Grammatical Constant Creation

Ian Dempsey[1], Michael O'Neill[1], and Anthony Brabazon[2]

[1] Biocomputing and Developmental Systems Group
University of Limerick, Ireland.
Ian.Dempsey@fmr.com, Michael.ONeill@ul.ie
[2] Dept. Of Accountancy, University College Dublin, Ireland.
Anthony.Brabazon@ucd.ie

Abstract. This study examines the utility of grammatical ephemeral random constants, and conducts an analysis of the preferences of evolutionary search when a number of different grammar based constant generation methods are provided with Grammatical Evolution. Three constant generation techniques are supplied, namely, grammatical ephemeral random constants, digit concatenation, and an expression based approach. A number of constant generation problems are tackled to analyse this approach, with results indicating a preference for both the digit concatenation and grammatical ephemeral random constants. The provision of different constant generation strategies allowing the evolutionary process to automatically determine which technique to adopt would, therefore, appear to be advantageous.

1 Introduction

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 [1–3]. 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. We now extend these studies with the introduction of a third constant creation technique based on ephemeral random constants called grammatical ephemeral random constants. Within a grammar, we study the benefits of defining all three constant generation methods simultaneously, and allowing evolution to *select* the most appropriate strategy.

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 prespecified range at runs' initialisation [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 /.

There have been a number of variations on the ephemeral random constant idea in tree-based GP systems, all of which have the common aim of making small changes to the initial constant values created in an individual. *Constant perturbation* [5] allows GP to fine-tune floating point constants by multiplying every constant within an individual proto-solution by a random number between 0.9 and 1.1, having the effect of modifying a constants value by up to 10% of their original value. *Numerical terminals* and a *numerical terminal mutation* were used in [6] instead of ephemeral random constants, the difference being that the numerical terminal mutation operator selects a real valued numerical terminal in an individual and adds a noise parameter, drawn from a Gaussian distribution, such that small changes are made to the constant values.

A *numeric mutation* operator, that replaces all of the numeric constants in an individual with new ones drawn at random from a uniform distribution within a specified selection range, was introduced in [7]. The selection range for each constant is specified as the old value of that constant plus or minus a temperature factor. This method was shown to produce a statistically significant improvement in performance on a number of symbolic regression problems ranging in difficulty. More recently Ryan and Keijzer [8] have found that it is important to maintain a diverse set of constants within a population, therefore suggesting that the continual introduction of new constants into the population is critical. In fact, their results suggest that the more disruptive the perturbations are to the constants present the more of them that are adopted in successful solutions.

In addition, there have been a number of attempts to fine tune constants using statistical and mathematical methods such as Linear Scaling, Least Mean Squares and Gradient Descent, for example see [9, 10].

This contribution is organised as follows. Section 2 provides a short introduction to 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 [11–15], and can be considered a form of grammar-based genetic programming. Rather than representing the programs as parse trees, as in GP [4, 16–19], 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 [11].

3 Problem Domain & Experimental Approach

In this study, we compare the utility of three different constant creation methods for evolving constants by defining all three methods within the same grammar, and allowing evolution to select the most appropriate approach. The constant generation problems tackled are; Finding a Static Constant, Finding Dynamic Real Constants, and the Logistic Equation. A description of each problem follows.

Finding a Static Constant The aim of this problem is to evolve a single real constant, namely 50. Fitness in this case is the absolute difference between the target and evolved values, the goal being to minimise the error.

Finding Dynamic Real Constants This problem involves a dynamic fitness function that changes its target real constant value at regular intervals (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, 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.

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 [20]. 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 [21–23]. 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 $\alpha < 3.0$ [23]. 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 [24]. In this study, three time-series are generated for differing values of α. The choice of these values is guided by [24], 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}).

Constant Creation Grammar Three constant generation techniques are provided within the same grammar for this study, with the grammar adopted provided below.

```
<exp> ::= <value>
```

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

```
<value> ::= <trad> | <catR> | <ephemeral>
<op> ::= + | - | / | *
<trad> ::= <trad> <op> <trad> | <tradT>
<tradT> ::= 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0
          | 6.0 | 7.0 | 8.0 | 9.0
<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 randomly generated real constants''
```

The concatenation part of the grammar (`<cat>`) only allows the creation of constants through the concatenation of digits. This is in contrast to the Traditional part of the grammar (`<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 part of the grammar 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.

4 Results

For every problem instance, 30 runs were conducted using population sizes of 500, running for 50 generations on the dynamic constant problems, and 100 generations for the static and logistic equation instances, 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.

Finding a Static Constant The results presented in Fig. 1 indicate a strong preference by GE for the Concatenation method in this problem.
By the final generation, on average, across thirty runs GE evolved 309 individuals using the concatenation method against 126 and 13 for the Ephemeral random constants and the Traditional methods respectively. Of the best performing individuals in the final generation 75% had evolved a Concatenated individual, 14% an Ephemeral random constant individual and 10% a Traditional. It was also worth noting from this set of experiments that after generation four GE consistently mapped each individual of type Concatenation while Ephemeral and Traditional produced a fluctuating number of un-mappable individuals within the range of 20 to 30 individuals after the early generations.

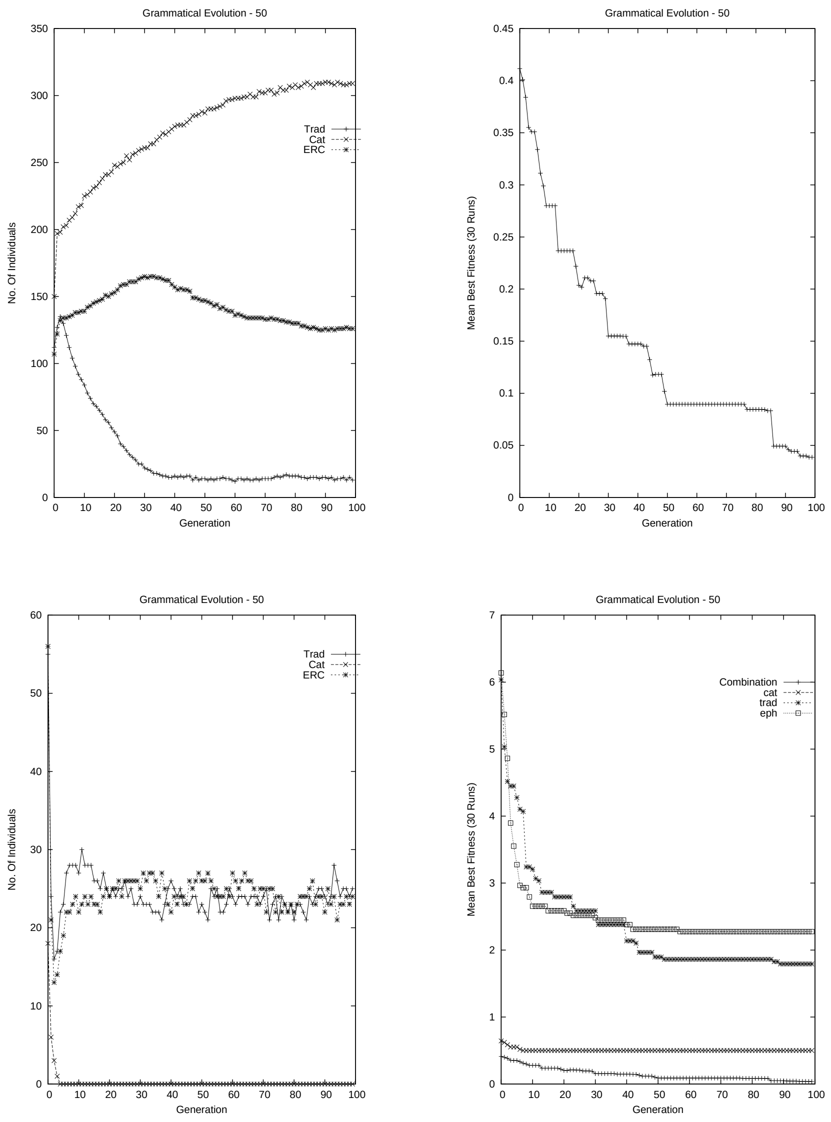

Fig. 1. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the static constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

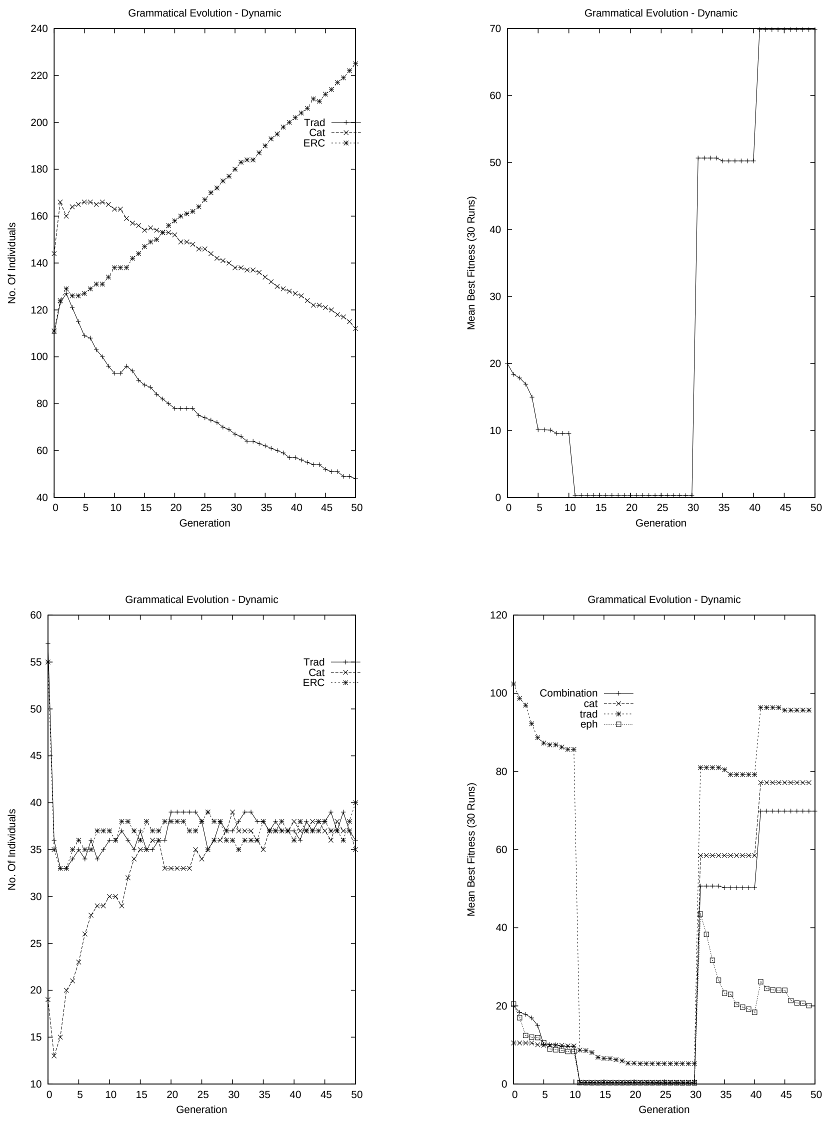

Fig. 2. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the first dynamic constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

Among the experiments incorporating each constant creation method exclusively as presented in Fig. 1 the benefits of the Concatenation method are clearly confirmed for this problem. Over the course of 30 runs Concatenation produced best performers with an average fitness of 0.50024 compared against 1.7931 and 2.27586 for the Traditional and Ephemeral methods respectively.

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. This time the Ephemeral constants gain a stronger foothold in the population over the course of the run, overtaking Concatenation before generation 20 at the same time presenting good fitness. However at generation 30, where the target changes to 173.59, this fitness deteriorates significantly. 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 was unable to successfully evolve from this position once the target left its range.

In the single method grammars however, the Ephemeral method does express a stronger ability to evolve to the targets outside its range taking large evolutionary steps towards the target after its initial change. The Concatenation and Traditional methods present performances similar to the combination grammar's performance.

Results for the oscillating non-stationary problem instance are presented in Fig. 3. In the second instance of this problem where the target oscillates from 192.47 to 71.84 every 10 generations we notice a similar trend. Again 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.

With the single grammars in the dynamic problem the results for the oscillation experiments provide similar performances with the Ephemeral method being able to take the larger evolutionary steps once the target changes.

The Logistic Equation Fig. 4 presents a sample of the results for the logistic equation with α values of 3.59, 3.8 and 3.84, which were very similar across the three α values. Here the Concatenation method gains the dominant position within the population as evolution progresses. The proportion of Ephemeral constants in the population is seen to approach the level of Concatenation constants initially, as in the dynamic experiments, however this time as evolution progresses, the Concatenation method gains the dominant position within the population. Among the best performing individuals for 3.59 60% were Ephemeral and 40% Concatenation, for 3.8 73% were Concatenation and with the remaining 27% being Ephemeral and for 3.84 80% Concatenation, 20% Ephemeral. No Traditional individuals achieved best performer in any test.

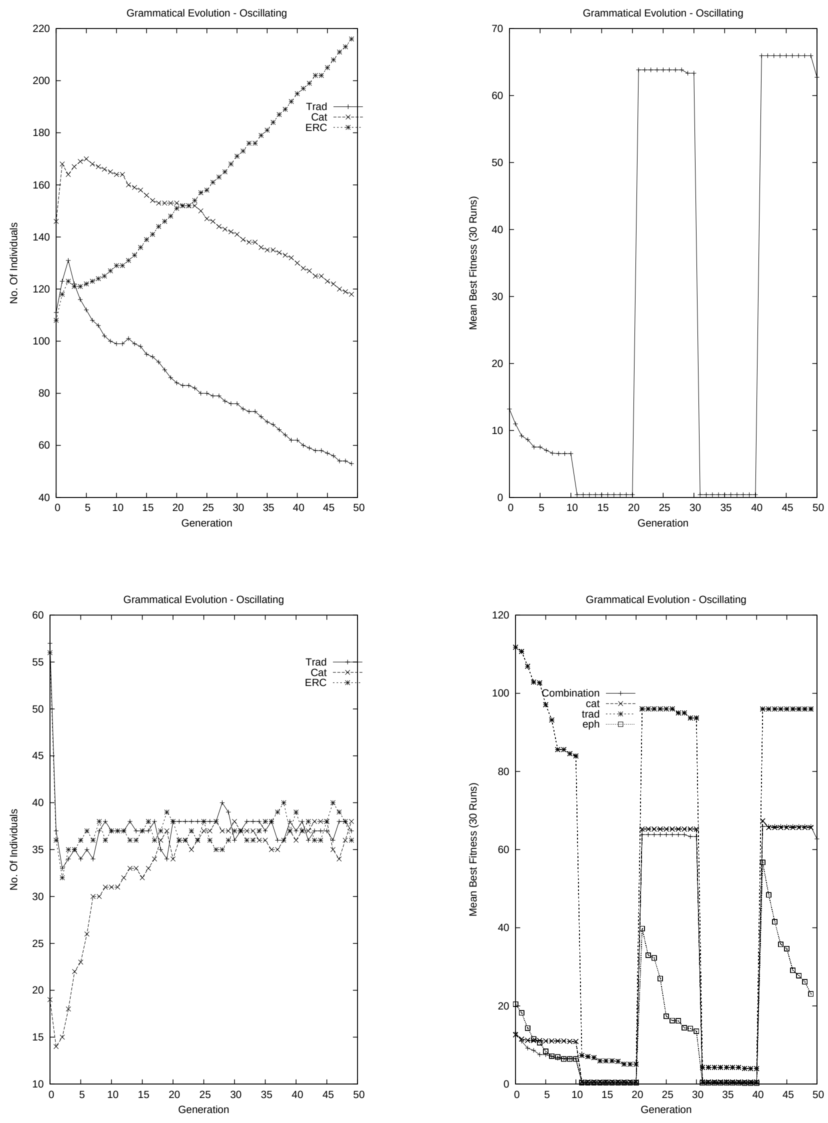

Fig. 3. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom left), on the second dynamic constant problem instance. A comparison of the combination grammar with the performance of the individual component grammars on their own is given (bottom right).

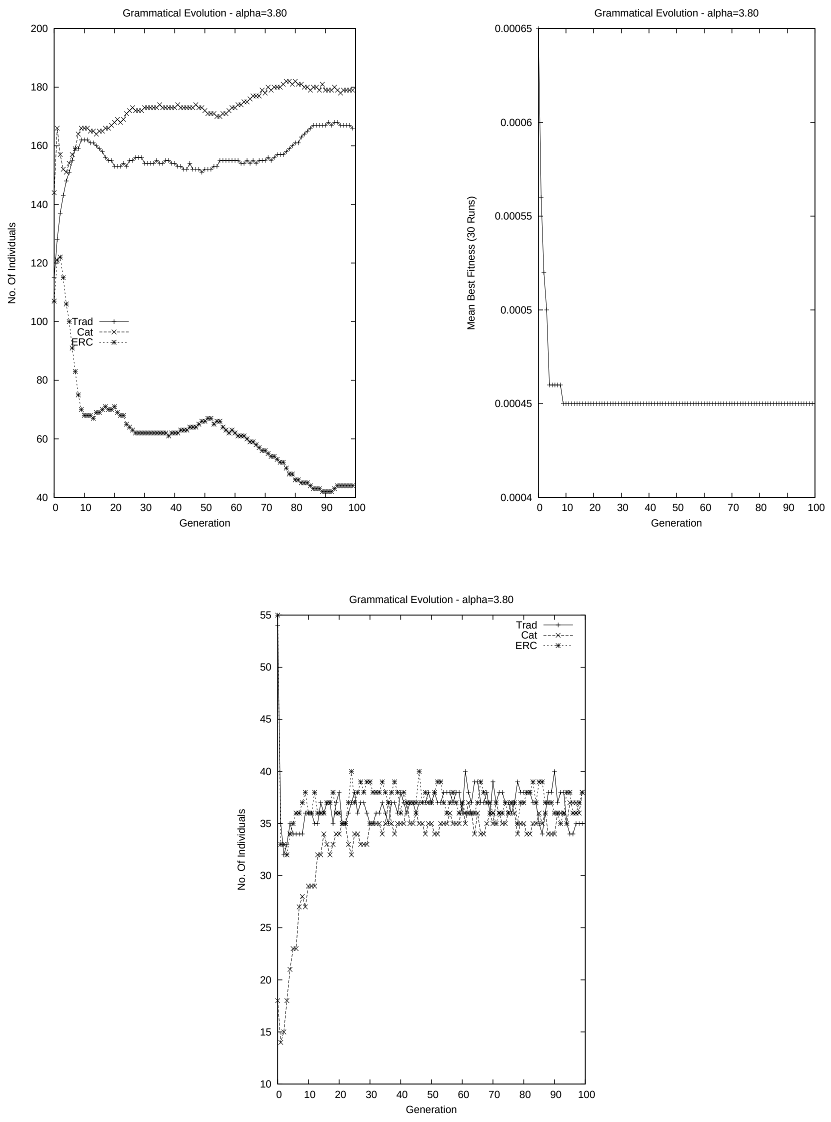

Fig. 4. Plot of the number of individuals that use each of the three constant generation methods (top left), the mean best fitness (top right), and the number of population members using each of the constant creation techniques that fail to map (bottom), on the logistic equation problem instance where α=3.80.

5 Conclusions & Future Work

Benefits of both the concatenation and newly introduced grammatical ephemeral random constants methods for constant creation are described in this paper with the traditional method performing consistently poorer than the competition for the problems analysed.

Given the dominance of the concatenation and grammatical ephemeral random constants techniques on the problem instances tackled here and the fact that there is an unclear winner between these two methods, further analysis is required on additional domains to ascertain the generality of these findings. It is not clear from this study as to why either of these approaches is superior, just that they are used in preference to the traditional method. Further investigations in this direction are therefore required.

Additionally, future work in this area will focus on the two stronger methodologies in examining their performance in evolving complex numbers outside the grammatical ephemeral random constant range, and where the concatenation method is also allowed to form expressions. A grammar that combines grammatical ephemeral random constants and concatenated constants in expressions will be examined in contrast to the grammar described here, which only allowed the exclusive use of one method for each individual. On the dynamic problem front, a more in-depth analysis of the role diversity plays within such populations is required, and into the number of generations required to improve accuracy when a target is changed.

From the results presented it is recommended that a grammar similar in form to the one in this study is adopted, which provides a facility to switch between the newly introduced grammatical ephemeral random constants and digit concatenation.

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. Spencer, G. (1994). Automatic Generation of Programs for Crawling and Walking. In Kenneth E. Kinnear, Jr. (Ed), Advances in Genetic Programming, Chapter 15, pp. 335-353, MIT Press.
6. Angeline, Peter J. (1996). Two Self-Adaptive Crossover Operators for Genetic Programming. In Peter J. Angeline and K. E. Kinnear, Jr. (Eds.), Advances in Genetic Programming 2, Chapter 5, pp.89-110, MIT Press.

7. Evett, Matthew and Fernandez, Thomas. (1998). Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming, Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, pp.66-71, Morgan Kaufmann.
8. Ryan, C., Keijzer, M. (2003). An Analysis of Diversity of Constants of Genetic Programming. In LNCS 2610 *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, pp.404-413. Springer-Verlag.
9. Keijzer, M. (2003). Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In LNCS 2610 *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, pp.70-82. Springer-Verlag.
10. Topchy, A., Punch, W.F. (2001). Faster Genetic Programming based on local gradient search of numeric leaf nodes. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2001*, pp.155-162, Morgan Kaufmann.
11. O'Neill, M., Ryan, C. (2003). Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers.
12. O'Neill, M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution. PhD thesis, University of Limerick, 2001.
13. O'Neill, M., Ryan, C. (2001) Grammatical Evolution, *IEEE Trans. Evolutionary Computation*, 5(4):349-358, 2001.
14. 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.
15. 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.
16. Koza, J.R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press.
17. 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.
18. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufmann.
19. 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.
20. Holland, J. (1998). *Emergence from Chaos to Order*,Oxford: Oxford University Press.
21. Nie, J. (1997). Nonlinear time-series forecasting: A fuzzy-neural approach, *Neurocomputing*, 16:63-76.
22. 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.
23. Saxen, H. (1996). On the approximation of a quadratic map by a small neural network, *Neurocomputing*, 12:313-326.
24. May, R. (1976). Simple mathematical models with very complicated dynamics, *Nature*, 261:459-467.