Chapter 5
Constant Creation and Adaptation in Grammatical Evolution

Fundamental to many applications of Genetic Programming [112, 113, 22, 152] is the generation and use of constants. Hence the discovery of an efficient means for creating diverse constants is important. When the domain in which a GP system operates is dynamic, an even greater level of adaptability is required as it may become necessary to generate novel constant values with changes in the environment.

This chapter introduces and explores two novel constant-generation schemes; *Digit Concatenation* and *Persistent Random Constants*, as a means of generating constants in GE. The impetus for the introduction of these schemes is to address the existing weaknesses in the state of the art in GP for the generation and adaptation of constants, and to explore the potential of alternative representations in adaptation. While current schemes perform satisfactorily in static scenarios where a level of convergence is required, by migrating them to dynamic environments, limitations are encountered. This chapter explores the different constant creation methods with the aim of identifying a flexible and adaptive constant creation approach that will satisfy the extra requirements of evolving in a dynamic environment.

Following an introduction, Section 5.2 outlines existing techniques used in Genetic Programming to create constants. Section 5.3 examines the performance of the Digit Concatenation method in comparison with the traditional method for generating constants within GE. Section 5.4 compares the Digit Concatenation method with another novel method of constant creation based on Ephemeral Random Constants (ERC). Section 5.5 takes this comparison further by providing the Digit Concatenation method with the ability to evolve expressions. An analysis of constant representation and its impact on problem difficulty is detailed in Section 5.6. Finally Section 5.7 arrives at conclusions as to which method of constant generation is best.

5.1 Introduction

Integral to the proper functioning of most GP solutions is the incorporation of constants: numbers are passed as parameters to functions, to scale return

I. Dempsey et al.: Foundations in Gramm. Evol. for Dyn. Environ., SCI 194, pp. 69–104.
springerlink.com

values, as constants in expressions and in many other roles. Evolutionary Algorithms (such as GAs and Evolutionary Strategies) have been shown to be successful in evolving specific constants as parameters to functions. However, in GP, constants are generally co-evolved in line with the structure and other program primitives. Little research has been undertaken to address this area. Specifically, Koza states that:

"... the finding of numeric constants is a skeleton in the GP closet... [and an] area of research that requires more investigation..." [63]

The issue Koza refers to is that constants in GP are represented as terminal nodes on the GP tree structure and, therefore, are unaffected by the canonical GP operators of crossover and mutation. This is because these are applied only to the tree structure itself. The result then is that new values are only arrived at through expressions between these nodes. The problem is magnified by the fact that constants in GP can be evolved out of the population as evolution progresses, leading to a reduction in the diversity of contants available to individuals. A special form of point mutation must be introduced to allow the values of the constants in any one node of a tree to be modified. Otherwise the generation of new constants can only be achieved through the recombination of existing constants in expressions.

In dynamic problems, the issue becomes more acute as the environment may shift significantly over time. This necessitates a greater level of adaptability among the constants or indeed the generation of new constants. Considering this, a number of requirements emerge as to what would be desirable in a constant generation mechanism for dynamic environments.

i. **Static Targets:** The mechanism naturally should be able to quickly and efficiently evolve solutions for static problems. Evidence of this will serve as a control and demonstrate that the mechanism works.
ii. **Types of Change:** All the types of change described in Chapter 3 can require the generation and adaptation of constants. Therefore the new mechanism should be able to perform under these circumstances. Reflecting upon the approaches outlined in Chapter 3, a potential solution may be able to maintain a dispersed level of diversity in order to achieve or allow a level of problem decomposition to form building blocks of useful relevant constants.
iii. **Adaptable:** The constants themselves should be adaptable so as to permit evolution to alter expressions by small degrees. To enable this, a constant generation mechanism must allow the actual numbers to be evolved themselves so as to conduct local searches of promising constants. This also ties in with Evolvability as outlined in Section 3.3.6. Such a mechanism should exhibit a high level of *latent evolvability* identified by Reisinger [179].
iv. **New Constants:** The ability to generate new constants is imperative in dynamic environments. The presence of such a feature will allow evolution to explore new areas of the solution landscape and also maintain diversity.

v. **Large and Small Constants:** A mechanism for generating constants for an evolutionary system operating in a dynamic environment should be equipped with the flexibility to generate numbers of any size – not just within a fixed range, decided upon a-priori.

This chapter will aim to highlight limitations in existing approaches to constant creation and adaptation. Two novel constant generation methods will be introduced which aim to address the above requirements for constant generation mechanisms for dynamic environments. While this chapter does provide coverage over different types of dynamic environments, it is not exhaustive and experimentation in later chapters will broaden the scope to simulate more complex types of change and different deterministic types. Over the course of the experiments undertaken, the most appropriate form of constant generation will be identified.

5.2 Constant Generation in GP

Ephemeral Random Constants are the standard approach to constant creation in GP, having values created randomly within a pre-specified range at the initialisation of a run [110]. The terminal $\Re$ is specified in the GP terminal set. At generation zero, whenever this symbol is encountered at a leaf node on the tree representation it is replaced with a random number generated within the pre-specified range. These values are then fixed throughout a run, and new values can only be arrived at through combinations of these values and other items from the function and terminal set in expressions.

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 [207] 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 [5]. 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 [63] 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 [96, 143, 102] 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 are explored in the neighbourhood.

A study in [194] 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.

Banzhaf [9] notes the *"inflexible approach"* adopted in Koza's ERC and uses a genotype-to-phenotype mapping to interpret 10 bits as a natural number in the range 0 to $2^{10} - 1$. Then, depending on the prior symbol being consumed, this random number is mapped to an interval of real or integer constants. The benefit of this approach is that now standard mutation and crossover operators can be applied directly to the genotype, which forms the numerical phenotype. Effectively, the constants become parameters that are coded on the genotype.

With the exception of the final two approaches, each of these methods uses Ephemeral Random Constants as its base with the focus on changing the original random values by small amounts to improve fitness. None of these approaches, however, address a fundamental flaw in ERC. The flaw is that once a constant is evolved out of the population of solutions, it cannot be re-introduced to the population at a later stage. The phenotypic representation of GP does not facilitate a mechanism whereby formerly successful constants can be recalled. This leads to an inevitable decline in the diversity of constants that are available for solutions to exploit, unless a genetic operator is introduced to address this specific problem. All of these approaches also suffer from the weakness that the constants are generated within a certain range, which is built into the system prior to its execution. Daida et al., have demonstrated that the standard ERC approach to constants can escalate problem difficulty on certain problems when the range of ERC's provided increases [48]. This can then present issues where the problem is dynamic in nature.

GE can borrow from the experience of GP by extending the established methodology and introducing a new form of constant creation that potentially addresses the issue of beginning an evolutionary run with a fixed range of constants and offers the possibility of creating new values over the course of a run. With this in mind, the utility of this novel approach is determined by examining it under GE's capacity to create and adapt constants in isolation in order to gather a clear view of its behaviour and relative performance.

The experiments in this chapter are focused on identifying the best method for constant creation and adaptation with a emphasis on dynamic environments. In doing this,r the opportunity is also presented to examine the behaviour of the evolutionary process, and GE itself, in dynamic environments. This is useful as the types of dynamic problems tackled here present less complex types of change and so make the analysis of GE's behaviour more amenable.

5.3 Evolving Constants Using Digit Concatenation

We now introduce constant creation in GE illustrating the standard approach before detailing the new digit concatenation representation. The objective of this section is to determine whether Digit Concatenation can outperform the traditional expression-based approach to constant creation in GE.

5.3.1 Traditional Constant Creation in GE

The traditional approach to constant generation in GE relies upon defining 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 that 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 ten 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 ) )
```

5.3.2 Digit Concatenation in GE

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

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

which could produce a sample output of

```
20034.51
```

In this grammar the digits zero to nine are used as the fundamental building blocks to create all other numbers simply by concatenating them together. By providing GE with the ability to create and evolve constants in this fashion, a number of the shortcomings of ERC in GP are potentially addressed.

- The problem of constant creation is approached from a fundamental representational level by addressing how evolution can create and adapt constants rather than providing it with a fixed pool.
- The requirement in ERC of generating the initial pool of numbers within a pre-specified range is overcome. Through using Digit Concatenation, constants of any size or floating-point precision can be created with evolution determining their usefulness.
- The constants themselves can be adapted and evolved in order to conduct local searches around relatively fit phenotypes.
- The facility exists to re-evolve previously successful constants.

Such features can then provide the ability for GE to maintain a satisfactory level of diversity of constants and allow the formation of useful strings of constants that can spread through the population, linking back to the approaches outlined in Chapter 3.

5.3.3 Problem Domain and Experimental Approach

Two grammars were constructed, one using Digit Concatenation and the other incorporating the Traditional method. The aim of these experiments is to examine the performance of GE in the creation and adaptation of constants in isolation, without involving any of the complexity of evolving the constants in line in programs. This is left to subsequent chapters. In this manner the experiments focus on the issue in its own right before adding context. Considering the requirements laid out in Section 5.1 and the types of change identified in Chapter 3, the performance of these grammars is measured on three different types of constant creation problems. The constant creation problems include: finding a static real constant in order to determine if the mechanism is functional under static conditions; satisfying the first requirement in Section 5.1; and finding dynamic constants where Markov and Deterministic types of change are tackled. In this manner these problems also represent a simplified testbed for the analysis of GE in dynamic environments while at the same time provding insights on how to create and evolve constants.

Finding a Static Real Constant

The aim of this problem is to evolve a single real constant and present a vanilla-type static problem to the mechanisms under examination. 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.

Finding Dynamic Real Constants

These experiments examine problems under two categories of change. The first test presents a Markov type problem. The target shifts through a sequence to a different real value every 10 generations. The targets in this experiment are a random sequence of values: 24.35, 5.67, 5.68, 28.68, and 24.35. For the next experiment, using the same fitness function, a second set of targets with a Deterministic oscillatory type of change is presented where the targets alternate between 24.35 and 5.67 every 10 generations. The aim of these problems is to make it easier to compare the different constant generation methods in terms of their ability to adapt to a changing environment. Further, the problems investigate their behaviour in the event of changes on both a small and a large scale. As in the static problem, fitness in this case is the absolute difference between the target and evolved values, with the goal being the minimisation of this difference.

The Logistic Difference Equation

A more complex type of regression problem is also tackled, where GE is tasked with evolving a coefficient for the logistic difference equation which exhibits chaotic behaviour. With systems exhibiting this behaviour, 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 [91]. 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 [142, 38, 195]. In this time series, information is drawn from a simple quadratic equation, the logistic difference equation.

$$x_{t+1} = \alpha x_t(1 - x_t) \qquad x \in (0.0, 1.0) \tag{5.1}$$

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 $\alpha < 3.0$ [195]. 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 [129]. In this study, three time series are generated for differing values of α. The choice of these values is guided by [129], 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}). Because of this, there will be a strong feedback from the problem back to GE with slightly more accurate solutions achieving a much stronger fitness and so standing a stronger chance of being selected for reproduction. The goal of this experiment is to evolve closely accurate individuals.

Constant Creation Grammars

The grammars adopted are given below. The Digit 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. The Traditional grammar is only provided with the basic mathematical operators to simplify the analysis of its behaviour.

Digit 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
```

5.3.4 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 in order to examine the efficiency of the methods. It also included 100 generations for the logistic difference equation, adopting one-point crossover at a probability of 0.9 and bit mutation at 0.1. Roulette selection and a generational rank replacement strategy of 25% was adopted where the weakest performers were replaced by the newly generated offspring. The GE wrapping operator was allowed to perform 8 wrapping operations before terminating and a binary 8-bit encoding was used for the individuals. A random initialisation was conducted up to 104 bits long with evolution allowing this to vary after the first generation. A section describing the results of each of the experiments is provided next, followed by a discussion of their implications.

Finding a Static Real Constant

On all three instances of this problem, a t-test and bootstrap t-test [26] (5% level) on the best fitness values reveal that the Digit Concatenation grammar significantly outperforms the standard expression-based approach. Performance statistics for each grammar are given in Table 5.1, and a plot of the mean best fitness at each generation for the three targets can be seen in Figure 5.1.

Table 5.1 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 significant margin. This demonstrates that a grammar with the Digit Concatenation approach to constant creation is significantly better at generating larger numbers. It is worth stressing that larger numbers could just as easily be large whole numbers or numbers with a high degree of precision (real).

Finding Dynamic Real Constants

For the first instance of this problem where the successive target constant values are 24.35, 5.67, 5.68, 28.68, and 24.35 over the course of 50 generations, performance statistics are given in Table 5.2, and a plot of mean best fitness values for each grammar can be seen in Figure 5.2.

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 Digit Concatenation grammar up to generation 30. However, 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 a similar trend is seen. In this case, the Digit Concatenation grammar is significantly better at the 5% level than the Traditional grammar at each of the 10, 20, 30, 40, and 50 generations. However, this difference is decreasing

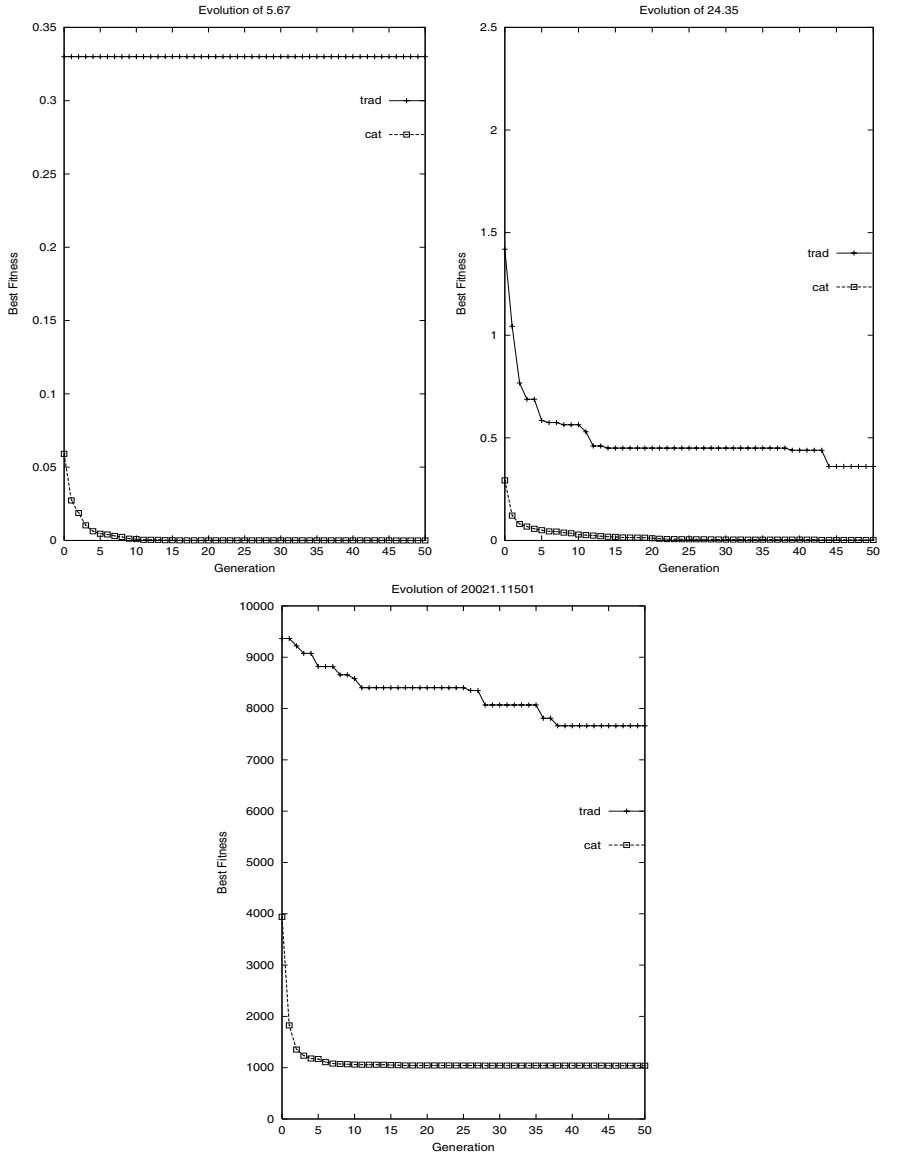

Fig. 5.1 Mean best fitness values (lower values are better) plotted against generations for each of the four grammars. Target values are 5.67 (top left), 24.35 (top right), and 20021.11501 (bottom).

over time. From the results of both of these dynamic problem instances, there are clearly adaptive advantages to using the Digit Concatenation grammar over the traditional expression-based approach.

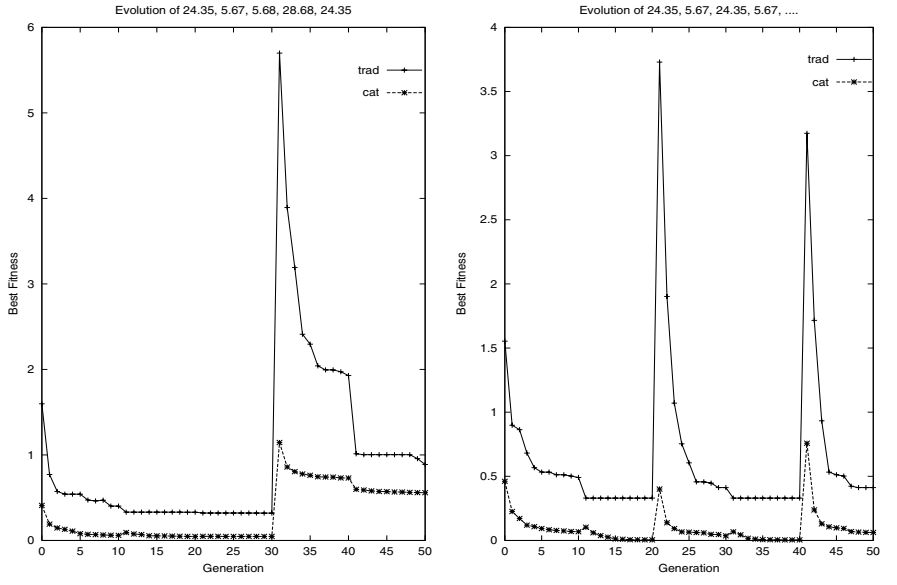

Fig. 5.2 Mean best fitness values (lower values are better) plotted against generations for each of the three grammars. Target values are 24.35, 5.67, 5.68, 28.24.35 (left) and 24.35, 5.67,... (right).

Table 5.2 Statistics for the best fitness values (lower value is better) in 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

The Logistic Difference Equation

The results for all three instances of this problem can be seen in Table 5.4 and Figure 5.3. Statistical analysis using a t-test and bootstrap t-test (5% level)

Table 5.3 Statistics for the best fitness values (lower value is better) in 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

reveal that the Digit Concatenation grammar significantly outperforms the traditional constant creation approach on each problem instance, successfully rediscovering the target α in each case.

Table 5.4 Statistics for the best fitness values (lower value is better) in the logistic difference 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

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 to evolve real numbers within its range or to alter expressed values in amounts of less than one. 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

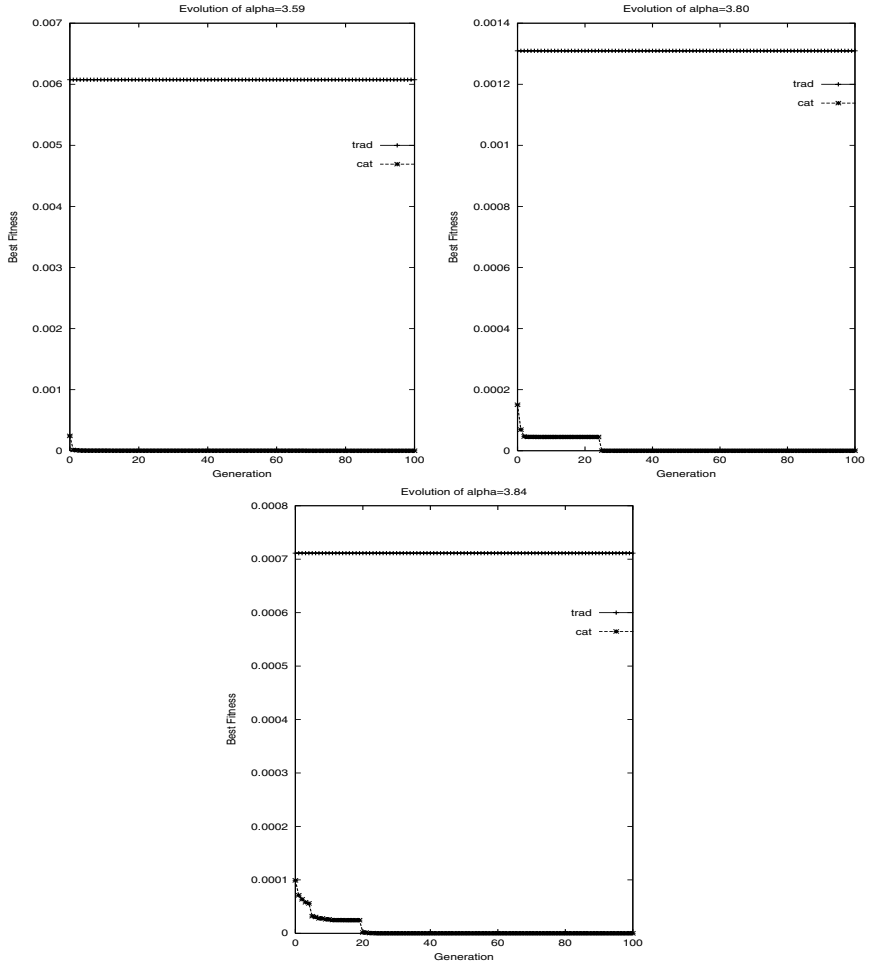

Fig. 5.3 Mean best fitness values (lower values are better) plotted against generations for each of the three grammars. Target values of α are 3.59 (top left), 3.80 (top right), and 3.84 (bottom).

to that of the target through different expressions. In order to simplify analysis, take, for example, the average fitness attained for the static experiment with target 5.67 (see Table 5.1). The average best fitness for this problem with the Traditional approach to constant creation was 0.33, with a standard deviation of 0 resulting in a flat-lined fitness series, meaning that each run settled upon a solution of 6 as the best phenotype. The problem encountered by the traditional approach here is that using the digits zero to nine and the operators provided, the task of adapting a relatively fit individual with one codon and a phenotype of 6 to 5.67 is too complex. In order to arrive

at the correct phenotypic solution, an individual involving a number of extra codons would have to be evolved to produce an expression that resulted in the correct value. Slight deviations from a correct expression, or indeed the phenotype of 6, would produce fitnesses inferior to the simple one-codon solution and stand a lesser chance of being selected for reproduction. On the other hand, the concatenation approach can build upon a one-codon solution by concatenating the decimal point and other digits to arrive at the correct solution. As the results suggest, this was attainable on each run of the system. Digit Concatenation presents an incremental approach for the evolutionary search where the fitness may improve with the addition of each extra codon demonstrating a greater level of evolvability. Considering this, it can be said that while it is good at attaining a reasonable fitness for real targets within its range, forming expressions using integers to get real values proves too complex for the Traditional approach as it is less evolvable.

The Digit Concatenation approach, however, proved to be successful in evolving the correct solution for the static experiments across all runs. In the dynamic Markov type changes and the logistic difference equation type change good results were also arrived at.

Reflecting on 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 this approach in its own right. The ability to mix different constant types illustrates the flexibility of the grammar-based approach to GP. To this end, a version of Ephemeral Random Constants for GE is investigated in the following section.

5.4 Analysis of Digit Concatenation and Persistent Random Constants

In Section 5.3, the Digit Concatenation method displayed superior performance over the Traditional technique for evolving constants in GE. In this section the Digit 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 Digit Concatenation, a novel grammar, defined as the 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 that only allows the use of one method exclusively. The preference of the evolutionary search is then examined across a range of constant generation problems.

5.4.1 Persistent Random Constant Creation in GE

In Section 5.2, a description of Ephemeral Random Constants in GP was given. Here, a form of ERC is introduced known as Persistent Random

Constants (PRC). 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. This 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, PRC uses a larger number of randomly generated constants that persist for the lifetime of the run with the implication of better coverage of the constant search space.

5.4.2 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 finding a coefficient for the logistic difference equation. The problems used in this section, though similar to those in Section 5.3, 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. Most of the targets in Section 5.3 resided within this range. This is done to examine further the issue discussed in Section 5.3.4, where the Traditional approach was observed to settle on local minima, and also to examine the Persistent Random Constant's ability to evolve targets outside its range. These three methods of constant creation are combined in one competitive combination grammar which selects one method exclusively for an individual. In conjunction with experiments using this grammar, experiments are also conducted that use each method on its own in a grammar, allowing comparative benchmarks to be drawn from the previous section.

Finding a Static Constant

The aim of this problem, as before, is to evolve a single integer constant in order to verify the functionality of the proposed grammar. For these experiments, two constants were selected: a simple integer value within the range of the Persistent random constants, 50; and a complex floating-point real number outside the range of the Persistent 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.

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, presenting Markov and Deterministic type problems as in the previous section. 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.

The Logistic Difference Equation

This problem is used in the same manner with the same parameters as in Section 5.3.

Constant Creation Grammar

Three constant generation techniques are employed within the same grammar provided below.

```
<exp> ::= <value>
<value> ::= <trad> | <catR> | <persistent>
<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 (`<catR>`) only allows the creation of constants through the concatenation of digits and the dot character for real numbers. 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. These are then directly

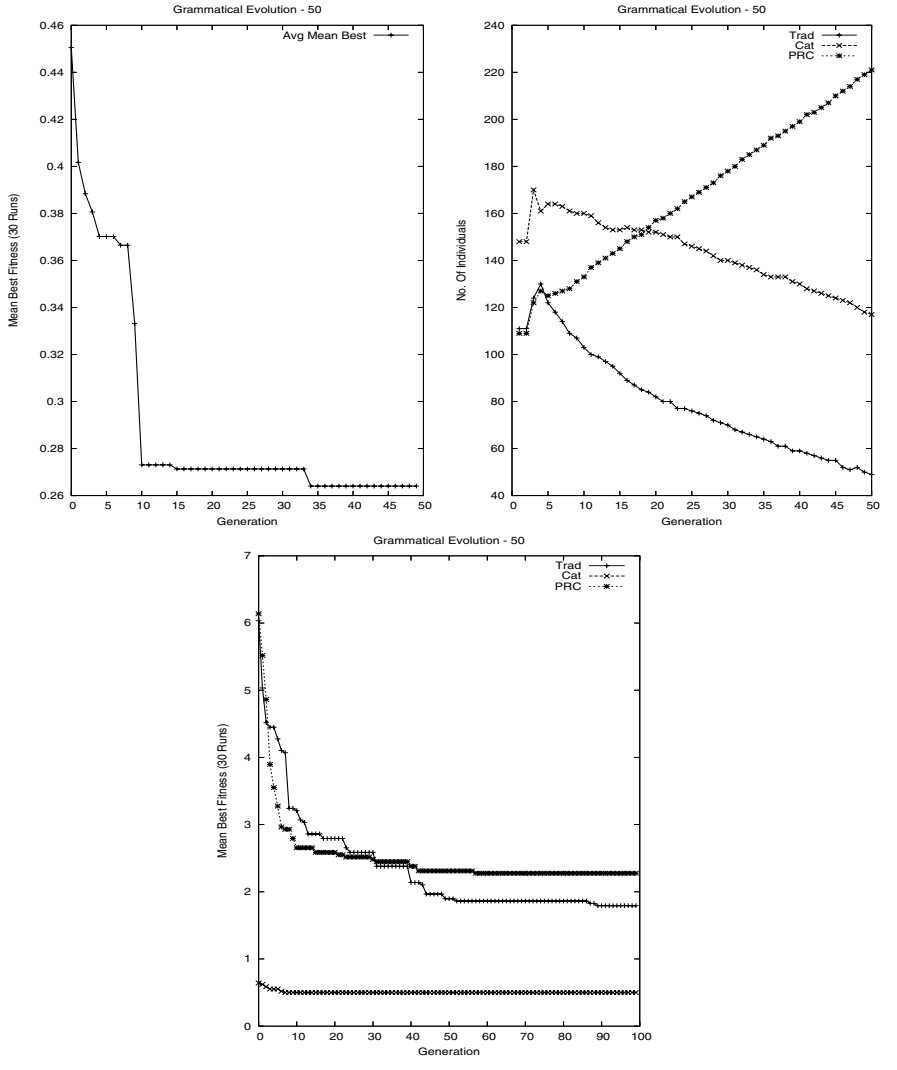

Fig. 5.4 Mean best fitness values (lower values are better) plotted against generations (top left), the number of individuals that use each of the three constant generation methods (top right), and a comparison of the performance of the exclusive component grammars (bottom)

incorporated as choices for the non-terminal `<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 that permits the exclusive choice of one of these methods for each individual.

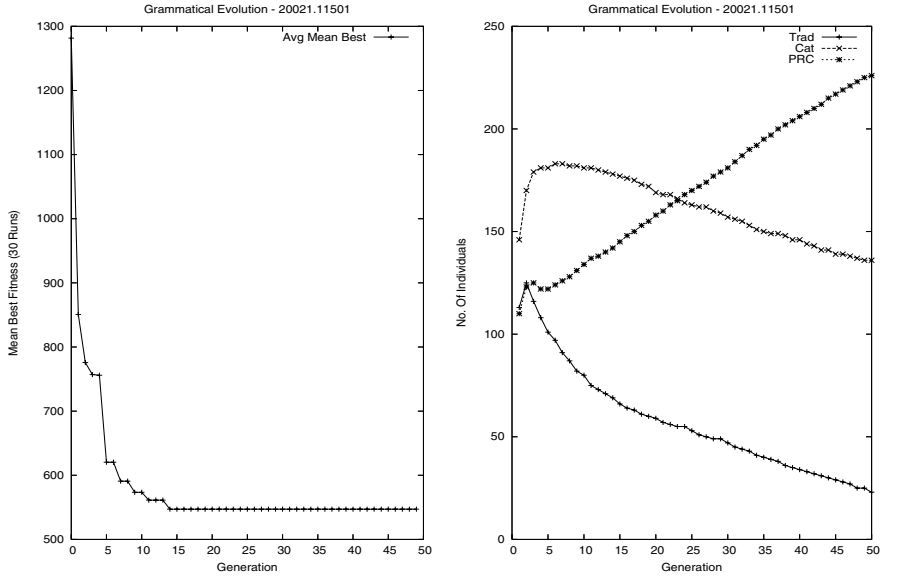

Fig. 5.5 Mean best fitness values (lower values are better) plotted against generations (left) and the number of individuals that use each of the three constant generation methods (right)

5.4.3 Results

For every problem instance, the parameters used and the number of runs conducted were the same as in Section 5.3.4.

Finding a Static Constant

For evolving the static constant 50, the results presented in Figure 5.4 indicate a preference by GE for the PRC in this problem. By the final generation, on average across thirty runs, GE evolved 221 individuals using the PRC method, against 117 and 59 for the Digit Concatenation and the Traditional methods respectively. Out of the 30 runs conducted of the best-performing individuals in the final generation, 60% had evolved a Concatenation individual and 40% a PRC individual.

Among the experiments incorporating each constant creation method exclusively as presented in Figure 5.4, the benefits of the Digit Concatenation method are highlighted for this problem. Over the course of 30, runs Digit Concatenation produced best performers with an average fitness of 0.50024 compared against 1.7931 and 2.27586 for the Traditional and PRC methods respectively.

Figure 5.5 presents the results for evolving 20021.11501. Here PRC is again seen to grow and dominate the population with 226 members against 136

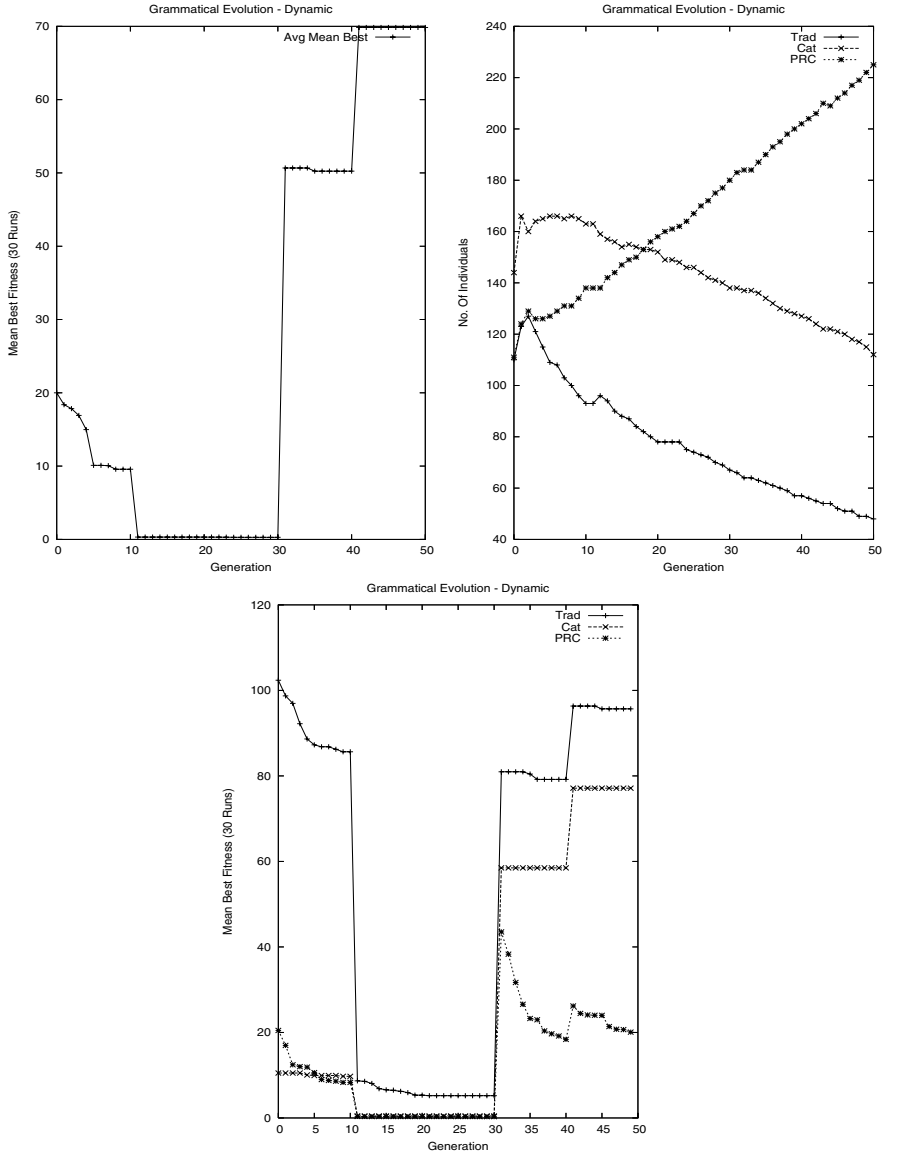

Fig. 5.6 Mean best fitness values (lower values are better) plotted against generations (top left), the number of individuals that use each of the three constant generation methods (top right), and a comparison of the performance of the exclusive component grammars (bottom)

and 23 for the Digit Concatenation and Traditional methods, respectively. However, in this instance Digit Concatenation is the method used for 100% of the best individuals, yielding an average best performance of 547.217.

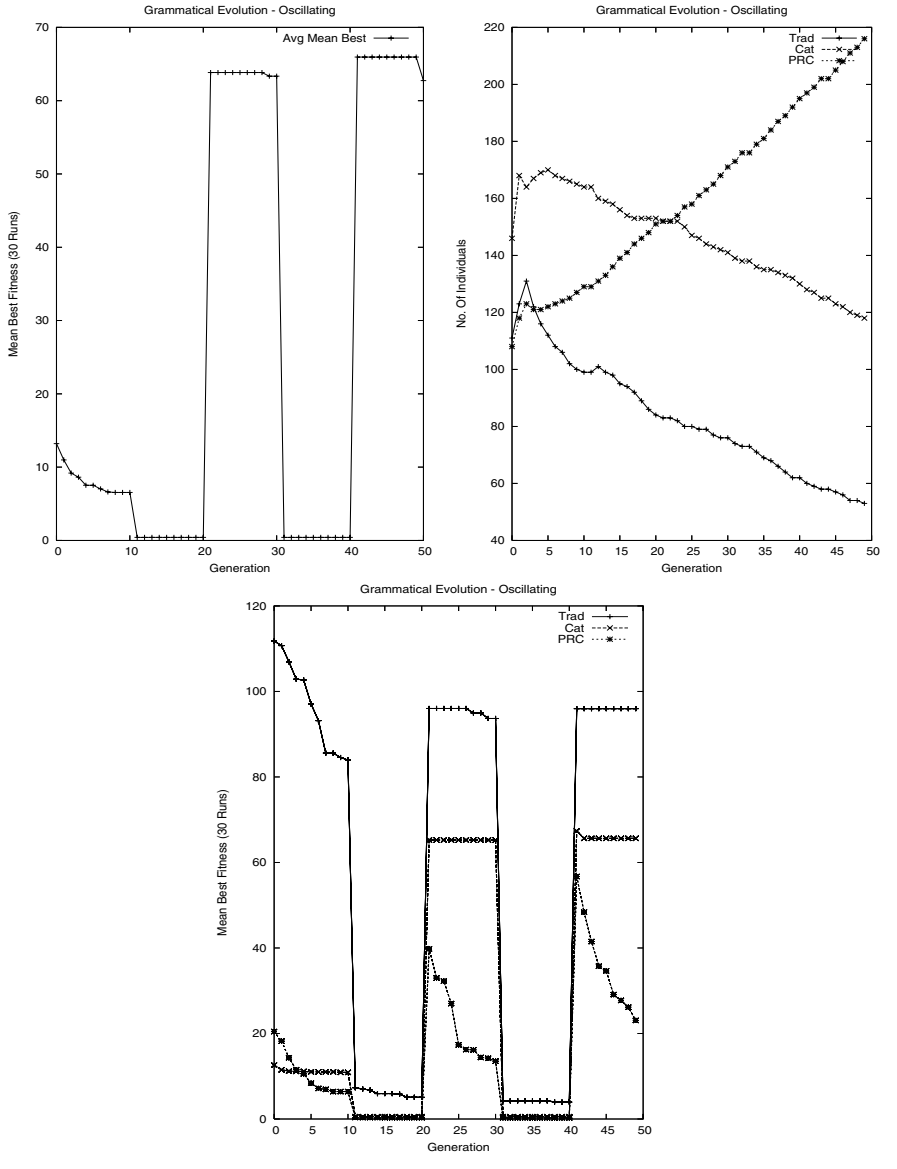

Fig. 5.7 Mean best fitness values (lower values are better) plotted against generations (top left), the number of individuals that use each of the three constant generation methods (top right), and a comparison of the performance of the exclusive component grammars (bottom)

In the experiments with exclusive grammars, the Digit Concatenation method was seen to provide the best average fitness at 1005.24, with the PRC method providing an average best fitness of 10070.5.

Finding Dynamic Real Constants

In Figure 5.6, 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 PRC gains a stronger foothold in the population over the course of the run, overtaking Digit Concatenation before generation 20 and at the same time presenting good fitness. However, at generation 30, where the target changes to 173.59, this fitness deteriorates significantly and remains poor. This suggests that while the target was within the range of the PRC, it was able to quickly attain a high fitness and a strong position in the population. However, it was unable to successfully evolve from this position once the target left its range.

In the single method grammars, however, the PRC 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 Digit Concatenation and Traditional methods present performances similar to the combination grammar's performance.

Results for the oscillating non-stationary problem instance are presented in Figure 5.7. In the second instance of this problem, where the target oscillates from 192.47 to 71.84 every 10 generations, a similar trend is noticed. Again by generation 20, PRC has 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 PRC method being able to take the larger evolutionary steps once the target changes.

The Logistic Difference Equation

Figure 5.8, presents a sample of the results for the logistic difference equation with α values of 3.59, 3.8, and 3.84, which were very similar across each of the values. Here the Digit Concatenation method gains the dominant position within the population as evolution progresses. The proportion of PRC individuals in the population is seen to approach the level of Digit Concatenation constants initially, as in the dynamic experiments. However, this time, as evolution progresses, the Digit Concatenation method gains the dominant position within the population. Among the best performing individuals for 3.59, 60% were PRC; and 40% were Digit Concatenation; for 3.8, 73% were Digit Concatenation and 27% were PRC and for 3.84, 80% of best individuals were Digit Concatenation with 20% were PRC. No Traditional individuals were the best performer in any test.

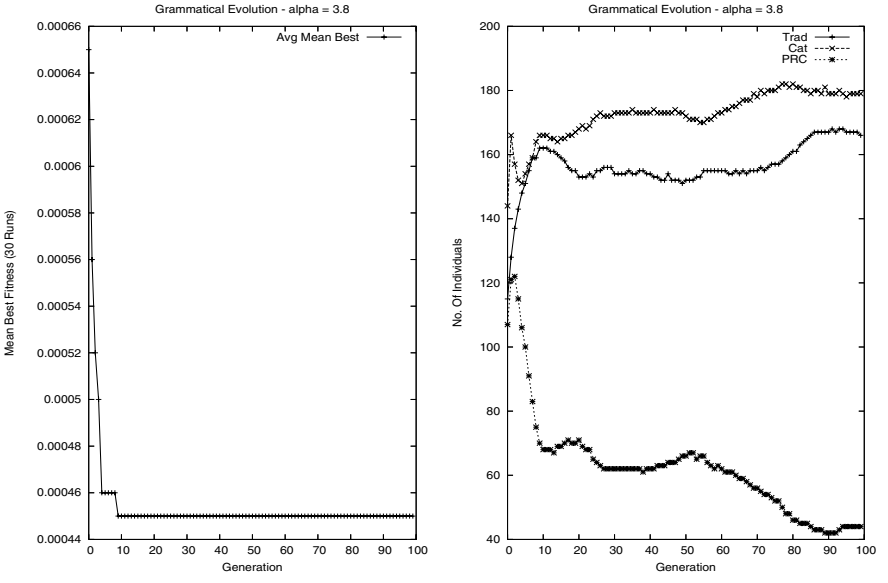

Fig. 5.8 Mean best fitness values (lower values are better) plotted against generations (left), and the number of individuals that use each of the three constant generation methods (right)

Discussion

Table 5.5 presents a summary of the the results from these experiments. A common feature seen in almost all the results across the problems examined in this section, with the exception of the logistic difference 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 that 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 5.3.4. Thus this method presents the evolutionary process with again a relatively large number of simple solutions, or 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 PRC method had already attained a strong postion in the population, gaining a certain reproductive momentum to the detriment of the Digit Concatenation and Traditional methods. In terms of evolvability the PRC method would also be able to attain reasonable fitnesses simply

Table 5.5 Summary of results including the percentage of population occupied by each method and the percentage of best individuals of each type by final generation. Note: The percentage of population figures do not add up to 100% due to individuals not mapping.

Exp	Metric	Trad	Cat	PRC
50	**Pop %**	12	23	**44**
	Best %	0	**60**	40
20021.11501	**Pop %**	5	27	**45**
	Best %	0	**100**	0
Dyn	**Pop %**	8	24	**43**
Osc	**Pop %**	7	23	**43**
3.59	**Pop %**	31	**37**	10
	Best %	0	40	**60**
3.8	**Pop %**	33	**35**	9
	Best %	0	**73**	27
3.84	**Pop %**	33	**36**	9
	Best %	0	**80**	20

by summing up two of its number terminals at the higher end of its range, requiring the exploitation of just three terminals or codons.

This explanation can then be related directly to the good performance seen by the Traditional method in attaining a larger proportion of the population in the logistic difference equation problems and entirely contrasting its performance in the other problems. In the logistic difference equation problem the target was within the range of the Traditional grammar and also within the range of PRC. 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 10 digits), all of which would have had reasonable fitness on their own, compared to the PRC method, which had 150 terminals dispersed evenly from 0 to 100. The Traditional approach presented a more evolvable set of choices for this specific problem. What held the Traditional method back in this case, however, is its difficulty in evolving fitter solutions that require small changes in the resultant value of a best-performer's expression. This problem was already discussed in Section 5.3.4. The Digit Concatenation method, on the other hand, offered greater evolvability in attaining fitter solutions in Section 5.3.4 for this problem, explaining how it could hold onto a majority position within the population on average.

In examining the static problem with target 20021.11501, a different situation is presented. In this problem the target was well outside the range of the PRC method yet it still grew to dominate the population. The behaviour of Digit Concatenation trend line in this instance, however, differs from the other problems. Here Digit Concatenation ends up with a proportion of the population that is just 12 less than what it began with on average. Considering that it provided 100% of the best-performing solutions, evolutionary pressure allowed it to maintain its population share. In the three-way evolutionary competition to gain population share the PRC method then grew its population share, to the detriment of the Traditional method. The Traditional method struggled to cope with the ease at which the PRC method could evolve much larger values with expressions involving its larger terminal values. Between the Traditional approach and PRC, the PRC method is seen to be more evolvable because it can sum or multiply out these larger terminals. PRC also has this advantage over the Digit Concatenation approach. However, because Digit Concatenation could accurately evolve the target, selection pressure allowed it to maintain its population share.

In summary, it can be seen that PRC presents an ability to take large evolutionary steps towards a target, while Digit Concatenation proved to be the most accurate in evolving its targets. The way in which both methods evolve their targets gives insight to this behaviour. PRC was able to achieve a high proportion of the population by evolving reasonably fit solutions with a lower number of codons through the use of expressions. Digit Concatenation, in comparison, had to construct a solution with the correct number of digits with a left-to-right dependency, whereby the first digit had the greatest impact on fitness. Having discovered a fit neighbourhood, Digit Concatenation then allowed a more finely grained approach to evolving the correct target accurately.

Both Digit Concatenation and PRC present relative advantages and disadvantages over each other: Digit Concatenation is accurate in evolving both large and small numbers though more slowly evolving while PRC is fast in attaining a good fitness though not as accurate. These are advantages that are desirable in constant generations mechanisms where dynamic environments are concerned. PRC has demonstrated an ability to quickly discover a fit neighbourhood, while Digit Concatenation can fine-tune a solution. Both advantages can be examined in terms of evolvability. While PRC can more easily evolve to a fit neighbourhood through multiplying or summing large terminals, Digit Concatenation can conduct evolutionary perturbation to more accurately find the target. However Digit Concatenation suffers from another weakness in that it cannot form a phenotype to produce the same value as certain fractions, for example a $1/3$, as it cannot concatenate digits to infinity. The next section will seek to address the evolvability weakness in the Digit Concatenation approach along with its inability to form factions by equipping it with an ability to form expressions and also exploring a grammar that allows both methods to cooperate.

5.5 Direct Comparison of Digit Concatenation and Persistent Random Constants

Section 5.4 demonstrates the superiority of both the Digit 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 both approaches, a further series of experiments were undertaken. This section compares the two methods using grammars similar to the previous section, along with grammars which use each approach exclusively. However, in these experiments the Digit Concatenation method is additionally given the ability to form expressions so that it can mimic the ability of the PRC approach to more easily evolve individuals to fit neighbourhoods. The mathematical operators adopted are again consistent with earlier experiments.

Below are the combination grammars derived from experiments in the previous section. The first is most similar to the previous section, except that it only uses the Digit 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 a symbolic regression problem to evolve the equation for 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’’
```

5.5.1 Experimental Approach

The experiments performed focus on three areas: one being 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, π, in the equation for calculating the area of a circle. This final experiment differs from previous experiments as having examined GE's ability to create and adapt constants, its ability to evolve an equation's form is now also examined.

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.

Finding Dynamic Real Constants

The same series of dynamic real constants, as used in section 5.4, are again considered; successive targets of 192.47, 71.84, 173.59, and 192.47 changing at the 10^{th} generation for fifty generations. Once more, the ability to compare results with the previous section is provided.

Finding the Equation for the Area of a Circle

For this section, a new problem is introduced. In this case two grammars are used that 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 $2 \rightarrow 102$, where the objective is to minimise the cumulative difference to the correct area across the 100 radii.

5.5.2 Results

For every problem instance, the parameters used and number of runs conducted were the same as in Section 5.3.4.

Finding a Static Constant

In this case, Figure 5.9 demonstrates that the Digit Concatenation method began to gain an upper hand on average within the populations at generation 13. Finishing at the final generation with a large majority of the population, Digit Concatenation has 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 Digit

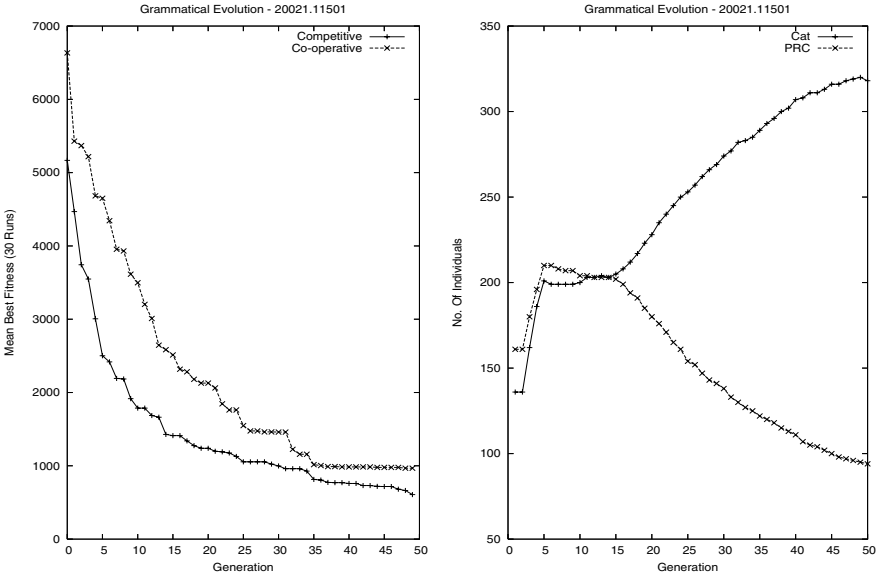

Fig. 5.9 Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each the constant generation methods (right) for the competitive grammar

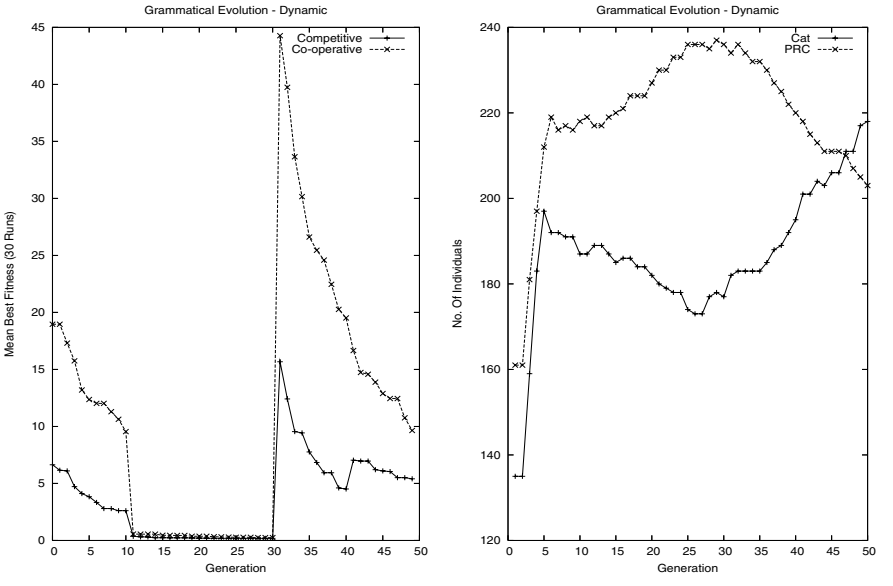

Fig. 5.10 Mean best fitness values (lower values are better) plotted against generations (left) and the number of individuals that use each of the constant generation methods (right)

Concatenation solution producing an expression that came to within 18.3872 of the solution. This solution is provided below.

```
20002 + 0.727829
```

Interestingly, the decimal and whole part are separated out into an expression. The whole part is also evolved on the more significant side of the phenotype according to GE's mapping process with the decimal part summed in on the right. 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.

Finding Dynamic Real Constants

Figure 5.10 displays the results for the dynamic experiments. A similar trend to that seen in Section 5.4.3 is presented. 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 Digit 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

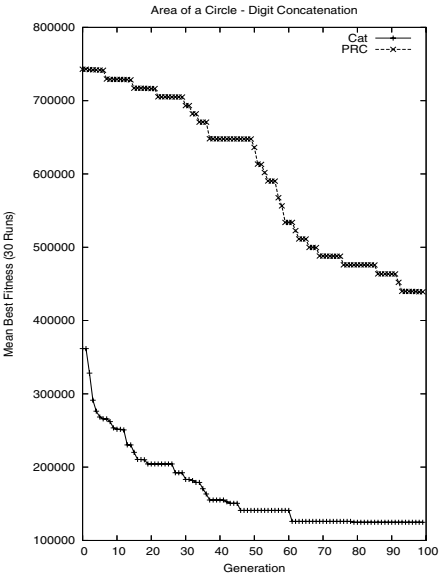

Fig. 5.11 Mean best fitness values (lower values are better) plotted against generations, where fitness is the cumulative difference of each individual for 100 radii to the correct area

when the target goes outside the Persistent Random Constants range. This combined with the higher frequency of Digit Concatenation individuals within the population would suggest that the ability for the Digit 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.

Finding the Equation for the Area of a Circle

In Figure 5.11, it can be seen that the Digit Concatenation produces superior fitness over Persistent Random Constants. By the final generation, Digit Concatenation produces an average fitness of 12489 compared to 439226 for Persistent Random Constants.

Discussion

These experiments have focused on the direct comparison of the Digit Concatenation and Persistent Random Constants methods, while introducing an extra feature to Digit Concatenation; the ability to create and evolve expressions in conjunction with the numbers. This added feature is beneficial

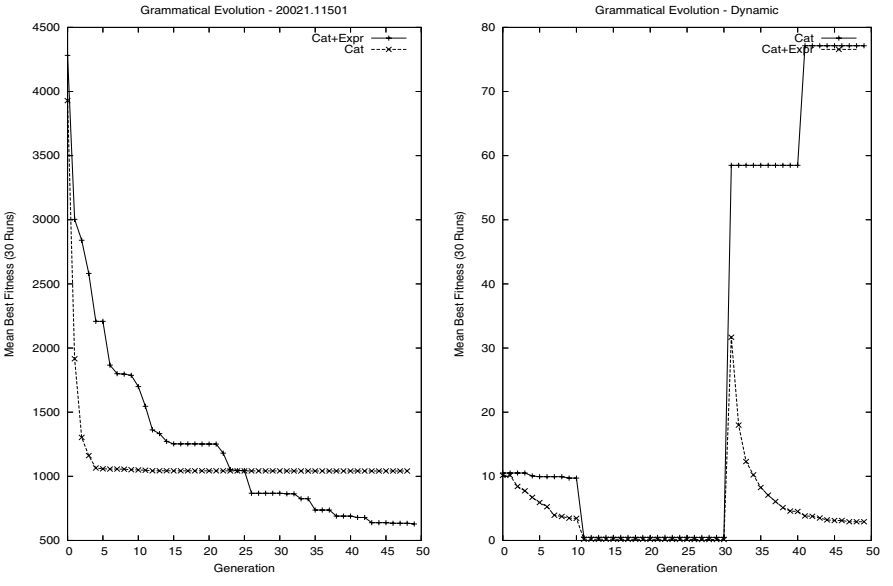

Fig. 5.12 Mean best fitness values (lower values are better) plotted against generations for 20021.11501 (left) and the dynamic experiment (right)

to Digit Concatenation. As this is the essential difference between the experiments both here and in Section 5.4, the results suggest that allowing Digit Concatenation to produce expressions enables it to achieve a higher degree of accuracy over the prior implementation. It also allows it to be more flexible in a dynamic environment, with the results for the dynamic experiment clearly outperforming those in Section 5.4 with positive evolutionary steps being taken at each generation after a target change. In order to gain direct perspective of the improvement in Digit Concatenation when it is provided with the ability to create expressions Figure 5.12 provides graphs that compare Digit Concatenation using expressions with pure Digit Concatenation. In these graphs, an improvement in performance over the pure Digit Concatenation results is seen in both the static problem and the dynamic problem.

5.6 Digit Representation and Problem Difficulty

In a recent series of experiments we turned to a tunably difficult problem, Binomial-3, as analysed for standard tree-based Genetic Programming by Daida et al [48]. Binomial-3 can be cast as a symbolic regression problem instance where the target function could take the following form:

$$1 + 3x + 3x^2 + x^3$$

There are a number of alternative representations for this target including:

$$(x+1)/(1/(1+(x/5)+(x/(1/x))))$$

$$(1+x)(1+2x+x^2)$$

$$1+x+x+x+x^2+x^2+x^2+x^3$$

Daida et al have demonstrated that this problem becomes exponentially more difficult for GP with Ephemeral Random Constants as the range of constants made available to GP increases. We wish to analyse the performance of the Digit Concatenation and Persistent Random Constant (PRC) representations on this problem to determine if they confer some advantage. The general form of the grammar adopted for this problem is presented below.

```
<expr> ::= (<op><expr><expr>) | <var> | <const>
<op> ::= + | - | *
<var> ::= x
<const>::= "constant generation method"
```

In the case of PRC 100 randomly generated constant values are generated in the allowed range. For Digit Concatenation, the grammars were range-specific, and an example of the grammars adopted is provided here.

```
For the range [0,5]
<int> ::= <nzdigit>.<digit><digit>
<digit> ::= 1|2|3|4|5|6|7|8|9|0
<nzdigit> ::= 0|1|2|3|4

For the range [0,100]
<int> ::= <nzdigit><digit>.<digit><digit>
        | <digit>.<digit><digit>
<digit> ::= 1|2|3|4|5|6|7|8|9|0
<nzdigit> ::= 1|2|3|4|5|6|7|8|9

For the range [0,5000]
<int> ::=<anzdigit><digit><digit><digit>.<digit><digit>
        | <nzdigit><digit><digit>.<digit><digit>
        | <nzdigit><digit>.<digit><digit>
        | <digit>.<digit><digit>
<digit> ::= 1|2|3|4|5|6|7|8|9|0
<nzdigit> ::= 1|2|3|4|5|6|7|8|9
<anzdigit> ::= 1|2|3|4
```

The experiments were implemented using GEVA [76] with the evolutionary parameter settings of population size 500, crossover rate 0.9, replication rate 0.1, maximum generations 200, roulette wheel selection and generational replacement, ramped-half-and-half initialisation with max derivation tree depth

Table 5.6 Results for Persistent Random Constants on the Binomial-3 problem

Case	BestFitness	stdDev	Average Fitness	stdDev
PRC5	9.56	10.9	4596.1	15122.4
PRC10	16.6	20.4	2579.5	6858.4
PRC50	90.09	64.3	13976.9	43121.6
PRC100	104.7442	71.9	2624609	1437136
PRC500	140.6	57.7	1512553876	8276308040
PRC1000	164.3	65.04	2996614	13835151
PRC2000	164.1	50.8	586146.7	305680.9
PRC5000	160.9	62.2	8.716e+16	4.77e+17

Table 5.7 Results for Digit Concatenation on the Binomial-3 problem

Case	BestFitness	stdDev	Average Fitness	stdDev
Concat5	1.14	1.65	727.3	2709.2
Concat10	3.19	5.16	10654.27	34445.8
Concat50	4.5	6.88	52486.8	124798.6
Concat100	5.2	6.97	1418010	6980128
Concat500	6.15	8.22	84063967	409599805
Concat1000	8.2	8.68	901673203	4857220418
Concat2000	5.44	7.25	2815668301	10691830451
Concat5000	3.43	4.75	240268558871	1.311852e+12

limit 17. Thirty independent runs were performed for each constant representation on each range of constants values, which were [0,5], [0,10], [0,50], [0,100],[0,500], [0,1000], [0,2000] and [0,5000].

The results are striking in Digit Concatenation's ability to seemingly ignore the increasing problem difficulty as the range increases. A summary of the results is outlined in Tables 5.7 and 5.6 and Figures 5.13 and 5.14. As evidenced by these results on problem difficulty and the experimental evidence detailed earlier in this chapter, representation can have a profound impact on the performance of Genetic Programming. The cumulative evidence of this chapter tends to support the adoption of Digit Concatenation with expressions as the method of choice for constant creation in GE.

5.7 Conclusions

The ability to evolve new, and adapt existing, constants plays an important role in the effective use of GP paradigms for dynamic environments. When the environment undergoes change, the population of solutions needs to be able to draw upon its diversity, have the ability to evolve to the new circumstances, and explore new areas of the solution space. The objective of this chapter was to identify the best method for constant creation and adaptation

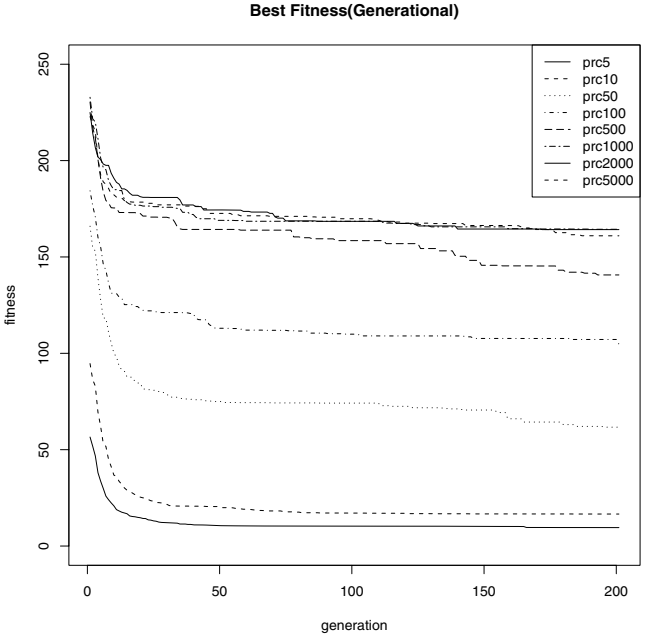

Fig. 5.13 PRC average best fitness over thirty runs on Binomial-3 with increasing range of constants

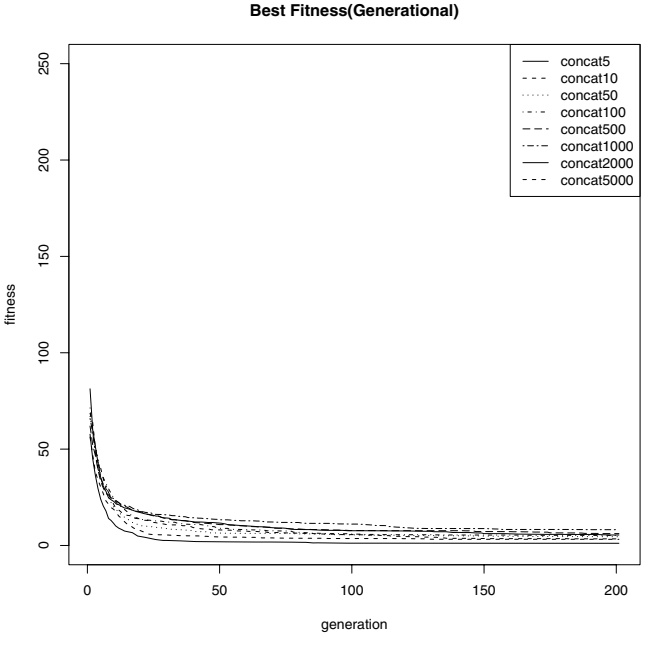

Fig. 5.14 Digit Concatenation average best fitness over thirty runs on Binomial-3 with increasing range of constants

with a focus on performance in dynamic environments, addressing the limitations of standard GP's ERC. To this end three methods in particular were examined on a series of benchmark problems; Traditional, Digit Concatenation and Persistent Random Constants. Throughout these experiments, the Digit Concatenation method was seen to produce the best results with more regularity than all other methods investigated. In Section 5.3, a pure Digit Concatenation grammar exhibited better performance across all of the problems.

Section 5.4 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 most experiments. However, this majority did not translate to PRC producing the best individuals, as Digit Concatenation produced the majority of best fit individuals. Among experiments with the exclusive grammars, the Digit Concatenation method provided superior fitness for the static problems again, with Persistent Random Constants giving better fitnesses for the dynamic problems. PRC thus demonstrated a greater level of evolvability in that it was able to shift to new solutions.

Section 5.5 sought to address the weakness in evolvability of the Digit Concatenation method by providing it with the ability to evolve expressions. This proved to significantly improve fitnesses in the dynamic problem and in contrast to Section 5.4, saw the Digit 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 the Persistent Random Constants method consumes 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 of its range. The result by the final generation gives a slight majority to the Digit Concatenation method. In terms of evolvability, when the target is within the PRC range, method selection is biased towards PRC as it able to more easily evolve the target given the presence of terminals around it. This evolutionary edge is lost when the target transitions outside the PRC range and Digit Concatenation can acquire a majority. The experiments conducted in this section indicate that the ability to create expressions among evolved constants provides Digit Concatenation with a mechanism to shift its search neighbourhood with greater ease and take steps to another area of the solution space.

These experiments have examined different problems involving static constants both complex and simple; dynamic problems with large and small variations in targets; oscillating targets; a co-efficient to a chaotic equation; and symbolic regression. Considering the results, a constant creation grammar that provides the Digit 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. At this point, it is useful to compare the performance of Digit Concatenation combined with an ability to form expressions over these experiments, with the desireable features outlined at the beginning of the chapter:

i. **Static Targets:** Of all the methods tested, Digit Concatenation proved to be the most accurate at evolving static constants. While PRC demonstrated an ability to perform well in Section 5.4, once Digit Concatenation was also able to form expresions it presented superior performance.
ii. **Types of Change:** In these experiments, the mechanisms were tested against Markov type changes in the dynamic problem and Deterministic in the oscillating. While Digit Concatenation presented superior performance against the Traditional approach in Section 5.3, it did not present as good a performance in Section 5.4, where the targets were over a wider range. Its results improved considerably however once combined with the ability to form expressions and outperformed PRC in Section 5.5.
iii. **Adaptable:** Through out these experiments, Digit Concatenation provided the most accurate solutions demonstrating an ability to fine-tune its constants. However, it suffered if the target changed by a large amount. Section 5.5 solved this problem by allowing the method to form expressions between its evolved constants, enabling it to take large evolutionary steps towards a new fit neighbourhood.
iv. **New Constants:** Of the three mechanisms tested Digit Concatenation is the only one that has the ability to create entirely new constants. Both the Traditional and PRC methods begin with a fixed range of constant terminals in the grammar that is specified a-priori. From here, all other values are only arrived at through expressions on these constants. As the experiments suggest this does not provide solutions as accurate as through evolving the individual constants themselves.
v. **Large and Small Constants:** In experiments evolving the co-efficient for the logistic difference equation, the mechanisms were required to evolve a small number accurately. Again Digit Concatenation presented the most accurate results. In evolving the large number, Digit Concatenation also provided the most accurate results. However, without the ability to form expressions Digit Concatenation did not attain reasonable fitnesses as easily as PRC.

While the focus of these experiments was to identify an efficient and effective means of constant creation and adaptation for GE in dynamic environments a number of these experiments also provided insights into the behaviour of GE itself in dynamic environments. As the problems were relatively simple, fundamental features were more easily identified. Emerging from this evolvability was identified as being key. Providing GE with an ability to take large evolutionary steps initially along with an ability to fine tune the phenotypes allows it to produce more accurate results faster.

Considering the results, a constant creation grammar that provides the Digit Concatenation method with the ability to perform expressions is the most beneficial, and efficient method explored and simultaneously addresses the weaknesses of ERC in standard GP. These results are a practical finding for the efficient and effective use of GE in dynamic environments. Thus, future experimentation in this book under the financial domain will use this grammar for generating constants.