Constant Generation for the Financial Domain using Grammatical Evolution

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

ABSTRACT

This study reports the work to date on the analysis of different methodologies for constant creation with the aim of applying the most advantageous method to the dynamic real world problem of a live trading system. The methodologies explored here are Digit Concatenation and Grammatical Ephemeral Random Constants with clear advantages identified for a digit concatenation approach in combination with the ability to form new constants through their recombination within expressions.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

Constant Creation, Digit Concatenation, Genetic Programming, Grammatical Evolution

1. INTRODUCTION

Many applications in Genetic Programming require the generation of constants, the application of GE to financial analysis is no exception. All technical and fundamental indicators require as parameters; constants. Hence the discovery of an efficient means of generating diverse constants is important. But this initial discovery is not the only factor to take into consideration when using these parameters in the financial domain. The markets of the world are dynamic in nature with behaviours varying by different degrees under different circumstances, and so it becomes key that once good parameters are discovered for a particular indicator that these parameters can be tuned within its local neighbourhood or changed radically. Thus a flexible efficient approach is required for constant generation which maintains a high level of diversity.

In this paper we explore Digit Concatenation [1, 2, 3]in Grammatical Evolution as a method for creating constants and analyse its utility under different benchmark problems with the aim of applying this method in a grammar for financial time series analysis. The next section gives a brief background in constant generations techniques in GP. Then Section 3 compares the performance of Concatenation with traditional method for constant creation in GE and a form of Ephemeral Random Constants (ERC) adapted to the GE paradigm. Section 4 develops the Concatenation technique further and section 5 presents our conclusions.

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

2. BACKGROUND

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 [4]. These values are then fixed throughout a run, and new constants can only be created through combinations of these values and other items from the function and terminal set.

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 [5] 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 [6]. 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 [7] 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 [8, 9, 10] 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 [11] 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 ERC as their base each of these methods focused on changing the original random values by small amounts to improve fitness with the exception of [11], which also examined wholesale transformation of constant values and found this feature to be more beneficial than slight changes. GE can borrow from the experience of GP by extending the established methodology and introducing a new form of constant creation which potentially addresses 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. Digit Concatenation works by simply forming constants through the concatenation of the digits 0 through 9, making obsolete the requirement that a large selection of random numbers be generated upon initialisation. An example of the a grammar using Digit Concatenation is provided below.

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

3. PROBLEM DOMAIN & EXPERIMENTAL APPROACH

In this section the Concatenation method is analysed 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 Ephemeral Random Constants technique is explored. Grammatical ERC places a variation on Koza's ERC in that the initial random constants are actually part of the grammar and as such are available to the system through out a run, i.e., they can be evolved out of the population and re-introduced at a later generation. As well as grammatical ERC the Traditional technique for constant generation in GE is added. This method is a watered down version of ERC in that initially, just a handful of basic constants are supplied to the grammar with all other values being derived through expressions. 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. The grammar adopted in these experiments is provided below.

```
<exp> ::= <value>
<value> ::= <trad> | <catR> | <eph>
<op> ::= + | - | / | *
<trad> ::= <trad> <op> <trad> | <tradT>
<tradT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<catR> ::= <cat> . <cat> | <cat>
<cat> ::= <cat> <catT> | <catT>
<catT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<eph> ::= <eph> <op> <eph> | <ephT>
<ephT> ::= ‘‘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 ephemeral 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 `<ephT>`. 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.

A comparison is performed on the utility of three different constant creation methods for evolving constants by performance analysis on two different types of constant creation problems. The problems tackled are, finding a static integer and finding dynamic real constants.

3.0.1 Finding a Static Constant

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

3.0.2 Finding Dynamic Real Constants

This problem of finding dynamic real constants involves a dynamic fitness function that changes its target real constant values at regular intervals (every 10th generation). This experiment sets successive target valuse to be 192.47, 71.84, 173.59 and 192.47. 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.

3.1 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, 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.

3.1.1 Finding a Static Constant

Fig. 1 presents the results for evolving 20021.11501. Here the ERC method is seen to grow to dominate the population with 226 members against 136 and 23 for the Concatenation and Traditional methods. However Concatenation is the method used for 100% of the best individuals yielding an average best performance of 547.217.

3.1.2 Finding Dynamic Real Constants.

In Fig. 2 graphs are presented for the experiments where the set of numbers to be evolved over the course of a run are: 192.47, 71.84, 71.83, 173.59 and 192.47. This time the Ephemeral constants gain a stronger foothold in the population over the course of the run, overtaking Concatenation before generation 20 at the same time presenting good fitness. However at generation 30, where the target changes to 173.59, this fitness deteriorates significantly. This suggests that while the target was within the range of the Ephemeral constants it was able to quickly attain a high fitness and

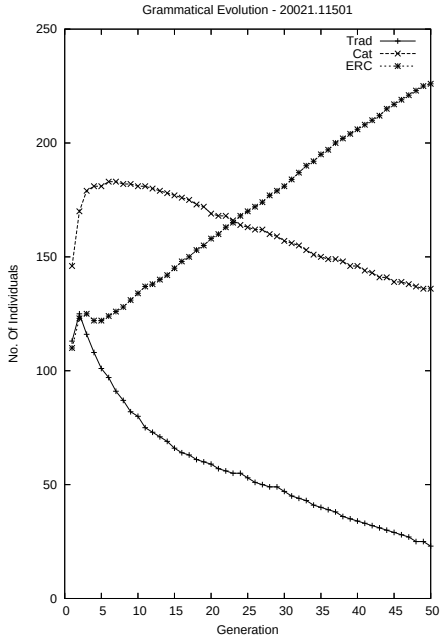

Figure 1: The number of individuals that use each of the three constant generation methods.

a strong position in the population but was unable to successfully evolve from this position once the target left its range.

4. FURTHER ANALYSIS OF DIGIT CONCATENATION

Section 3 demonstrated the superiority of both the Concatenation and Ephemeral 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 grammar similar to the previous section. However the Concatenation method is additionally provided with the ability to form expressions. The grammar used here is provided below.

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

The experiments performed here are the same as in 3 which focus on the creation of a large complex number outside the range of the ERC and on the flexibility of the methods in a dynamic environment. This approach also allows a direct comparison of results.

4.1 Results

For every problem instance the parameters used and number of runs conducted were the same as in section 3.

4.1.1 Finding a Static Constant.

In this case Concatenation and grammatical ERC occupy similar portions of the population up until generation 13. From this point a strong divergence occurs which continues until the final generation. This leads to Concatenation taking the lions share of the population at 318 versus 94 for the ERC method. Of the best performers only 1 of the 30 runs provided a solution using the ERC method with the best Concatenation solution producing an expression which came to within 18.3872 of the solution. By the final generation the best performer on average produced a fitness of 607.968.

4.1.2 Finding Dynamic Real Constants

Fig. 3 displays the results for the dynamic experiments. Again the Ephemeral 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 at generation 30 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 ERC 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.

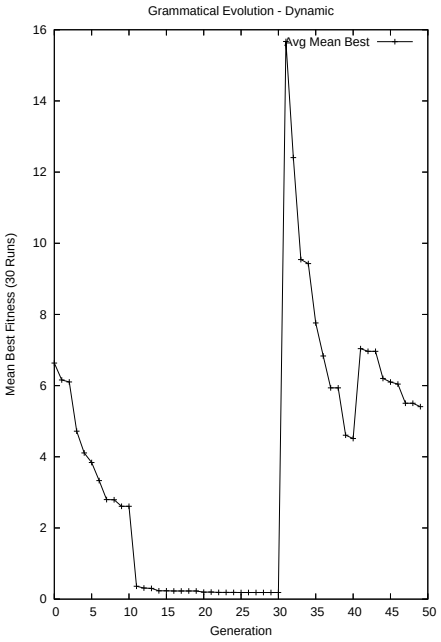

Figure 3: Mean best fitness values (lower values are better) plotted against generations (left), the number of individuals that use each the constant generation methods (right).

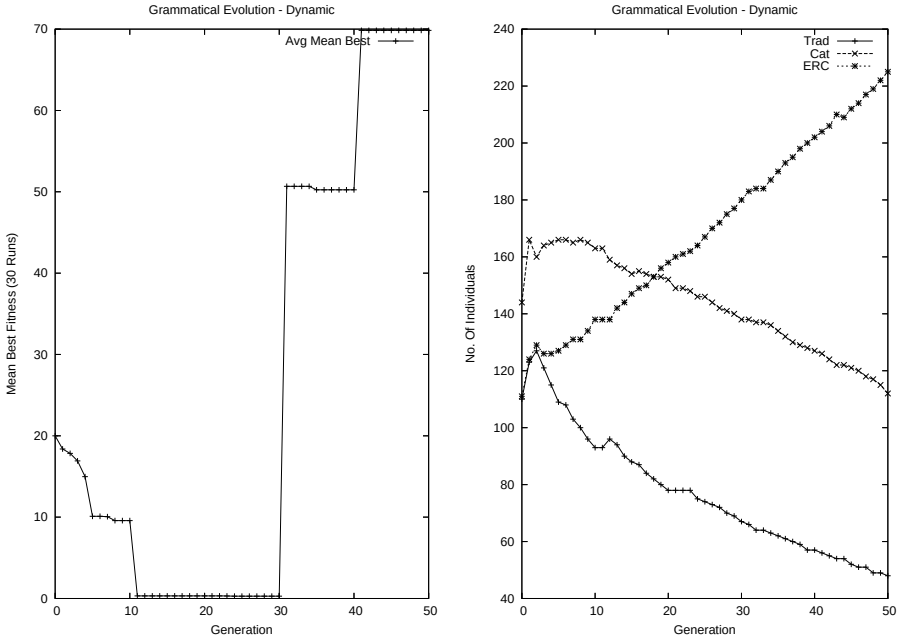

Figure 2: Mean best fitness values (lower values are better) plotted against generations (left) for the dynamic problem, the number of individuals that use each of the three constant generation methods (right).

5. CONCLUSION & FUTURE WORK

This study was undertaken with the aim of exploring Digit Concatenation as a solution to the problem of creating parameters and constants for use in technical expressions for the financial domain. In examining its performance across a series of benchmark problems in comparison with grammatical ERC, it was seen that Concatenation consistently produced more best final solutions and when combined with the ability to form expressions displayed a strong ability to quickly evolve towards a target in the dynamic experiments. This evidence demonstrates a generality of the Concatenation method create new numbers and quickly evolve to new targets are utilities sought across all problems embedded in dynamic environments and not just the financial one.

Initial work has been conducted in using Digit Concatenation with Grammatical Evolution by Grammatical Evolution [12]. Future work in this area will focus on the application of Digit Concatenation in the financial domain along with the use of Grammatical Evolution by Grammatical Evolution.

6. REFERENCES

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

[2] Dempsey, I., O'Neill, M., 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 AICS Conference, pp.165-170,Springer-Verlag.

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

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

[5] Spencer, G. (1994). Automatic Generation of Programs for Crawling and Walking. In Kenneth E. Kinnear, Jr. (Ed), Advances in Genetic Programming, Chapter 15, pp. 335-353, MIT Press.

[6] Angeline, Peter J. (1996). Two Self-Adaptive Crossover Operators for Genetic Programming. Advances in Genetic Programming 2, Chpt 5, pp.89-110, MIT Press.

[7] Evett, Matthew and Fernandez, Thomas. (1998). Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming, Genetic Programming 1998: Proceedings of 3rd Annual Conference, U. of Wisconsin, Madison, Wisconsin, USA, pp.66-71, Morgan Kaufmann.

[8] Iba, H and Nikolaev, N. *Genetic programming polynomial models of financial data series*, Proceedings of CEC 2000, IEEE Press, pp. 1459–1466.

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

[10] Keijzer, M. *Improving Symbolic Regression with Interval Arithmetic and Linear Scaling*. In LNCS 2610 Proceedings of the 6th EuroGP 2003, pp. 70–82.

[11] 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.

[12] Dempsey, I., O'Neill, M., Brabazon, A. meta-Grammar Constant Creation with Grammatical Evolution by Grammatical Evolution. In the proceedings of GECCO 2005.