-grammar should be conducted in dynamic environments where the ability to quickly navigate a larger search space is beneficial.

6. REFERENCES

[1] O'Neill, M., Ryan, C. (1999). Automatic Generation of Caching Algorithms, In K. Miettinen and M.M. Mäkelä and J. Toivanen (Eds.) Proceedings of EUROGEN99, Jyväskylä, Finland, pp.127-134, University of Jyväskylä.

[2] Dempsey, I., O'Neill, M. and Brabazon, T. (2002). Investigations into Market Index Trading Models Using Evolutionary Automatic Programming, In *Lecture Notes in Artificial Intelligence, 2464*,Proceedings of the 13th Irish Conference in Artificial Intelligence and Cognitive Science, pp. 165-170, edited by M. O'Neill, R. Sutcliffe, C. Ryan, M. Eaton and N. Griffith, Berlin: Springer-Verlag.

[3] O'Neill, M., Dempsey, I., Brabazon, A., Ryan, C. (2003). Analysis of a Digit Concatenation Approach to Constant Creation. In LNCS 2610 Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003, pp.173-182. Springer-Verlag.

[4] Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

[5] O'Neill, M., Ryan, C. (2003). Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers.

[6] O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution. PhD thesis, University of Limerick, 2001.

[7] O'Neill, M., Ryan, C. (2001) Grammatical Evolution, *IEEE Trans. Evolutionary Computation*, 5(4):349-358, 2001.

[8] Ryan C., Collins J.J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, 83-95, Springer-Verlag.

[9] O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.

[10] O'Neill, M., Ryan, C. 2004. *Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code*. In Proceedings of EuroGP 2004, LNCS 3003, pp.138-149, Coimbra, Portugal, Springer 2004.

[11] Dempsey, I., O'Neill, M., Brabazon, T. (2004). *Grammatical Constant Creation.* Proceedings of the Genetic and Evolutionary Computation Conference Part II, 447-458, Springer-Verlag.

[12] Koza, J.R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press.

[13] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann.

[14] Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufmann.

[15] Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers.

[16] Holland, J. (1998). *Emergence from Chaos to Order*,Oxford: Oxford University Press.

[17] Nie, J. (1997). Nonlinear time-series forecasting: A fuzzy-neural approach, *Neurocomputing*, 16:63-76.

[18] Castillo, E. and Gutierrez, J. (1998). Nonlinear time series modeling and prediction using functional networks. Extracting information masked by chaos, *Physics Letters A*, 244:71-84.

[19] Saxen, H. (1996). On the approximation of a quadratic map by a small neural network, *Neurocomputing*, 12:313-326.

[20] May, R. (1976). Simple mathematical models with very complicated dynamics, *Nature*, 261:459-467.

[21] Ryan, C., O'Neill, M. (2002). How to do anything with Grammars. *Proc. of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference 2002*, pp. 116-119.

[22] Chomsky, N. (1975). *Reflections on Language*. Pantheon Books. New York.

[23] Pinker, S. (1995). *The language instinct: the new science of language and the mind.* Penguin, 1995.

[24] Lewin, B. (2000). *Genes VII.* Oxford University Press, 2001.

[25] Efron, B., Tibshirani, R. (1993). *An Introduction to the Bootstrap.* CRC Press, ISBN 0-412-04231-2.