Grammatical Evolution : Solving Trigonometric Identities

Conor Ryan, Michael O' Neill & JJ Collins
Dept. Of Computer Science And Information Systems
University of Limerick
Ireland
{Conor.Ryan|Michael.ONeill|J.J.Collins}@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 already been applied to a symbolic regression problem. Here we apply our system to find Trigonometric Identities for Cos 2x.

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. Koza originally employed Lisp as his target language, however, 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 Backus Naur Form definitions we allow a genetic algorithm to control what production rules are used. GE has proved successful [Ryan 98] when applied to a symbolic regression problem from the literature [Koza 92]. Here we apply GE to another problem, that of, finding trigonometric identities. The goal is to discover a new mathematical expression that is equivalent to the target expression. In this case we successfully attempt to find the $1 - 2Sin^2x$ symbolic identity for $Cos\ 2x$.

1.1 Backus Naur Form

Backus Naur Form (BNF) is a notation for expressing the grammar of a language in the form of production rules. BNF grammars consist of `terminals`, which are items that can appear in the language, i.e $+, -$ etc. and `non-terminals`, which can be expanded into one or more terminals and non-terminals. A grammar can be represented by 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 maps the elements of N to T, and S is a start symbol which is a member of N. For example, below is the BNF used for this problem, where

$$N = \{expr, op, pre\_op\}$$

$$T = \{Sin, Cos, Tan, Log, +, -, /, *, X, 1.0, (,)\}$$

$$S =< expr >$$

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

(4) <var> ::= X   (A)
      | 1.0 (B)
```

Unlike a Koza-style approach, there is no distinction made at this stage between what he describes as functions (operators in this sense) and terminals (variables in this example), however, this distinction is more of an implementation detail than a design issue.

Whigham [Whigham 96] also noted the possible confusion with this terminology and used the terms **GPFunctions** and **GPTerminals** for clarity. We will also adopt this approach, and use the term **terminals** with its usual meaning in grammars.

For the above BNF, Table 1 summarizes the production rules and the number of choices associated with each.

Rule no.	Choices
1	4
2	4
3	1
4	2

Table 1: The number of choices available from each production rule.

2 Grammatical Evolution

Our system codes a set of pseudo random numbers, which are used to decide which choice to take when a non terminal has one or more outcomes. A chromosome consists of a variable number of binary genes, each of which encodes an 8 bit number.

Consider rule #1 from the previous 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, our system takes the next available random number from the chromosome 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 fashion similar to natural biology the genes in GE are expressed as a protein. These proteins can act either independently or in conjunction with other proteins [Elseth 95], the physical results depending on the other proteins that are present immediately before and after a genes expression. It is possible for individuals to run out of genes, 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. Each time the gene is expressed it will always generate the same protein, but depending on the other proteins present, may have a different effect. The latter is the more biologically plausible approach, and often occurs in nature. What is crucial, however, is that each

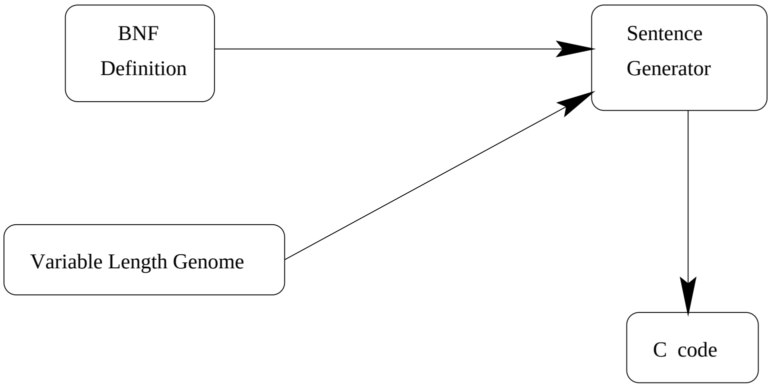

Figure 1: The Grammatical Evolution System

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.

2.1 Example Individual

Consider an individual, see Figure 2, made up of the following genes (expressed in decimal for clarity). Figure 2

These numbers will be used to look up the table in Section 1 which describes the BNF grammar for this particular problem. 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);
```

The first few rules don't involve any choice, so all individuals are of the form:

```
float symb(float x)
{
  a = <expr>;
  return(a);
}
```

220	240	220	203	101	53	202	203	102	55	220	202	241	130	37	202	203	140	39	202	203	102

Figure 2: An example indivdual.

Concentrating on the `<expr>` part, we can see that there are four productions to choose from. To make this choice, we read the first gene from the chromosome, and use it to generate a protein in the form of a number. This number will then be used to decide which production rule to use, thus we have 220 MOD $4 = 0$ which means we must take the first production, namely, 1A. We now have the following

```
<expr> <op> <expr>
```

Notice that if this individual is subsequently wrapped, the first gene will still produce the protein 220. However, depending on previous proteins, we may well be examining the production of another rule, possibly with a different amount of choices. In this way, although we have the same protein it results in a different physical trait.

Continuing with the first <expr>, a similar choice must be made, again using 240 MOD $4 = 0$, that is 1A. We now have the following

```
<expr> <op> <expr> <op> <expr>
```

Similarly again we have the same choice for the first <expr>, the result being

```
<expr> <op> <expr> <op> <expr> <op> <expr> <op> <expr>
```

Now the first <expr> will be determined by the gene value 203 which gives us rule 1D which is <var>. The next gene then determines what value <var> shall take, 101 MOD $2 = 1$ i.e. rule 4B, which turns out to be 1.0. We now have the following

```
1.0 <op> <expr> <op> <expr> <op> <expr> <op> <expr>
```

The next gene will determine what <op> will become, so we have 53 MOD $4 = 1$, which gives a $-$. The next <expr> has then to be expanded using the gene value 202, that is 202 MOD $4 = 2$. So we now have

```
1.0 - <pre-op><expr> <op> <expr> <op> <expr> <op> <expr>
```

There can only be one outcome for a <pre-op> that being Sin, therefore, no decision has to be made and so no gene is read.
The next <expr> is then expanded by the value 203 MOD $4 = 3$ which is rule 1D, or <var> . It's value is then determined by 102 MOD $2 = 0$, rule 4A, and the resulting expression is

```
1.0 - Sin(x) <op> <expr> <op> <expr> <op> <expr>
```

The mapping continues until eventually, we are left with the following expression:

```
1.0 - Sin(x)*Sin(x) - Sin(x)*Sin(x)
```

Notice how all of the genes were required in this case, had there been any extra genes they would have been simply ignored.

3 The Problem Space

As proof of concept, we have applied our system to a trigonometric identity problem described by Koza [Koza 92]. 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 $[0, 2\pi]$. Table 2 contains a tableau which summarizes Koza's experiments.

Other identities exist for $Cos\ 2x$, such as $2Cos^2x - 1$. If we were to include Cos as one of the <pre-op> rules, 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 consistency reasons.

We adopt a similar style to Koza of summarizing information, using a modified version of his *tableau* in Table 3. Notice how our *terminal operands* and *terminal operators* are analogous to *GPTerminals* and *GPfunctions* respectively.

Objective :	Find a new mathematical expression, in symbolic form that equals a given mathematical expression, for all values of its independent variables.
GPTerminal Set:	X , the constant 1.0.
GPFunction Set	$+, -, *, \%, \sin$
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	None
Parameters	$M = 500, G = 51$

Table 2: A Koza-style tableau

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 3: Grammatical Evolution Tableau

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.

4 Results

GE successfully found the $1 - 2Sin^2\ x$ identity for $Cos\ 2x$ on the majority of runs. An example of how one solution was arrived at will now be given. Early on in the run very simple expressions existed, such as 1.0 which had a fitness 0.08 (The best fitness an individual may have is 1.0), $Sin\ x$ whose fitness was 0.07, $Sin\ 1.0$ whose fitness was 0.10, and $(Sin\ x) * (Sin\ x)$ whose fitness was 0.06, while not particularly fit these individuals are the essential building blocks for the ultimate solution.

After the first 20 generations the best individuals were

$Sin(1.0) - Sin(Sin(Sin\ 1.0)) * x$ which had a fitness 0.08,

$1.0 - Sin(Sin\ x) * x$ which had a fitness 0.13,

$1.0 - Sin(Sin\ x * Sin\ x)$ which had a fitness 0.15.

Shortly after this, two individuals appeared that gradually spread amongst the population. These were

$1.0 - (Sin\ x) * (Sin\ x)$,

$Sin\ 1.0 - (Sin\ x) * (Sin\ x)$,

whose fitness were 0.17 and 0.20 respectively. Again, while not particularly fit themselves, they are important steps on the path to the solution. All that was needed now was another $-(Sin\ x) * (Sin\ x)$ to be added onto the first of these expressions. Generation 39 saw this come to pass with a slight variation, the individual being

$1.0 - (Sin\ x) * (Sin\ x) + 1.0 - 1.0 - (Sin\ x) * (Sin\ x)$,

which had the fitness of 1.00.

Subsequent runs produced other notable individuals. One individual seen in Figure 3,

$1.0 - Sin\ x * (x + Sin(Sin(Sin\ x)))$,

approximated the function $Cos\ 2x$ very closely in the range -1 to $+1$ but outside this range it deviated from $Cos\ 2x$. Others resembled the target functions overall appearance almost exactly, but were phase shifted away by a very small number of radians, one of these individuals seen in Figure 3 was

$Sin(x + Sin(Sin(x/x)) + x + Sin(Sin(x/x)))$.

Another interesting individual

$Sin(x + x + 1.0/Sin(Sin(Sin(Sin(1.0)))))$,

seen in Figure 4, was almost identical to the target function. Interestingly, in this individual we can see the successive application of the Sin function to the constant 1.0. Koza also observed a similar phenomenon with Genetic Programming [Koza 92]. Effectively GE is evolving a numerical constant. Successive application of the Sin function to 1.0 results in a constantly decreasing value. After four

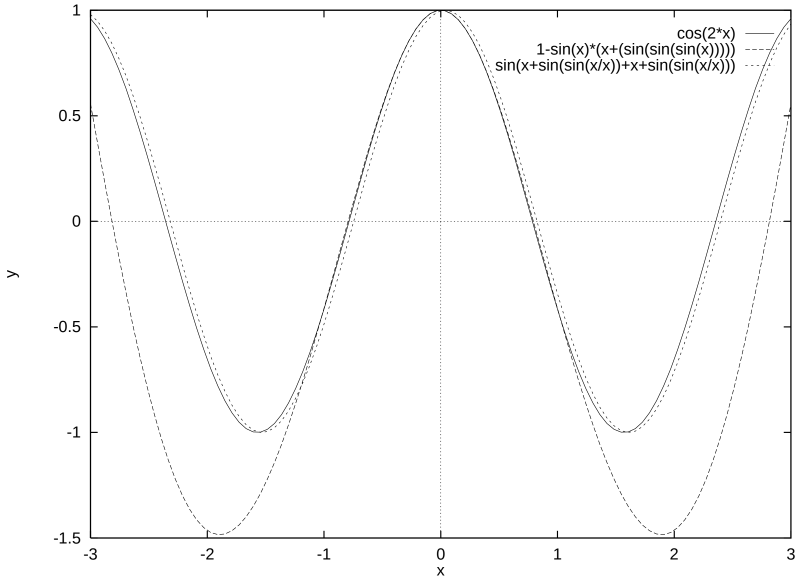

Figure 3: Some interesting individuals.

applications of Sin to 1.0 the result is 0.628, and then the subsequent reciprocal produces 1.59. This is close to $\pi/2$ which is approximately 1.57. Simplified the expression can now be shown as follows: $Sin(2x + 1.59)$, which is very close to another well known indentity of $Cos\ 2x$, that being $Sin(\pi/2 - 2x)$.

Figure 5 shows a cumulative frequency measure of when the solution was arrived at. When compared with Genetic Programming, GE is somewhat slower at generating the solution for this particular problem. We are not unduly concerned at GP being faster, as we don't expect GE to be viewed as a replacement to GP. Rather, that it complements GP, and is intended to be used for the generation of functions of arbitray complexity. We are more concerned with proving that this system can generate functions of this type. It is envisaged that when it comes to problems requiring multi-line function solutions, GE will come into it's own.

5 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 two different types of problems, namely, symbolic regression [Ryan 98] and finding trigonometric identities. We have shown how GE created trigonometric identities for $Cos\ 2x$ such as $1 - 2Sin^2\ x$, and a close approximation to $Sin(\pi/2 - 2x)$.

Genetic Programming was also applied to these problems [Koza 92] with comparable success. GE, like GP, has the ability to evolve its own constants, as was seen during these experiments with the successive application of Sin to the constant 1.0, and through the generation of 1.0 with expressions such as x/x. Whilst GE evolves the required solutions in both problem sets in the designated number of generations, it can take longer to evolve the solutions when compared with Genetic Programming. The major strength of GE with respect to GP is its ability to generate multi-line functions in any language.

The next step will be to apply GE to a problem that requires multi line functions. Because our mapping technique employs a BNF definition, the system is language independant, and, theoretically can generate arbitrarily complex functions.

GE can easily generate functions which use several lines of code specifically, if the rule for `<code>` in the earlier definition was modified to read:

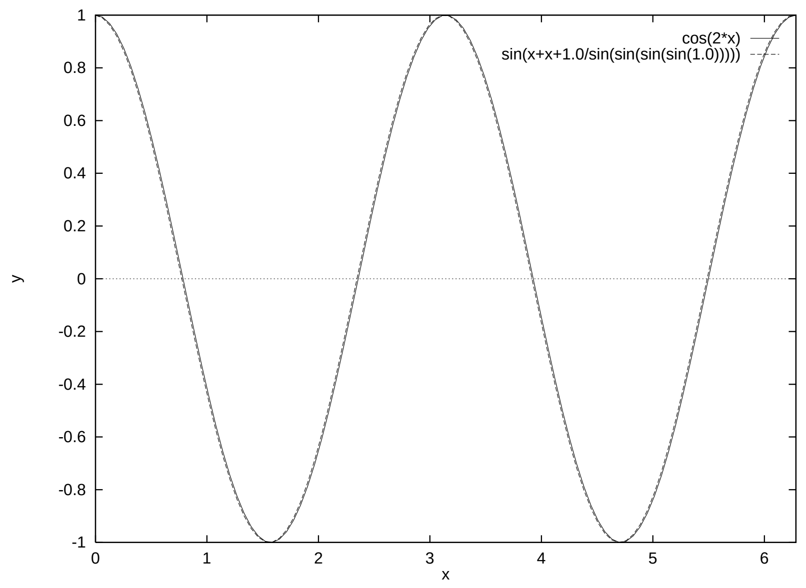

Figure 4: Generating numerical constants to give $Sin(\pi/2 - 2x)$.

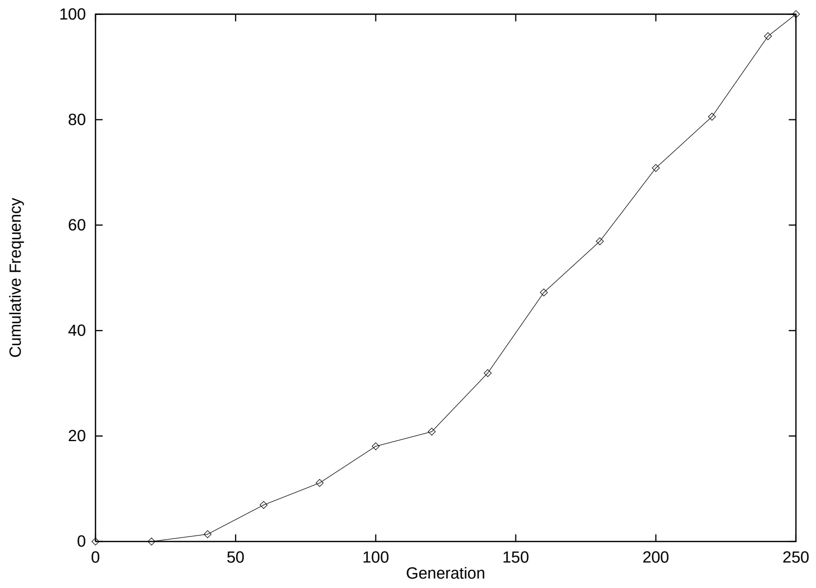

Figure 5: Cumulative frequency measure of when the solution was found.

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

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

then the system could generate functions of arbitrary length. We therefore, do not see GE as a replacement for GP, but to be used in the generation of arbitrarily complex functions.

References

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

[Goldberg 89] Goldberg D E, Korb B, Deb K. Messy genetic algorithms: motivation, analysis, and first results.
Complex Syst. 3

Vienna University of Economics.

reproduction and genotype-phenotype mapping from linear binary genomes into linear LALR phenotypes. In *Genetic Programming 1996*, pages 116-122. MIT Press.

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

caching algorithms in C by GP. In *Genetic Programming 1997*, pages 262-267. MIT Press.

Provable Parallel Programs. In *Genetic Programming 1996*, pages 406-409. MIT Press.

scheme. In *Proceedings of Mendel '97*, pages 140-147. PC-DIR, Brno, Czech Republic.

[Ryan 98] Ryan C., Collins J.J., O'Neill M. Grammatical Evolution: Evoloving Programs for an Arbitrary Language
In print

[Schutz 97] Schutz, M. 1997. Gene Duplication and Deletion, in the Handbook of Evolutionary Computation. (1997) Section C3.4.3

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