Constant Creation in Grammatical Evolution

Ian Dempsey, Michael O'Neill and Anthony Brabazon

dept CSI/NCRA

Abstract: We present an investigation into constant creation in Grammatical Evolution, a form of grammar-based Genetic Programming. The methods constant creation evaluated include digit concatenation and a grammatical version of ephemeral random constants, called persistent random constants. Experiments conducted on a diverse range of benchmark problems uncover clear advantages for a digit concatenation approach.

Keywords: Grammatical Evolution, Constant Creation, Digit Concatenation, Ephemeral Random Constants.

Reference TBD

Biographical Notes: OUR BIO

1 Introduction

Many applications in Genetic Programming Koza et al. (1999, 2003); Brabazon and O'Neill (2004); O'Neill et al. (2002) require the generation of constants, and hence the discovery of an efficient means of generating constants is important. Real world applications often operate in dynamic environments and hence, developed applications must themselves be adaptive and capable of maintaining a diverse set of constants within the population. One aspect of this adaptation is the ability to generate novel constant values as the environment changes. This study introduces and explores grammatical approaches to constant creation including persistent random constants and the novel constant-generation scheme of *Digit Concatenation* through the extension of earlier studies on concatenation O'Neill and Ryan (1999); Dempsey et al. (2002); O'Neill et al. (2003); Dempsey et al. (2004) as means of generating constants in Grammatical Evolution (GE) O'Neill and Ryan (2003); O'Neill (2001); O'Neill and Ryan (2001); O'Neill et al. (2003); Ryan et al. (1998). This method allows the creation of constants by concatenating together individual digits.

The next section outlines existing techniques used in Genetic Programming to create constants. Section 3 examines the performance of the Concatenation method in comparison with the traditional method for generating constants within GE. Section 4 compares the Concatenation method with the grammar-based Persistent Random Constant generation method. Section 5 takes this comparison further by

providing the Concatenation method with the ability to evolve expressions. Finally section 6 arrives at conclusions as to which method of constant generation is best.

2 Constant Generation in GP

Ephemeral random constants are the standard approach to constant creation in Genetic Programming (GP), having values created randomly within a pre-specified range at a run's initialisation Koza (1992). These values are fixed on creation, and beyond the initial generation new constants can only be created through combinations of these values and other items from the function and terminal set. Once a constant has disappeared from the population it is not possible to retrieve it, except by recreation through the remaining constants in the population.

A number of variations on the ephemeral random constant concept have been applied in tree-based GP systems, all of which have the common aim of making small changes to the initial constant values.

Constant perturbation Spencer (1994) allows GP to fine-tune floating point constants by rescaling them by a factor between 0.9 and 1.1. This has the effect of modifying a constant's value by up to 10% of its original value.

Numerical terminals and **numerical terminal mutation** were used in Angeline (1996). The numerical terminal mutation operator selects a real valued numerical terminal in an individual and adds a Gaussian distributed noise factor, such that small changes are made to the constant values.

The **numeric mutation** operator Evett and Fernandez (1998) replaces the numeric constants in an individual with new ones drawn at random from a uniform distribution with a pre-specified range. The selection range for each constant is specified as the old value of that constant plus or minus a temperature factor.

Linear scaling This method Iba and Nikolaev (2000); Nikolaev and Iba (2001); Keijzer (2003) has been used to optimise values within their local neighbourhood. It is performed using linear regression on the values expressed where a line is derived to fit the data and new values explored in the neighbourhood.

A study in Ryan and Keijzer (2003) used two forms of constant mutation, **creep** and **uniform** mutation, where values are altered by a small amount or mutated to a randomly different number. The study found greater benefits in uniform mutation where the ability to introduce new constants into a population as evolution progresses and maintain a highly diverse array of constants is generally beneficial to the fitness of individuals.

With Ephemeral random constants as their base, each of these methods focused on changing the original random values by small amounts to improve fitness with the exception of Ryan and Keijzer (2003), which also examined wholesale transformation of constant values finding this feature to be more beneficial than smaller changes.

GE can borrow from the experience of GP by extending the established methodology and in the spirit of GE introduce new grammatical forms of constant creation which potentially address the issue of beginning an evolutionary run with a fixed range of constants and providing the feature of creating new values over the course of a run. With this in mind we first determine the utility of this novel approach

by examining it under GE's capacity to create and adapt constants in isolation to gather a clear view of its behaviour and relative performance.

3 Evolving Constants using Digit Concatenation

The objective of this section is to determine whether Digit concatenation can outperform the traditional expression-based approach to constant creation in GE.

3.1 Traditional Constant Creation in GE

The traditional approach to constant generation in GE relies upon the defining of a handful of constants in the BNF grammar, with the recombination of these terminals using expressions and function terminals leading to the creation of "new" values. Below is an example of a grammar which adopts such an approach.

```
<value> ::= <value> <op> <value>
            | ( <value> )
            | <number>
            | <func> ( <number> )
<func> ::= sin | cos | tan
<op> ::= + | - | / | *
<number> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Here the grammar is provided with six constants, operators and trigonometric functions as terminals. The grammar may then combine these terminals to form expressions using the first production rule with a sample output looking like the following:

```
4 + 7 * (sin ( 8 + 7 ) )
```

3.2 Concatenation Constant Creation in GE

The Concatenation method for constant creation provides GE with the fundamental building blocks for the construction of numerical values. An example of a grammar using Concatenation is given below.

```
<int> ::= <int><digit> | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Which could produce a sample output of

```
2003451
```

In this grammar the digits zero through nine are used as the fundamental building blocks to create all other numbers simply by concatenating them together.

3.3 Problem Domain and Experimental Approach

Two grammars were constructed, one using Digit Concatenation and the other incorporating the Traditional method. With the aim of this study being the examination of the performance of GE in the creation and adaptation of constants in isolation the performance of these grammars is measured on three different types of constant creation problems. The constant creation problems included; finding a static real constant, finding dynamic constants and finding a coefficient of the logistic difference equation. A description of each problem follows.

3.3.1 Finding a Static Real Constant

The aim of this problem is to evolve a single real constant. Three target constants of increasing difficulty were selected arbitrarily, 5.67, 24.35 and 20021.11501. Fitness is defined as the absolute difference between the target and the evolved values, the goal being to minimise the difference.

3.3.2 Finding Dynamic Real Constants

This test involves a dynamic fitness function that changes its target real constant value at regular intervals (every 10th generation). Two case are tackled. The first sets the successive target values to be 24.35, 5.67, 5.68, 28.68 and 24.35, and the second case oscillates the target value between 24.35 and 5.67. The aim of these problems is to compare the different constant generation methods 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 finding a static real constant problem, fitness in this case is the absolute difference between the target and evolved values, with the goal being the minimisation of this difference value.

3.3.3 The Logistic Difference 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 Holland (1998). 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 Nie (1997); Castillo (1998); Saxen (1996). In this study we use time-series information drawn from a simple quadratic equation, the logistic difference equation.[a] 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 $\alpha < 3.0$ Saxen (1996). 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 May (1976).

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

In this study, three time-series are generated for differing values of α. The choice of these values is guided by May (1976), 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}).

3.3.4 Constant Creation Grammars

The grammars adopted are given below. The Concatenation grammar (Cat) only allows the creation of constants through the concatenation of digits, whereas the Traditional grammar (Trad) restricts constant creation to the generation of values from expressions.

Concatenation (Cat) Grammar

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

Traditional (Trad) Grammar

```
<value> ::= <value> <op> <value>
            | ( <value> <op> <value>)
            | <number>
<op> ::= + | - | / | *
<number> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

3.4 Results

For each grammar on every problem instance 30 runs were conducted using population sizes of 500, running for 50 generations on the static and dynamic constant problems, and 100 generations for the logistic difference equation, adopting one point crossover at a probability of 0.9 and bit mutation at 0.1, along with roulette selection and a generational rank replacement strategy of 25% where the weakest performers were replaced by the newly generated offspring.

3.4.1 Finding a Static Real Constant

On all three instances of this problem, a t-test and bootstrap t-test (5% level) on the best fitness values reveal that the digit concatenation grammar (Cat) significantly outperforms the standard expression based approach (Trad). Statistics of performance for each grammar are given in Table 1, and a plot of the mean best fitness at each generation for the three targets can be seen in Fig. 1.

Table 1 Statistics for the best fitness values (lower value is better at generation 50 on the Static Real Constant Problem)

Target Constant	Grammar	Mean	Median	Std. Dev.
5.67	Trad	0.33	0.33	0.0
	Cat	0.0	0.0	0.0
24.35	Trad	0.36	0.35	0.055
	Cat	0.002	0.0	0.009
20021.11501	Trad	7741.35	10000	3828.9
	Cat	1005.24	0.91	3049.5

Notably, the Trad grammar did not perform as well as the Cat grammar in evolving the large number by a large margin. This demonstrates that a grammar that has a concatenation approach to constant creation is significantly better at generating larger numbers. It is worth noting that larger numbers could just as easily be large whole numbers or real numbers with a high degree of precision.

3.4.2 Finding Dynamic Real Constants

For the first instance of this problem where the successive target constant values are 24.35, 5.67, 28.68 and 24.35 over the course of 50 generations, performance statistics are given in Table 2, and a plot of mean best fitness values for each grammar can be seen in Fig. 2.

Performing a t-test and a bootstrap t-test on the best fitness values at generations 10, 20, 30, 40 and 50, it is shown that there is a significant (5% level) performance advantage in favour of the Concatenation grammar (Cat) up to generation 30, beyond this point the advantages of one grammar over the other are not as clear cut.

In the second instance of this problem, where the target constant values oscillates, every 10 generations, between 24.35 and 5.67 over the 50 generations, again we see a similar trend. In this case, the concatenation grammar (Cat) is significantly better at the 5% level than the Traditional (Trad) grammar at each of the 10, 20, 30, 40 and 50 generations, however, this difference is decreasing over time. A plot of the mean best fitness can be seen in Fig. 2, and statistics are presented in Table 3. From the results on both of these dynamic problem instances, there are clearly adaptive advantages to using the concatenation grammar over the traditional expression based approach.

3.4.3 The Logistic Difference Equation

The results for all three instances of this problem can be seen in Table 4 and Fig. 3. Statistical analysis using a t-test and bootstrap t-test (5% level) reveal that the concatenation grammars (Cat & Cat+Trad) significantly outperform the traditional constant creation approach on each problem instance successfully rediscovering the target α in each case.

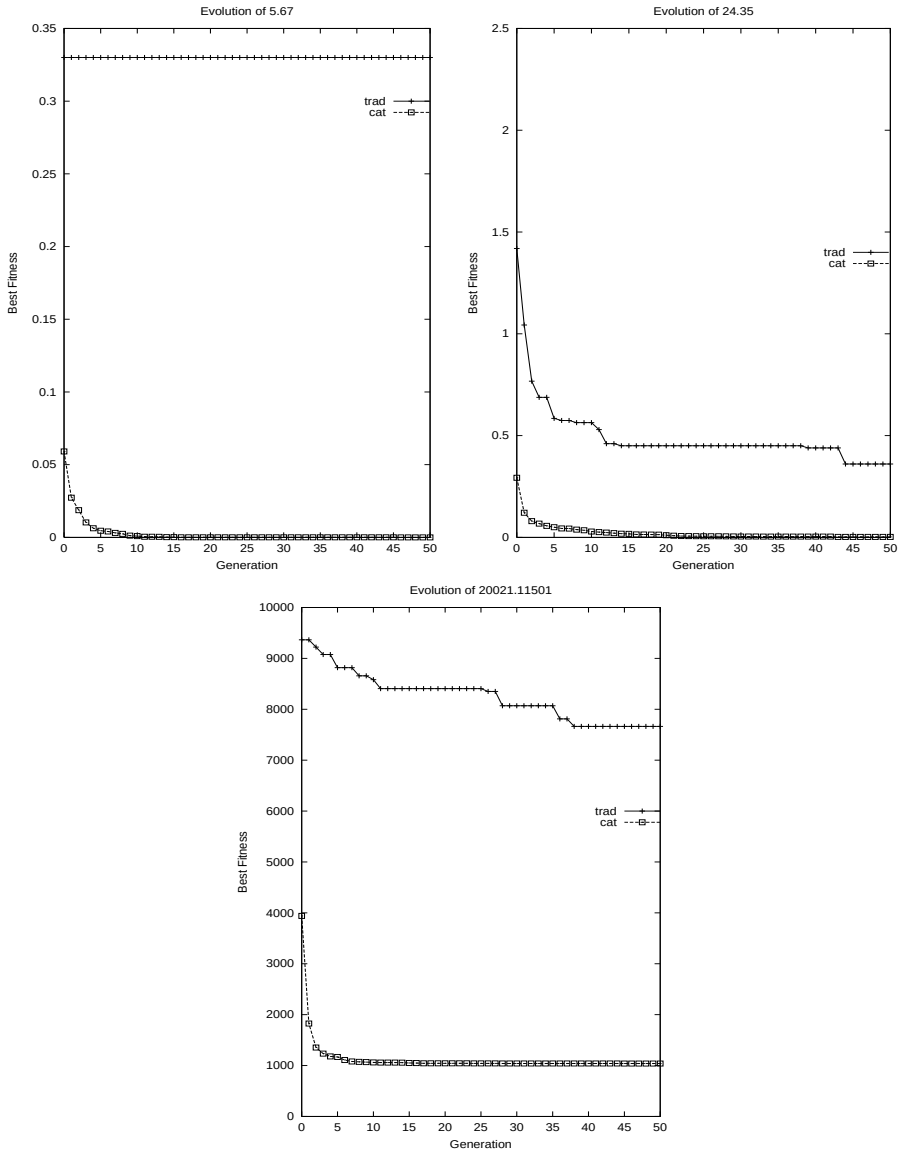

Figure 1 Mean best fitness values (lower values are better) plotted against generations for each of the four grammars. Target values are 5.67 (left), 24.35 (center), and 20021.11501 (right).

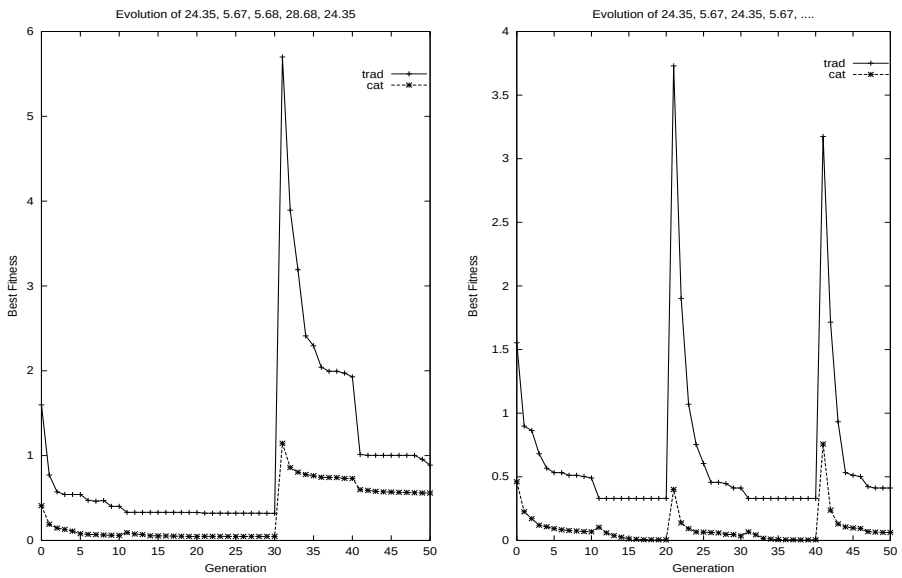

Figure 2 Mean best fitness values (lower values are better) plotted against generations for each of the three grammars. Target values are 24.35, 5.67,5.68,24.35 (left) and 24.35, 5.67,... (right).

Table 2 Statistics for the best fitness values (lower value is better) the Dynamic Real Constant Problem.

Generation	Target Constant	Grammar	Mean	Median	Std. Dev.
10	**24.35**	Trad	0.4	0.35	0.114
		Cat	0.061	0.01	0.133
20	**5.67**	Trad	0.33	0.33	0.0
		Cat	0.047	0.0	0.17
30	**5.68**	Trad	0.32	0.32	1.129e-16
		Cat	0.046	0.0	1.724e-01
40	**28.68**	Trad	2.063	1.5	3.474
		Cat	0.046	0.0	1.724e-01
50	**24.35**	Trad	0.937	0.35	2.755
		Cat	0.541	0.002	2.799

Table 3 Statistics for the best fitness values (lower value is better) the Oscillating Dynamic Real Constant Problem.

Generation	Target Constant	Grammar	Mean	Median	Std. Dev.
10	**24.35**	Trad	0.507	0.35	0.426
		Cat	0.089	0.011	0.193
20	**5.67**	Trad	0.33	0.33	0.0
		Cat	0.005	0.0	0.0167
30	**24.35**	Trad	0.487	0.35	0.426
		Cat	0.046	0.022	0.07
40	**5.67**	Trad	0.33	0.33	0.0
		Cat	0.0004	0.0	0.01
50	**24.35**	Trad	0.487	0.35	0.426
		Cat	0.061	0.014	0.131

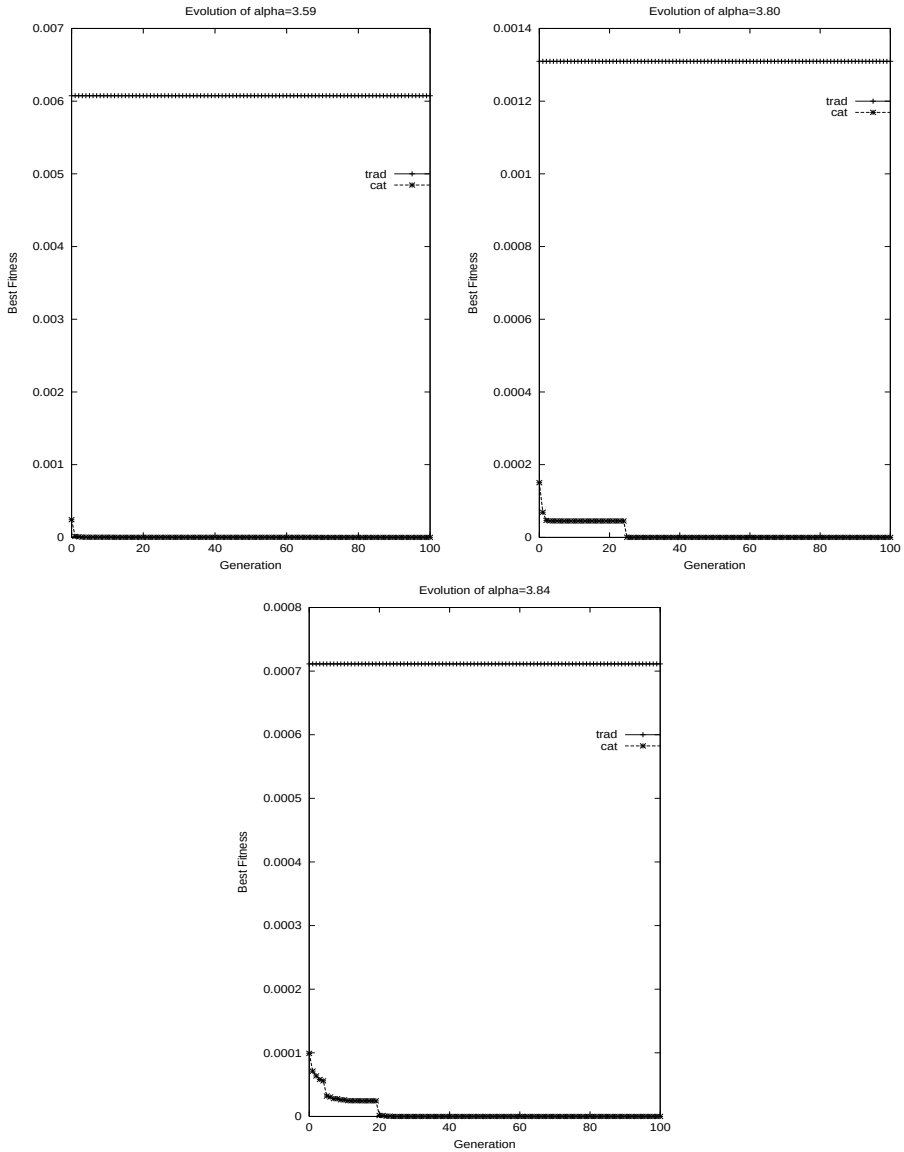

Figure 3 Mean best fitness values (lower values are better) plotted against generations for each of the three grammars. Target values of α are 3.59 (left), 3.80 (center), and 3.84 (right).

Table 4 Statistics for the best fitness values (lower value is better) the Logistic Equation Problem.

Target Constant	Grammar	Mean	Median	Std. Dev.
3.59	Trad	6.074e-03	6.074e-03	2.647e-18
	Cat	4.818e-07	3.902e-19	1.249e-06
3.80	Trad	1.310e-03	1.310e-03	6.616e-19
	Cat	4.724e-19	4.724e-19	0.0
3.84	Trad	7.113e-04	7.113e-04	2.206e-19
	Cat	6.065e-19	6.065e-19	9.794e-35

3.4.4 Discussion

An interesting feature to note in the logistic experiment results is the presence of flat line averages for the traditional grammar in each of the problem instances. What this demonstrates is a difficulty on the part of the traditional approach in evolving *real* numbers within its range. When the grammar produced a reasonably good fitness at the first generation it settled upon this local minima. New expressions brought about through crossover and mutation failed to alter in small amounts the fitness of the best individual by bringing its value closer to that of the target through different expressions. Take for example the average fitness attained for $\alpha = 3.59$ which was about 0.006, to reduce an expression which already yielded about 3.596 by 0.006 using integers zero through nine and the operators provided would be a relatively complex task. To confirm this problem the presence of a flat line average in evolving 5.67 for the Traditional grammar is also demonstrated where the same conditions again exist: the target is a real number within the terminal set of digits provided to the grammar. Considering this we can say that while it is good at attaining a reasonable fitness for *real* targets within its range, the forming of expressions using integers to get *real* values proves too complex for the Traditional approach. These results would also suggest that the provision of a larger set of real values in the grammar might enhance the performance of the Trad approach in its own right. To this end we investigate a version of ephemeral random constants for GE in the following section.

4 Analysis of Digit Concatenation & Persistent Random Constants

In section 3 the Concatenation method displayed superior performance over the traditional technique for evolving constants in GE. In this section the Concatenation method is analysed further by examining the preferences of evolutionary search when a number of different grammar-based constant generation methods are provided to GE. Along with Concatenation, a grammar defined Persistent Random Constants technique is explored as well as the Traditional technique described in the previous section. All three methods are included in a grammar which only allows the use of one method exclusively. The preference of the evolutionary search is then examined across a range of constant generation problems.

4.1 Persistent Random Constant Creation in GE

In section 2 a description of Ephemeral Random Constants in GP was given. Here we introduce a grammatical form of this known as Persistent Random Constants. Like the GP approach to Ephemeral random constants, the grammatical approach also generates a number of real values within a pre-specified range. Where it differs is that these numbers are then added to the grammar to be used by GE. This has the added effect that the random numbers become available to the evolutionary process throughout the lifetime of the experiment as they are part of the grammar itself. In GP Ephemeral random constants these numbers, once evolved out of the population, cannot be re-introduced into the population which can lead to a potential loss in the diversity of numbers available. Indeed this approach does have similarities to the Traditional method for GE, where instead of a small fixed set of constants, Persistent Random Constants uses a larger number of randomly generated constants which are persistent for the lifetime of the run (as they are stored within the grammar) with the implication of better coverage of the constant search space.

4.2 Experimental Approach

A comparison is performed on the utility of three different constant creation methods for evolving constants by performance analysis on three different types of constant creation problems. The problems tackled are, finding a static integer, finding dynamic real constants, and the finding a paramater to the logistic difference equation. The problems used in this section though similar to those in section 3 use different targets. The reason behind this is to get a spread of targets both inside and outside the range of the Persistent Random Constants range as most of the targets in section 3 resided within this range. This is done to examine further the issue discussed in section 3.4.4, where the Traditional approach was observed settle on local minima and also to examine the Persistent Random Constants ability to evolve targets outside its range. Comparative benchmarks can be drawn from the previous section by examining the performance of the grammars which use each method exclusively.

4.2.1 Finding a Static Constant

The aim of this problem is to evolve a single integer constant. For these experiments two constants were selected; a simple integer value within the range of the Ephemeral random constants, 50, and a complex floating point real number outside the range of the Ephemeral random constants, 20021.11501. Fitness in these experiments is the absolute difference between the target and evolved values, the goal being to minimise the difference value.

4.2.2 Finding Dynamic Real Constants

This instance of finding dynamic real constants involves a dynamic fitness function that changes its target real constant values 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, 173.59 and 192.47, the second instance oscillates between the two values 192.47 and 71.84. The aim here as in the previous section 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 the evolved values, with the goal being the minimisation of this difference value.

4.2.3 The Logistic Difference Equation

This problem is used in the same manner with the same parameters as in section 3.

4.2.4 Constant Creation Grammar

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

```
<exp> ::= <value>
<value> ::= <trad> | <catR> | <presistent>
<op> ::= + | - | / | *
<trad> ::= <trad> <op> <trad> | <tradT>
<tradT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<catR> ::= <cat> <dot> <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .
<persistent> ::= <persistent> <op> <persistent> | <persistentT>
<persistentT> ::= ‘‘150 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 persistent random constants. In this method, a set of 150 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 `<persistentT>`. In a standard GP manner, these constants can then be utilised in arithmetic expressions to generate new constant values. The `<value>` production then is essentially the rule which permits the exclusive choice of one of these methods for each individual.

4.3 Results

For every problem instance the parameters used, and the number of runs conducted were the same as in section 3.4.

4.3.1 Finding a Static Constant

The results presented in Fig. 4, indicate a preference by GE for the Persistent Random Constant technique in this problem. By the final generation, on average, across thirty runs GE evolved 221 individuals using the Persistent method against 117 and 59 for the Concatenation and the Traditional methods respectively. Of the best performing individuals in the final generation 60% had evolved a Concatenation individual and 40% a Persistent Random Constant individual.

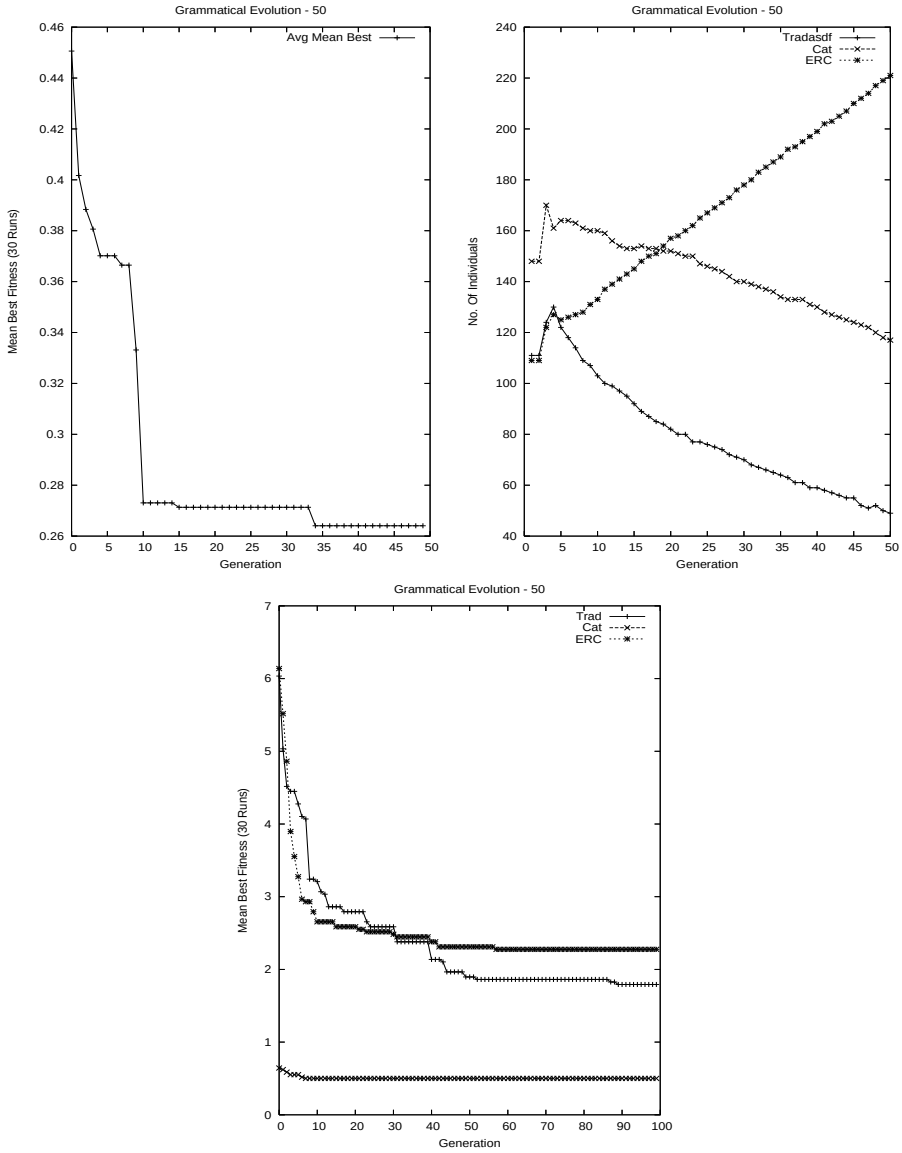

Figure 4 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each of the three constant generation methods (center) and a comparison of the performance of the exclusive component grammars (right)

Among the experiments incorporating each constant creation method exclusively as presented in Fig. 4 the benefits of the Concatenation method are highlighted 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 Persistent Random Constant methods respectively.

Fig. 5 presents the results for evolving 20021.11501. Here Persistent Random Constants are again seen to grow to dominate the population 226 members against 136 and 23 for the Concatenation and Traditional methods. However in this instance Concatenation is the method used for 100% of the best individuals yielding an average best performance of 547.217.

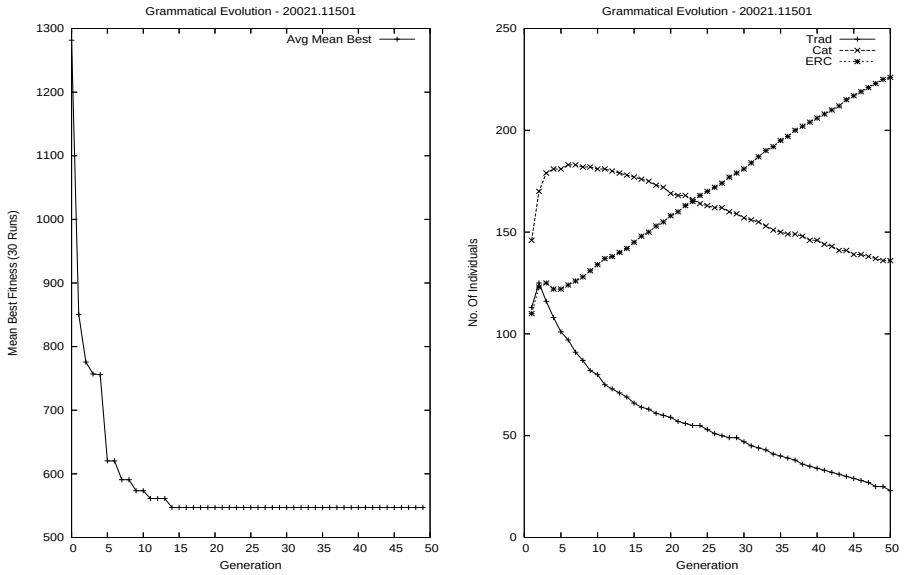

Figure 5 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each of the three constant generation methods (right).

In the experiments with exclusive grammars the concatenation method was seen to provide the best average fitness at 3140.84 with the Persistent Random Constant method providing an average best fitness of 10070.5.

4.3.2 Finding Dynamic Real Constants.

In Fig. 6, 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 Persistent Random 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 within the range of the Persistent Random 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 Persistent Random Constant 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 7. 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 Persistent Random 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.

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

4.3.3 The Logistic Equation.

Fig. 8, 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 each of the values. Here the Concatenation method gains the dominant position within the population as evolution progresses. The proportion of Persistent Random 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% used Persistent Random Constants and 40% Concatenation; for 3.8, 73% used Concatenation, 20% Persistent Random Constants. No Traditional individuals were the best performer in any test.

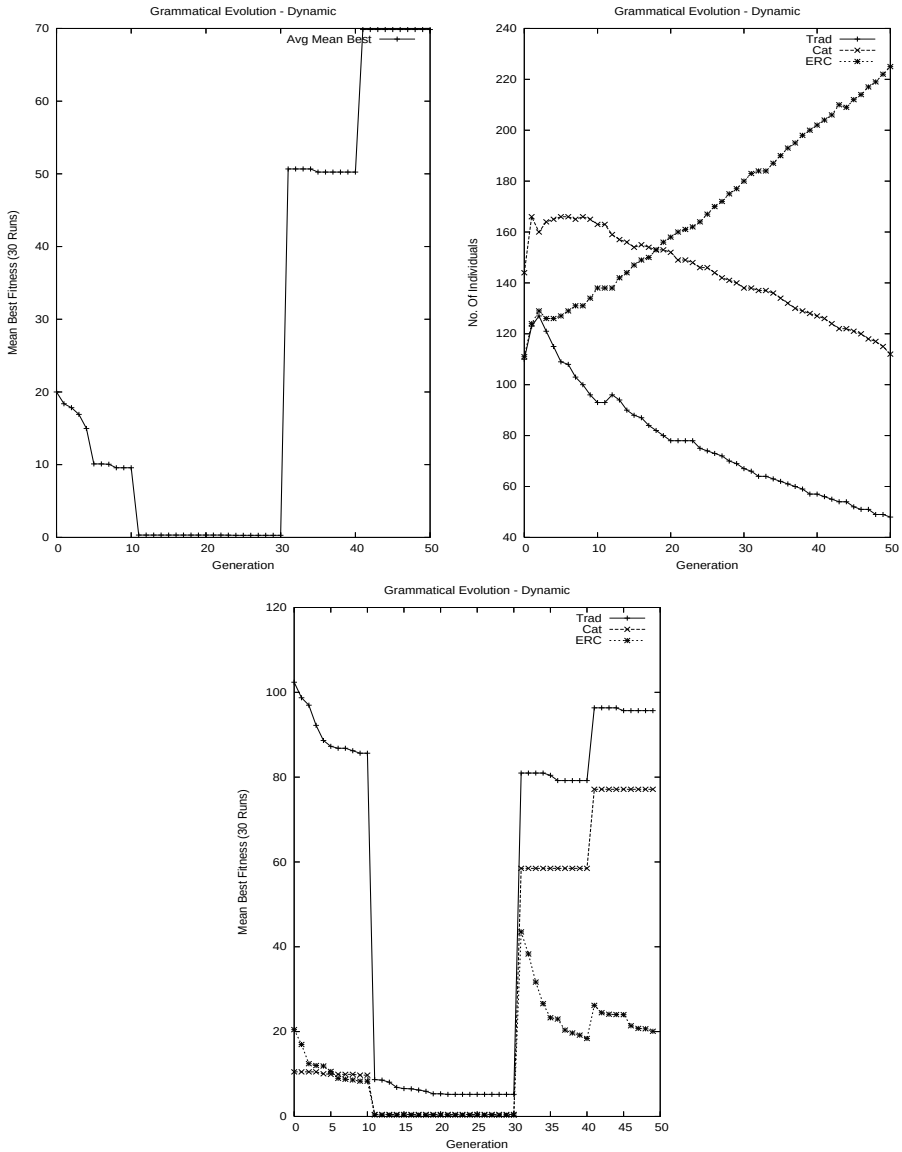

Figure 6 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each of the three constant generation methods (center) and a comparison of the performance of the exclusive component grammars (right)

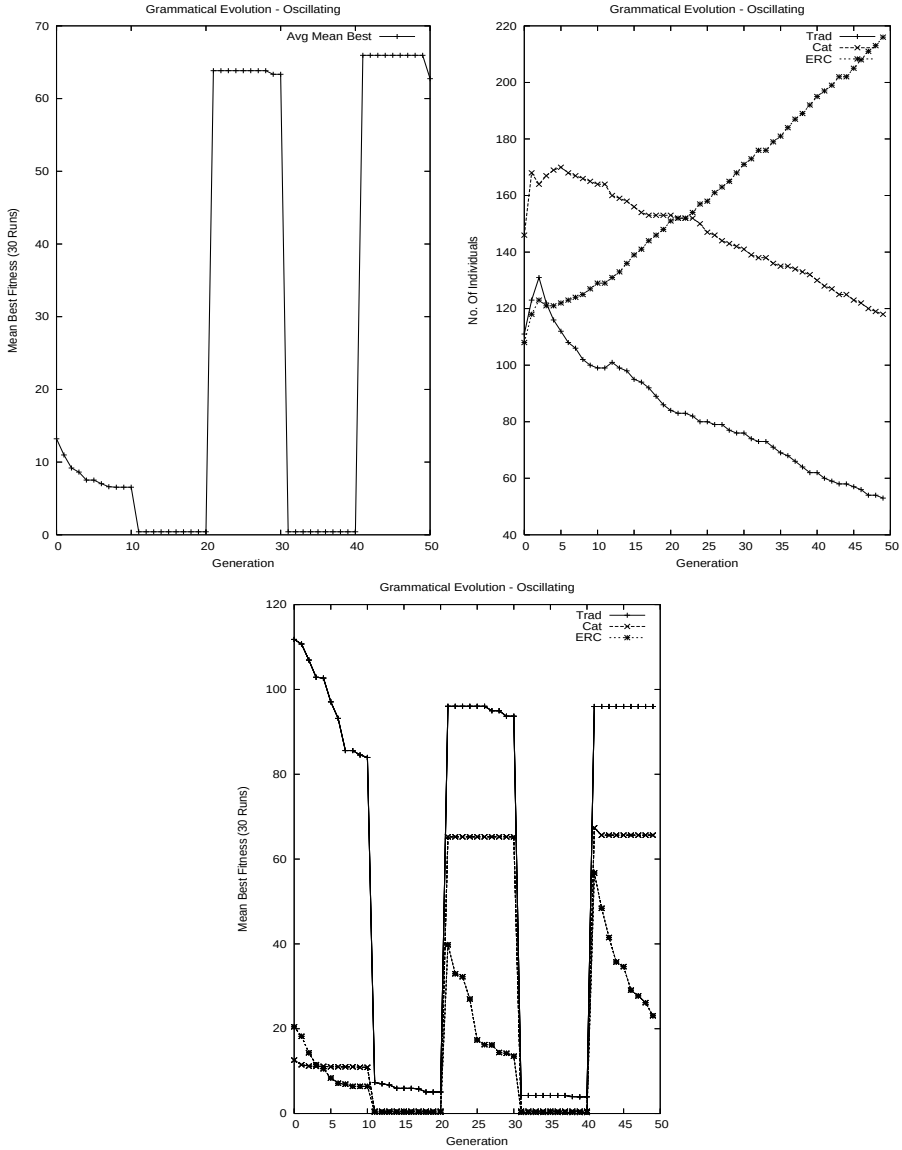

Figure 7 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each of the three constant generation methods (center) and a comparison of the performance of the exclusive component grammars (right).

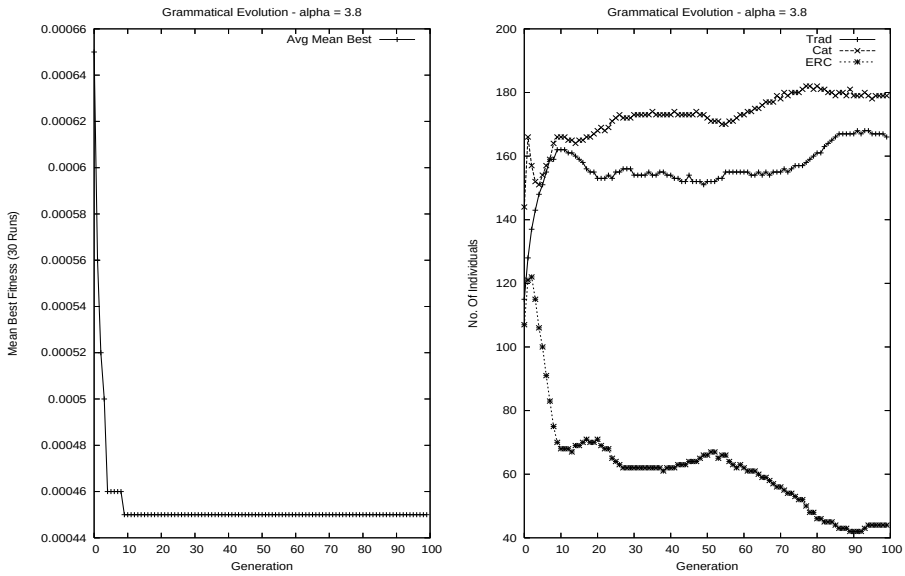

Figure 8 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each of the three constant generation methods (right).

4.3.4 Discussion

A common feature seen in almost all the results across the problems examined in this section, with the exception of the logistic equation problem, shows the Persistent Random Constants method consistently gaining larger portions within the population while not seeing this population dominance reflected in the number of best performing solutions. For the problems of evolving the static integer 50, and the dynamic targets, numbers are presented which are within the range of the Persistent Random Constants. Considering this the reason why the Persistent Random Constants method gains such a large portion is likely to be that this method would contain a relatively large number of terminals within the neighbourhood of the target, reflecting a similar advantage to the Traditional method in section 3.4. Thus presenting the evolutionary process with again a relatively large number of simple solutions, i.e. terminals, with good fitnesses in proportion to the other two methods in the grammar. In the dynamic problems once the target moves outside its range the Persistent Random Constants method had already attained a strong postion in the population over the other two methods gaining a certain reproductive momentum to the detriment of the Concatenation and Traditional methods. The Persistent Random Constant method would also be able to attain reasonable fitnesses simply by summing up two of its number terminals at the higher end of its range, requiring the exploitation of just three terminals.

This reason can then be related directly to the good performance seen by the Traditional method in attaining a larger proportion of the population in the logistics equation problems and entirely contrasting its performance in the other problems. In the logistic equation problem the target was within the range of the Traditional

grammar and also within the range of the Persistent Random Constants. The difference however is that while it was within both their ranges, the Traditional method presented the evolutionary process with a much smaller selection of terminals (the ten digits), all of which would have had reasonable fitness on their own, compared to the Persistent Random Constant method which had 150 terminals many of which could have been potentially as high as 100. What held the Traditional method back in this case is its difficulty in evolving fitter solutions that require small changes in the resultant value of a best performers expression, a problem already discussed in section 3.4.3. The Concatenation method on the otherhand proved to be better at evolving fitter solutions in section 3.4.3 for this problem explaining how it could hold onto a majority position within the population on average for this problem.

In examining the static problem with target 20021.11501, a different situation is presented. In this problem the target is well outside the range of the Persistent Random Constants yet it still grew to dominate the population. The behaviour of the Concatenation trend line in this instance however differs to the other problems. Here Concatenation ends up with a proportion of the population which is just 12 less than what it began with on average. Considering that it provided 100% of the best performing solutions, selective pressure allowed it to maintain its population share. The Persistent Random Constant method then, it seems, grew its population share to the detriment of the Traditional method which struggled to cope with the ease at which the Persistent Random Constant method could create much larger values with expressions involving its larger terminal values.

In summary, the presence of greater diversity of constants as presented by Persistent Random Constants approach and the ability to generate smaller numbers more rapidly with the Concatenation approach suggests that these are both desirable constant generation mechanisms to adopt within our GE grammars.

5 Further Study in Digit Concatenation & Persistent Random Constants

Section 4 demonstrates the superiority of both the Concatenation and Persistent Random Constant methods over the Traditional approach. In order to gain a more accurate understanding of the relative advantages of these two methods, and the merits of a combination of these approaches, a further series of experiments was undertaken. This section compares the two methods using a grammars similar to the previous section along with grammars which use each approach exclusively. However in these experiments the Concatenation method is additionally provided with the ability to form expressions.

Below are the combination grammars derived from experiments in the previous section. The first is most similar to the previous section except it only uses the Concatenation method and Persistent Random Constants. The second grammar presents GE with a method to ascertain whether the two approaches may mutually complement each other as this (cooperative) grammar allows the formation of expressions using constants derived from both paradigms.

Competitive Grammar

```
<exp> ::= <number>
<number> ::= <catE> | <persistent>
<op> ::= + | - | / | *
<catE> ::= <catE> <op> <catE> | <catR>
<catR> ::= <cat> <dot> <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .
<persistent> ::= <persistent> <op> <persistent> | <persistentT>
<persistentT> ::= ‘‘150 randomly generated real constants’’
```

Cooperative Grammar

```
<exp> ::= <number>
<number> ::= <value> <op> <value> | <value>
<value> ::= <catE> | <persistent>
<op> ::= + | - | / | *
<catE> ::= <catE> <op> <catE> | <catR>
<catR> ::= <cat> <dot> <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .
<persistent> ::= <persistent> <op> <persistent> | <persistentT>
<persistentT> ::= ‘‘150 randomly generated real constants’’
```

The following grammars incorporate each method for constant creation exclusively and are designed for symbolic regression problem to evolve the area of a circle.

Exclusive Cat

```
<exp> ::= <catE>
<op> ::= + | - | / | *
<catE> ::= <catE> <op> <catE> | (<catE><op><catE>) | <catR>
<catR> ::= <cat> <dot> <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<dot> ::= .
```

Exclusive Persistent

```
<exp> ::= <prc>
<op> ::= + | - | / | *
<prc> ::= <prc> <op> <prc> | (<prc><op><prc>) | <prcT>
<prcT> ::= ‘‘150 randomly generated real constants’’
```

5.1 Experimental Approach

The experiments performed focus on three areas; the creation of a large complex number outside the range of the Persistent Random Constants, on the flexibility of the methods in a dynamic environment and on evolving the form and constant, Pi, in the equation for calculating the area of a circle. This final experiment differs from previous experiments as having examined GEs ability to create and adapt constants we now also examine its ability to evolve an equations form.

5.1.1 Finding a Static Constant.

The target of 20021.11501 was again chosen for these experiments to enable direct comparisons with previous sections and also because of its difficulty, as it represents a high precision floating point number outside the range of the Persistent Random constants.

5.1.2 Finding Dynamic Real Constants

Here again we consider the same series of dynamic real constants as used in section 4 , successive targets of 192.47, 71.84, 173.59 and 192.47 changing at the tenth generation for fifty generations. Once more the ability to compare results with the previous section is provided.

5.1.3 Finding the Equation for the Area of a Circle

For this section a final new experiment is introduced. In this case two grammars are used which incorporate each method exclusively with the aim of evolving the equation of a circle, πr^2. The setups are tested against 100 radii each generation with the range 2-102 where the objective is to mimimise the cumulative difference to the correct area across the 100 radii.

5.2 Results

For every problem instance the parameters used and number of runs conducted were the same as in section 3.4.

5.2.1 Finding a Static Constant.

In this case Fig. 9, demonstrates that the Concatenation method began to gain an upper hand on average within the populations at generation 13. Finishing at the final generation with the lions share of the population at 318 versus 94 for the Persistent Random Constants method. Of the best performers only 1 of the 30 runs provided a solution using the Persistent Random Constants method with the best Concatenation solution producing an expression which came to within 18.3872 of the solution, this solution is provided below.

```
20002 + 0.727829
```

By the final generation the best performer on average produced a fitness of 607.968. Among the experiments with the complementary grammar the average best fitness by the final generation was a comparable 688.798, with a t-test and bootstrap t-test demonstrating no statistical difference between the results.

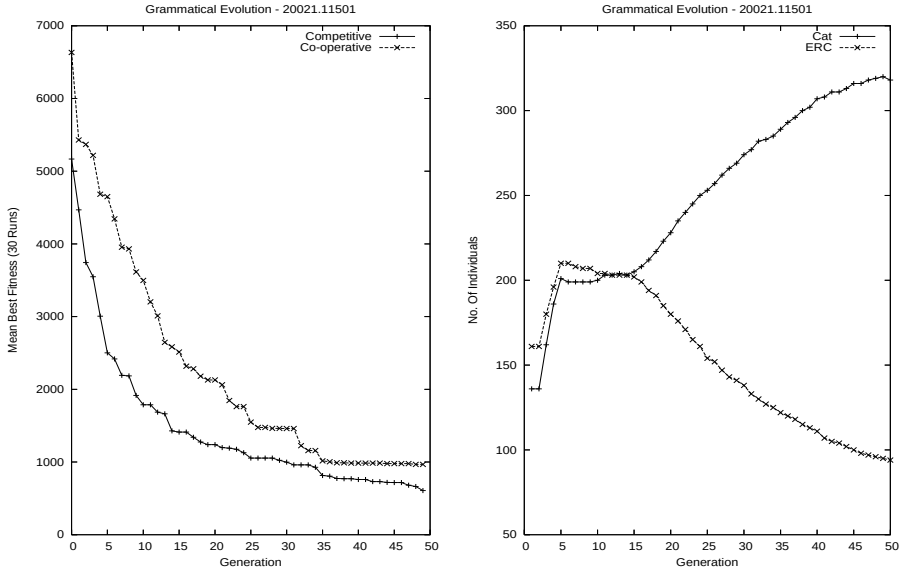

Figure 9 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each the constant generation methods (right) for the competitive grammar.

5.2.2 Finding Dynamic Real Constants

Fig. 10, displays the results for the dynamic experiments. Here we can see a similar trend to that seen in section 4.3.2. Again the Persistent Random Constants method gains a stronger position within the population while the target is within its range. The difference here is that once the target leaves this range the the Concatenation method begins to gain a bigger share of the population and ends up with a slight majority at 218 to 203. It can also be noted that a higher rate of evolution occurs in these experiments when the target goes outside the Persistent Random Constants range. This combined with the higher frequency of Concatenation individuals would suggest that the ability for the Concatenation method to create expressions is directly responsible for the improvement in the rate of evolution across both grammars.

Comparative analysis of the grammars at generations 10, 20, 30, 40 and 50 using a t-test and bootstrap t-test reveal a statistical significance in the difference in results at generations 10 and 40. No other transition generations showed a statistically significant difference.

5.2.3 Finding the Equation for the Area of a Circle

In Fig 11, it can be seen that the Digit Concatenation produces superior fitness over Persistent Random Constants. By the final generation Digit Concatenation

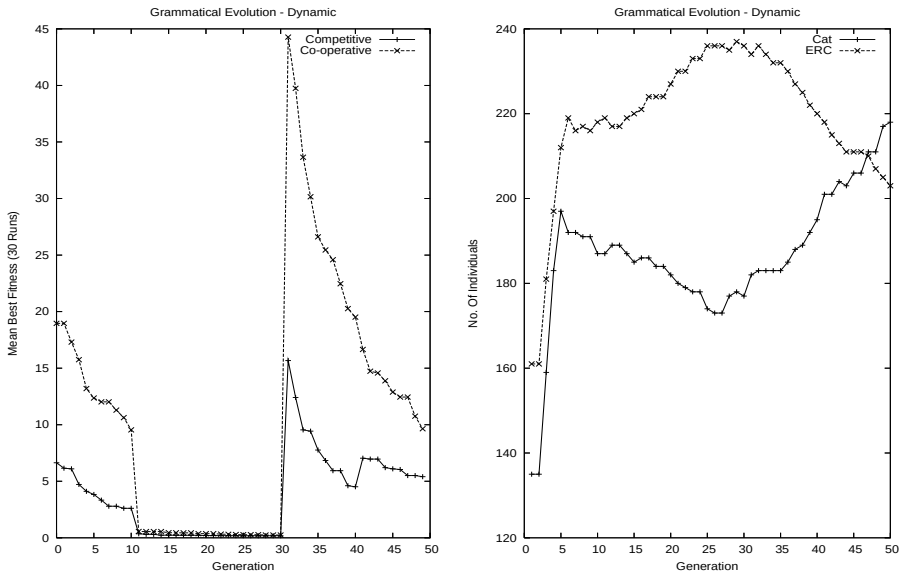

Figure 10 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each the constant generation methods (right).

produces an average fitness of 12489 compared to 439226 for Persistant Random Constants.

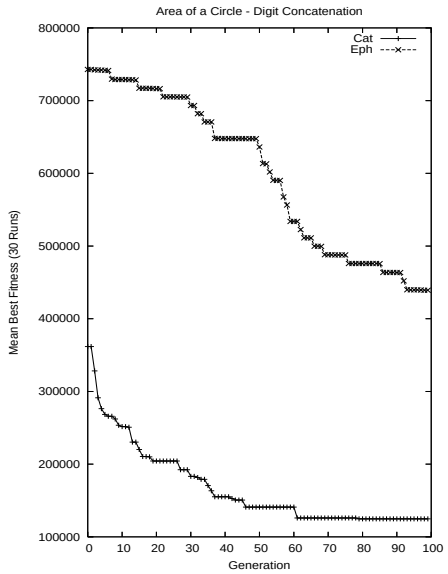

Figure 11 Mean best fitness values (lower values are better) plotted against generations, where fitness is the cumulative difference of each individual for 100 radii to the correct area.

5.2.4 Discussion

These experiments have focused on the direct comparison of the Concatenation and Persistent Random Constants methods while introducing an extra feature to Concatenation, the ability to create and evolve expressions in conjunction with the numbers. This added feature is beneficial to Concatenation. As this is the essential difference between the experiments here and in section 4, the results suggest that allowing Concatenation to produce expressions enables it to not only achieve a higher degree of accuracy in reaching the target but also allows it to be more flexible in a dynamic environment with the results for the dynamic experiment clearly outperforming those in section 4 and the results for finding the equation for the area of a circle reinforce this message. In order to gain direct perspective of the improvement in Concatenation when it is provided with the ability to create expressions Fig. 12 provides graphs which compare Concatenation using expressions with pure Concatenation. In these graphs we see an improvement in performance over the pure Concatenation results in both the static problem and the dynamic problem.

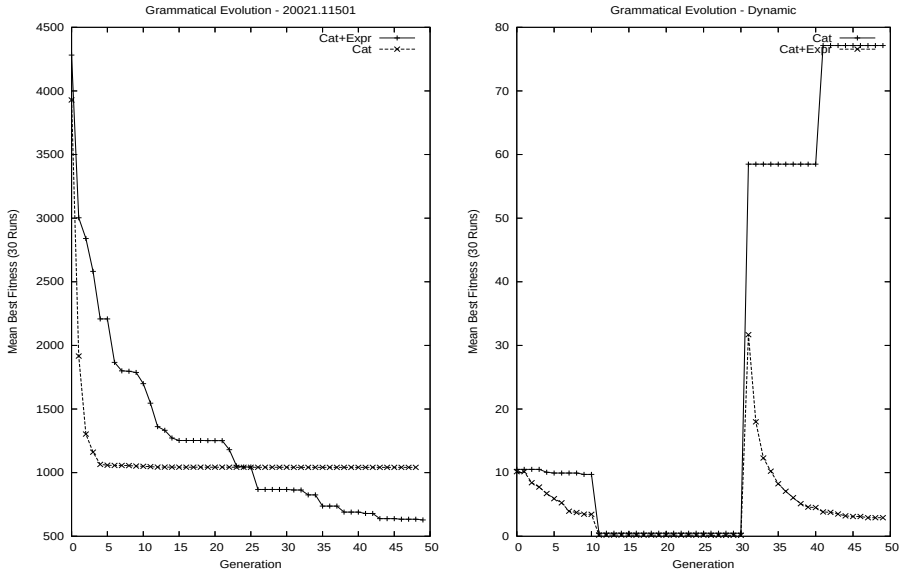

Figure 12 Mean best fitness values (lower values are better) plotted against generations for 20021.11501 (left) and the dynamic experiment (right).

6 Conclusions

The objective of this study was to identify the best method for constant creation and adaptation. To this end three methods in particular; Traditional, Digit Concatenatin and Persistant Random Constants, were examined on a series of benchmark problems. Through out these experiments the Concatenation method was seen to produce the best results with more regularity than all other methods investigated. In section 3 a pure Concatenation grammar exhibited better performance across most of the problems.

Section 4 presented a combination grammar where the choice of the different methods was left up to the evolutionary process. Here Persistent Random Constants were seen to grow and occupy the majority of populations on average in all experiments. However this majority did not translate to it producing the best individuals as Concatenation produced the most best fit individuals for the static problems and the logistic equation problems. Among experiments with the exclusive grammars the Concatenation method provided superior fitness for the static problems again with Persistent Random Constants giving better fitnesses for the dynamic problems.

In section 5 the Concatenation method is extended to give it the ability to evolve expressions. This proved to significantly improve fitnesses in the dynamic problem and in contrast to section 4 saw the Concatenation method grow to take up a majority position within the population for the static problem from an early stage. An interesting story is told by the population graph for the dynamic problem where we see the Persistent Random Constants method consume a majority of the population while the target was within its range but the trend peaks and reverses at generation 30 when the target moves outside its range. The result by the final generation gives a slight majority to the Concatenation method.

These experiments have examined different problems where static constants both complex and simple were targeted, dynamic problems with large and small variations in targets as well as oscillating targets, a co-efficient to a chaotic equation and symbolic regression. Considering the results, it would appear that a constant creation grammar which provides the Concatenation method with the ability to create expressions is the most advantageous method explored. Its ability to constantly introduce new constants to the system, take consistent evolutionary steps towards targets and produce a higher proportion of best performing individuals in comparative tests mark it apart from the other methods explored.

In the future we wish to extend this study to include an investigation of meta-grammars for constant creation, particularly as the meta-grammar approach adopted in Grammatical Evolution by Grammatical Evolution O'Neill and Ryan (2004) has shown promising results in dynamic environments. Additionally, we would like to investigate the use of the syntax of scientific notation as a grammatical method of generating very large or very small numbers quickly, and also to investigate the use of dynamic modifications to the persistent random constants using a dynamic grammar. In this way we could define a set of insertion and deletion operators that can add or delete constants, and also some mutation operators that could manipulate the stored values in the spirit of the perturbation methods described earlier.

References

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.

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 Arti-

ficial Intelligence and Cognitive Science, pp. 165-170, edited by M. O'Neill, R. Sutcliffe, C. Ryan, M. Eaton and N. Griffith, Berlin: Springer-Verlag.

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.

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

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.

Angeline, P., 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.

Evett, M. and Fernandez, T. (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.

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

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

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

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.

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.

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.

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.

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

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.

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

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

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.

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

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

O'Neill, M., Brabazon, A., and Ryan, C. (2002). Forecasting market indices using evolutionary automatic programming: A case study. In Chen, S.-H., editor, *Genetic Algorithms and Genetic Programming in Economics and Finance*. Kluwer Academic Publishers.

Brabazon, A. and O'Neill, M. (2004). Evolving Technical Trading Rules for Spot Foreign-Exchange Markets Using Grammatical Evolution. *Computational Management Science*. Springer, 2004.

Iba, H and Nikolaev, N. *Genetic programming polynomial models of financial data series*, Proceedings of the 2000 Congress on Evolutionary Computation CEC 2000, IEEE Press, pp. 1459–1466.

Nikoaev, N. and Iba, H. *Regularization Approach to Inductive Genetic Programming*, IEEETransactions on Evolutionary Computing 54 (2001), no. 4, pp. 359–375.

Keijzer, M. *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.

Ryan, C. and Keijzer, M. *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