Grammatical Evolution: A Steady State approach.

Conor Ryan
Dept. of Computer Science And Information Systems
University of Limerick, Ireland
Conor.Ryan@ul.ie

Michael O' Neill
Dept. of Computer Science And Information Systems
University of Limerick, Ireland
Michael.ONeill@ul.ie

ABSTRACT

We describe a Genetic Algorithm that can evolve complete programs. Using a variable length linear genome to govern the mapping of a Backus Naur Form grammar definition to a program, expressions and programs of arbitrary complexity may be evolved. Our system, *Grammatical Evolution*, has been applied to problems such as a Symbolic Regression problem, and finding Trigonometric Identities.

In this paper we describe how we applied GE to a Symbolic Integration problem, that being finding a function which is an integral of $Cos(X) + 2X + 1$. We also show how a Steady State selection mechanism was found to improve the performance of *Grammatical Evolution*.

1 Introduction

Evolutionary Algorithms have been used with much success for the automatic generation of programs. In particular, Koza's [Koza 92] Genetic Programming has enjoyed considerable popularity and widespread use. While Koza originally employed Lisp as his target language many experimenters generate a home grown language peculiar to their particular problem. Grammatical Evolution (GE) can be used to generate programs in any language, using a genetic algorithm to control what production rules are used in a Backus Naur Form (BNF) grammar definition.

Whigham has used grammars with respect to Genetic Programming [Whigham 95] to define the structure of the individuals in the initial population and to direct operators such as crossover to ensure syntactically correct programs are created. Whigham also used the grammar as a method of biasing the initial population towards the target solution [Whigham 96].

Whereas Whigham used a grammar to define the initial population and lets Genetic Programming operate on the resulting parse trees, GE simply encodes a binary string which the Genetic Algorithm operates on. The grammar is then used to map the individual bit strings onto a program when evaluating fitness.

GE has proved successful when applied to a symbolic regression problem [Ryan 98a], and finding trigonometric identities [Ryan 98b], here we apply GE to a symbolic integration problem taken from the literature [Koza 92]. This involves finding a function which is an integral of $Cos(X) + 2X + 1$.

In each of these cases we take a subset of C as our target language which is described in Backus Naur Form definition. A Steady State selection mechanism [Syswerda 89] has been employed and was found to reduce the number of generations required to achieve a correct solution. Using this selection mechanism we reapplied our system to the two problems previously tackled and again found an improvement in performance for both of these problems.

2 Grammatical Evolution

Grammatical Evolution codes a set of pseudo random numbers on a chromosome which consists of a variable number of 8 bit binary genes. These numbers are used to select an appropriate rule from a BNF grammar definition, an example of which is given below. A BNF grammar consists of the tuple $\{N, T, P, S\}$, where N is the set of non-terminals, T, the set of terminals, P, a set of production rules that map the elements of N to T, and S is a start symbol which is a member of N. The non terminals of the grammar are then mapped onto the terminals of the grammar by continuously applying the rules dictated by the gene values. On completion of the mapping process the final code produced consists of these terminals.

$$N = \{expr, op, pre_op\}$$

$$T = \{Sin, Cos, Tan, Log, +, -, /, *, X, ()\}$$

$$S = < expr >$$

And P can be represented as:

```
(1) <expr> ::= <expr> <op> <expr>      (A)
             | ( <expr> <op> <expr> )  (B)
             | <pre-op> ( <expr> )     (C)
             | <var>                   (D)

(2) <op> ::= + (A)
```

```
            | - (B)
            | / (C)
            | * (D)

(3) <pre-op> ::= Sin (A)
               | Cos (B)
               | Tan (C)
               | Log (D)

(4) <var> ::= X
```

Consider rule #1 from the above example:

```
(1) <expr> ::= <expr> <op> <expr>
             | ( <expr> <op> <expr> )
             | <pre-op> ( <expr> )
             | <var>
```

In this case, the non-terminal can produce one of four different results, to decide which one to use our system takes the next available random number from the chromosome and, in this case gets the modulus four of the number to decide which production to take. Each time a decision has to be made, another pseudo random number is read from the chromosome, and in this way, the system traverses the chromosome.

In a manner similar to natural biology, [Elseth 95] the genes in GE are ultimately operated on to produce a phenotype, or in our sense, a program. The phenotype is produced as a result of a mapping process which we liken to the process of gene transcription, followed by translation to a protein, see Figure 1.

Gene transcription is the process of converting a strand of DNA into a strand of RNA. The RNA then carries the encoded instructions of the genes to the machinery of the cell which can synthesize proteins. Translation is the mapping of the RNA onto amino acids. Put simply, the RNA is comprised of four different types of bases, groups of three bases, known as a codon, specify a particular amino acid. The amino acids are thus joined together in the order specified by the RNA to produce a protein. These proteins can act either independently or in conjunction with other proteins to produce a phenotype. To complete the analogy with GE then, the transcription process would be equivalent to the conversion of the binary string data structure into a string of integers which is brought from the Genetic Algorithm into the machinery of GE. The integers are then used in the translation process, that is the selection of rules in the BNF definition that result in the mapping of non-terminals to terminals. The production rules, equivalent to amino acids, combine to produce terminals, or proteins. It is the proteins that make up the final program.

In GE it is possible for individuals to run out of genes during the mapping process, and in this case there are two alternatives. The first is to declare the individual invalid and punish them with a suitably harsh fitness value; the alternative is to wrap the individual, and reuse the genes. This is quite an unusual approach in EAs, as it is entirely possible for certain genes to be used two or more times. What is crucial, however, is that each time a particular individual is mapped from its genotype to its phenotype, the same output is generated. This is because the same choices are made each time.

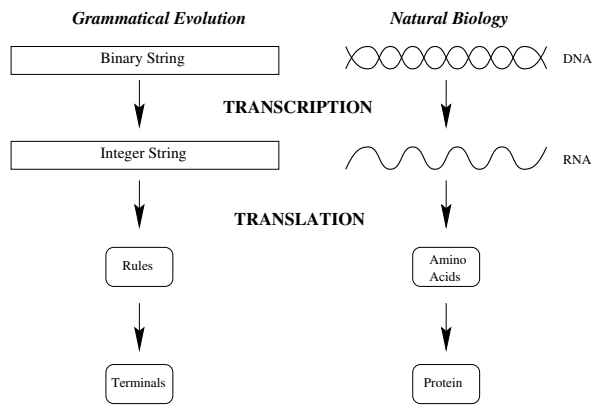

Figure 1 A comparison between Grammatical Evolution and natural biology.

To complete the BNF definition for a C function, we need to include the following rules with the earlier definition:

```
<func> ::= <header>

<header> ::= float symb(float X) { <body> }

<body> ::= <declarations><code><return>

<declarations> ::= float a;

<code> ::= a = <expr>;

<return> ::= return (a);
```

Notice that this function is limited to a single line of code, returns a float, and is passed a single float value X as a parameter. However, this is because of the nature of the problems tackled here, the system could easily generate functions which use several lines of code and a different parameter set by simply modifying the BNF grammar definition.

3 The Problem Spaces

We have previously applied our system to two problems which were approached by Koza using his Genetic Programming paradigm [Koza 92]. What follows is a brief description of both of these problems, and a third Symbolic Integration problem, each of which is summarised in a *tableau* style adopted from Koza. It should be noted here that we use operators and terminals with their meaning with respect to grammars [Whigham 96], and are distinct from

Koza's terminals and operators. With the exception of the Trigonometric Identities problem, which used a slightly modified BNF grammar definition the reasons for which are given below, each of these problems used the same production rules. The production rules for <expr> are as given earlier. As this and subsequent rules are the only ones that require a choice, they are the ones that will be evolved in each of the three problems described.

3.1 Symbolic Regression

In the first problem we apply our system to a symbolic regression problem summarised in Table 1 [Ryan 98a]. The system is given a set of input and output pairs, and must determine the function that maps one onto the other. The particular function examined is $X^4 + X^3 + X^2 + X$ with the input values in the range $[-1.. + 1]$. As the target function has no constant values, only the variable X, it was decided to use only one terminal operand, i.e. X. So as not to unduly bias the system towards the correct solution, the terminal operators set consisted of more than the binary addition and multiplication operators, which alone would be sufficient to reach the target function. To determine the fitness of an individual program 20 test points were applied in the range -1 to 1, and the fitness was taken as the sum of the error.

Objective :	Find a function of one independent variable and one dependent variable,in symbolic form that fits a given sample of 20 (x_i, y_i) data points, where the target functions is the quartic polynomial $X^4 + X^3 + X^2 + X$
Terminal Operands:	X (the independent variable)
Terminal Operators	The binary operators $+, *,$ $-$ and, the unary operators Sin, Cos, Exp and Log
Fitness cases	The given sample of 20 data points in the interval $[-1, +1]$
Raw Fitness	The sum, taken over the 20 fitness cases, of the error
Standardised Fitness	Same as raw fitness
Hits	The number of fitness cases for which the error is less than 0.01
Wrapper	Standard productions to generate C functions
Parameters	M = 500, G = 51

Table 1 Grammatical Evolution Tableau for Symbolic Regression

3.2 Finding Trigonometric Identities

The second problem involves applying our system to a trigonometric identity problem summarised in Table 2 [Ryan 98b]. The particular function examined was $Cos\ 2x$, and the desired trigonometric identity was $1 - 2Sin^2\ x$. The system was given a set of input and output pairs, and must determine the function that maps one onto the other, with the input values in the range $[0, 2\pi]$. Other identities exist for $Cos\ 2x$, such as $2Cos^2 x - 1$. If we were to include Cos as one of the terminals in our BNF grammar definition, GE would naturally produce simple Cos identities for $Cos\ 2x$, which was verified in early experiments which included Cos. These runs consistently found the target expression $Cos\ 2x$ and, therefore, we decided to exclude Cos from the terminal operator set. Koza included the constant 1.0 in his terminal operator set, although Genetic Programming has shown an ability to generate the constant 1.0, as has GE with expressions such as x/x. However, we included 1.0 for reasons of consistency.

Objective :	Find a new mathematical expression, in symbolic form that equals a given mathematical expression, for all values of its independent variables.
Terminal Operands:	X , the constant 1.0
Terminal Operators	The binary operators $+, *,$ $/, -$ and the unary operator Sin
Fitness cases	The given sample of 20 data points in the interval $[0, 2\pi]$
Raw Fitness	The sum, taken over the 20 fitness cases, of the error
Standardised Fitness	Same as raw fitness
Hits	The number of fitness cases for which the error is less than 0.01
Wrapper	Standard productions to generate C functions
Parameters	M = 500, G = 51

Table 2 Grammatical Evolution Tableau for finding Trigonometric Identities

3.3 Symbolic Integration

Thirdly, we have applied our system to a symbolic integration problem as summarised in Table 3. Symbolic Integration involves finding a function which is the integral of the given curve, in this case $Cos(X) + 2X + 1$. The system is given a set of input and output pairs, and must determine the function that maps one onto the other. The particular function examined was $Cos(X) + 2X + 1$ with the input values in the range $[0..2\pi]$ and, the target integral curve is $Sin(X) + X + X^2$. Similarly to the Symbolic Regression problem described earlier, this problem used the same BNF grammar, including the terminal operators $+, *, /, -$, Sin, Cos, Exp and Log. The addition, multiplication and Sin operators would have been suffcient to solve this problem but, again to avoid biasing the system other terminal operators were included.

Objective :	Find a function, in symbolic form, that is the integral of a curve presented either as a mathematical expression or as a given finite sample of points (x_i, y_i)
Terminal Operands:	X (the independent variable)
Terminal Operators	The binary operators $+, *,$ $/, -$ and the unary operators Sin, Cos, Exp and Log
Fitness cases	The given sample of 50 data points in the interval $[0, 2\pi]$
Raw Fitness	The sum, taken over the 50 fitness cases, of the absolute value of the difference between the individual genetically produced function $f_j(x_i)$ at the domain point x_i and the value of the numerical integral $\mathbf{I}(x_i)$
Standardised Fitness	Same as raw fitness
Hits	The number of fitness cases for which the error is less than 0.01
Wrapper	Standard productions to generate C functions
Parameters	M = 500, G = 51

Table 3 Grammatical Evolution Tableau for Symbolic Integration

4 Results

GE was successful in finding correct solutions to all of the problems described here. Results for the first two problems, that of Symbolic Regression and finding Trigonometric Identities, have been reported previously in [Ryan 98a] [Ryan 98b]. However, whilst carrying out the experiments for the Symbolic Integration problem it was decided to try a Steady State selection mechanism. The reasoning behind this was that GE produced a lot of invalid individuals, whom with a generational selection mechanism are passed onto the following generation, whereas with a Steady State mechanism these individuals would be less likely to survive into the next generation. Steady State selection was found to enhance the performance of the system and so, it was decided to apply this new selection mechanism to both of the problems previously tackled. The Steady State selection was also found to improve on the results reported previously for these problems and these results are reported here.

4.1 Symbolic Regression

GE successfully found the $X + X^2 + X^3 + X^4$ target function in the original experiments and again with the Steady State selection. Using the Steady State selection saw an improvement over previous results and a cumulative frequency measure can be seen in Figure 2. In the original experiments only 2% of the runs would find the target expression within the designated 50 generations. Now, however 95% of the runs successfully find the solution within this time frame.

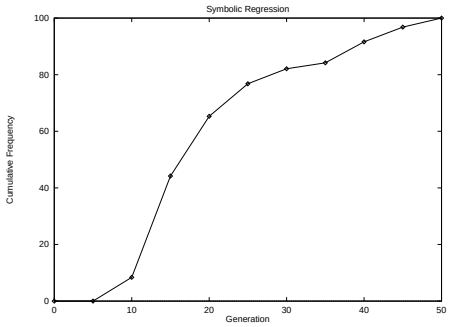

Figure 2 Symbolic Regression cumulative frequency measure.

4.2 Trigonometric Identities

In the previous experiments the $1 - 2Sin^2(X)$ identity was found as well as a close approximation to the $Sin(\pi/2 - 2x)$, which was achieved by successive application of the Sin function to 1.0. This results in a constantly decreasing value that closely approximated $\pi/2$, which shows that GE is effectively evolving a numerical constant. On using the Steady State selection an improvement in performance was again achieved, increasing from 4% of runs being successful within 50 generations to 87% with the Steady State selection, see Figure 3.

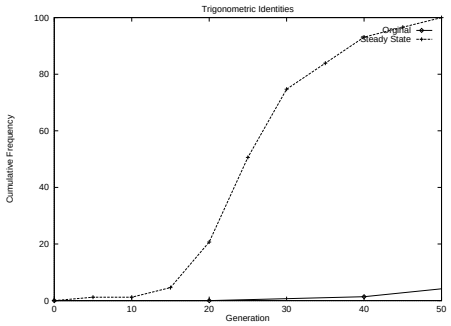

Figure 3 Finding Trigonometric Identities cumulative frequency measure.

4.3 Symbolic Integration

Originally experiments were carried out on this problem without the Steady State selection, and GE successfully found the target integral function $Sin(X) + X + X^2$. A comparison of cumulative frequency measures for both sets of experiments can be seen in Figure 4. With Steady State selection 87% of the runs successfully found the solution within the designated 50 generations, whereas before only 22% achieved this.

5 Discussion

As can be seen from the results in Table 4, Steady State selection was responsible for the improved performance in all three problems dealt with here. The use of a Steady State selection mechanism, especially with larger populations [De Jong 92], has been regarded as a way to improve performance of a Genetic Algorithm. In our case the fact that so

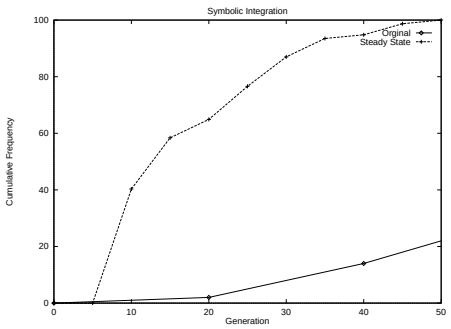

Figure 4 Symbolic Integration cumulative frequency measure.

many invalid individuals were being produced when creating the next generation, caused the system to perform poorly, because these invalid individuals were being passed onto the next generation as a result of the generational selection mechanism. However with Steady State selection this problem is overcome by only replacing a few individuals at every new generation, thus preventing these invalid individuals from entering the population.

Another possible reason for the improvement in performance could also be attributed to the fact that using Steady State selection means those individuals with a better fitness value are more likely to be preserved from one generation to the next, however poorer individuals are more prone to deletion from the population. It is likely that individuals (or parts of) that represent useful building blocks are more likely to be preserved when using a Steady State as opposed to a Generational selection mechanism. In effect the population as a whole is working toward the solution.

Problem Domain	**% Successful**	
	Steady State	Generational
Symbolic Regression	95	2
Trigonometric Identities	87	4
Symbolic Integration	87	22

Table 4 Comparison of results for the 3 problems, with and without Steady State selection.

6 Conclusions

We have described a system, Grammatical Evolution (GE) that can map a binary genotype onto a phenotype which is a high level program. GE has proved successful for three different types of problems, namely, Symbolic Regression [Ryan 98a], finding Trigonometric Identities [Ryan 98b], and a Symbolic Integration problem reported here.

We have demonstrated that by using a Steady State selection mechanism the performance of GE can be improved significantly for the problems addressed here, thus bringing GE on a par with Genetic Programming in terms of the number of generations it takes to reach a correct solution. A more detailed comparison between the two systems will be carried out at a later date. Other problem domains are being tackled with GE and it is hoped to show that it can perform at least as well if not better than GP across a broad spectrum of problems.

In particular, it is intended to apply GE to a problem that requires multi line functions. Because our mapping technique employs a BNF definition, the system is language independent, and, can easily generate functions which use several lines of code, specifically, if the rule for `<code>` in the earlier definition was modified to read:

```
<code> ::= <line>;  | <line>; <code>

<line> ::= <var> = <expr>
```

then the system could generate functions of arbitrary length.

References

[De Jong 92] De Jong Kenneth A., Sarma Jayshree. 1992. Generation Gaps Revisited. *Foundations of Genetic Algorithms 2*, pages 19-28. Morgan Kauffmann.

[Elseth 95] Elseth Gerald D., Baumgardner Kandy D. 1995. Principles of Modern Genetics. *West Publishing Company*

[Koza 92] Koza, J. 1992. *Genetic Programming*. MIT Press.

[Ryan 98a] 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*, pages 83-95. Springer-Verlag.

[Ryan 98b] Ryan C., O'Neill M., Collins J.J. 1998. Grammatical Evolution: Solving Trigonometric Identities. In *Proceedings of Mendel '98*, pages 111-119.

[Syswerda 89] Syswerda, Gilbert. 1989. Uniform crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2-9. Morgan Kauffmann.

[Whigham 95] Whigham, P. 1995. Grammatically-based Genetic Programming. In *Proceeding of the Workshop on Genetic Programming : From Theory to Real-World Applications*, pages 33-41. Morgan Kaufmann Pub.

[Whigham 96] Whigham, P. 1996. Search Bias, Language Bias and Genetic Programming. In *Genetic Programming 1996*, pages 230-237. MIT Press.