

Grammatically-based Genetic Programming

- Koza's Tree-structured framework for Genetic Programming (GP) assumed that a 'syntactic closure' property was enforced (Koza, 1992).
 - Any node could be crossed over with any other node.
 - Might well represent constraints on the representation assumed.
 - Strongly typed GP went a step further to enforce type constraints on classes of instructions; hence instructions have specific variable types associated with arguments and return values (Montana, 1995).
- *Context Free Grammars* potentially represent a different mechanism for enforcing type and syntax.
- In the following we assume the approach suggested by (Whigham, 1995)

Context Free Grammar in Backus-Naur Form

- A Context Free Grammar (CFG) defines a symbol set in which the symbol syntax is independent of the neighboring symbols.
- The symbols themselves take one of two forms.
 - A terminal set, T , which is the set of symbols appearing in legal sentences of the CFG.
 - A Non-terminal set, N , which expand into terminals or non-terminals under the application of a set of production rules, P .
- A Backus-Naur Form (BNF) expresses the CFG as a tuple, $\{N, T, P, S\}$ in which S is the predefined start symbol, selected from the set of non-terminals.
- The specific choice of start symbol (S), terminal (T), non-terminal (N) and production rules (P) are all design decisions made in the light of the application/ task at hand.
- Consider the classic 6-bit multiplexer benchmark (Koza, 1992):
 - Input attributes: $a_0, a_1, d_0, d_1, d_2, d_3 \in \{0, 1\}$
 - Output: $f \in \{0, 1\}$
 - Instructions: AND (x, y), OR (x, y), NOT (x), IF (x, y, z)
 - Where $\{x, y, z\}$ represent arguments to the instruction.

- IF (x, y, z) is interpreted as ‘if the argument 0 (x) is false, then return the result of the third argument (z), otherwise return argument 2 (y)
 - All 2⁶ exemplars appear in the training partition (no independent test),
 - Task is more about can the search process consistently rebuild the logic for the control mechanism with respect to each of the two control inputs (a₀, a₁)
- A CFG capable of supporting the identification of such a control mechanism might have the form,

$S =$	N
$N =$	$\{B\}$
$T =$	$a_0 \ a_1 \ d_0 \ d_1 \ d_2 \ d_3$
$P =$	$B \rightarrow \{AND(B, B), OR(B, B), IF(B, B, B)\}$

Derivation Step

- Application of a production, P, to non-terminal node $A \in N$, or
 - $\alpha A \beta \xRightarrow{A \rightarrow \theta} \alpha \theta \beta$
 - where $\alpha, \beta, \theta \in \{N \cup T\}$ and $A \in N$
- In this case the production $A \rightarrow \theta$ is a stochastic selection operator and subject to a priori thresholds for depth of the resulting ‘program tree’, see Figure 1.

Crossover

- Crossover is limited to locations defined by a non-terminal
- On choosing the first non-terminal, the second non-terminal must be of the same type or ‘grammar label’ (in the above example there is only one non terminal ‘B’).

Mutation

- Takes the form of a macro-mutation operator
- A non-terminal in the current tree is selected with uniform probability
- A new subtree is created using the ‘derivation step’
- The new subtree is inserted into the tree under the identified location

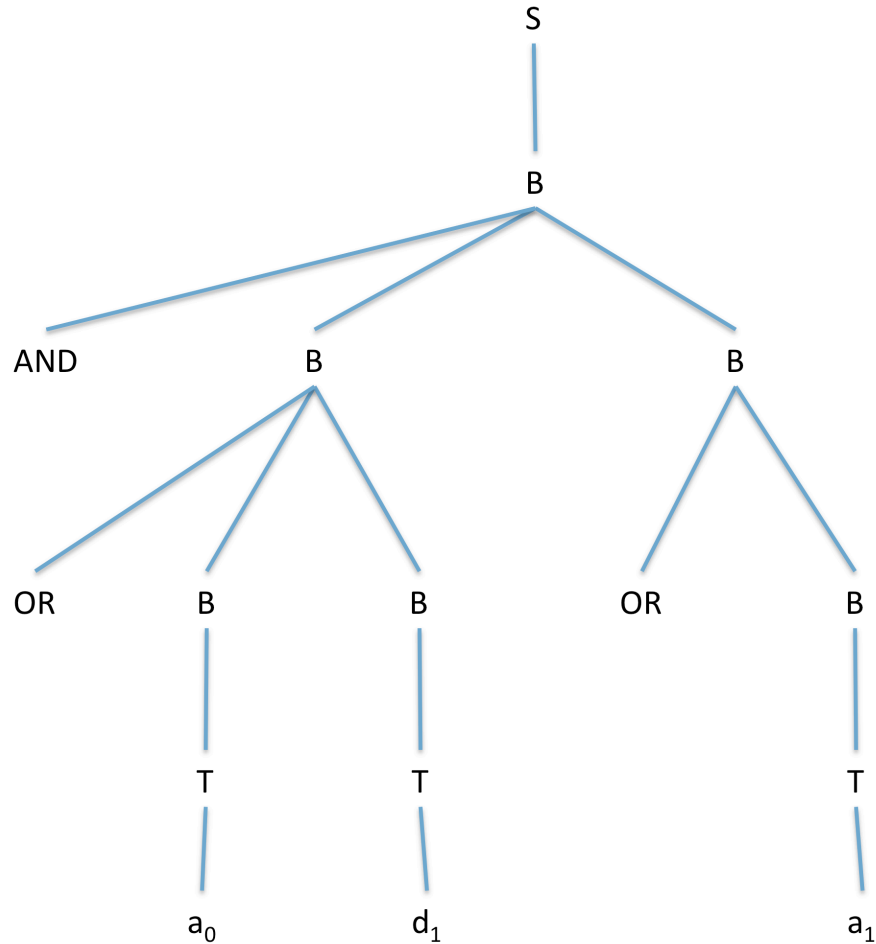


Figure 1: Example CFG derived program individual.

Introducing a priori Biases

- Multiple opportunities now exist for introducing a priori biases into individuals
- Under the multiplexer task we might begin by forcing the root to always take the form of a conditional statement, or
 - $S \rightarrow \text{IF } (B, B, B)$
- In addition we might add the knowledge that attribute ‘ a_1 ’ represents a significant control variable, or
 - $S \rightarrow \text{IF } (a_1, B, B)$
- Performance of the framework improves from a success rate of 29% to 41% to 73% respectively.

Biasing the productions applied

- At each generation, the population incrementally less diverse cf., convergence.
 - Promotion of more fit ‘building blocks’
- How can such building blocks be identified and actively promoted for reuse?
- Tree Structured GP introduced predefined ‘Automatically Defined Functions’
 - These had to be declared a priori
 - Increases the size of the search space, resulting in worse performance when the task is not sufficiently ‘difficult’ (Koza, 1994).
- Approach for CFG-GP will be based on the following assumptions:

Assumption 1: The population at each generation represents a sufficiently good source of candidate productions (building blocks).

Assumption 2: Fit programs contain suitable material for capturing as the basis for a new production.

- Implication being that we only consider the single fittest individual at each generation

A) Identifying useful productions

- Initial grammar is the most general. Thereafter any attempt to incorporate new productions will make the grammar more application specific.
- **Simplest scenario:**
 - Promote the terminal at the deepest point in the program to the parent non-terminal (figure 2).
 - New production of the form:
 - $B \rightarrow IF (a0 B B)$
 - When there is more than 1 terminal at the same depth, promote the left-most.
- **Anticipated impact:**
 - We incrementally refine arguments to establish broader functional units.

B) Marking productions through merit values

- Initially all productions carry the same ‘merit’ or probability of selection
- When productions are initialized they carry a merit of unity.
- Each time a production is used its merit increases, with the probability of applying a production being in proportion to its merit count.

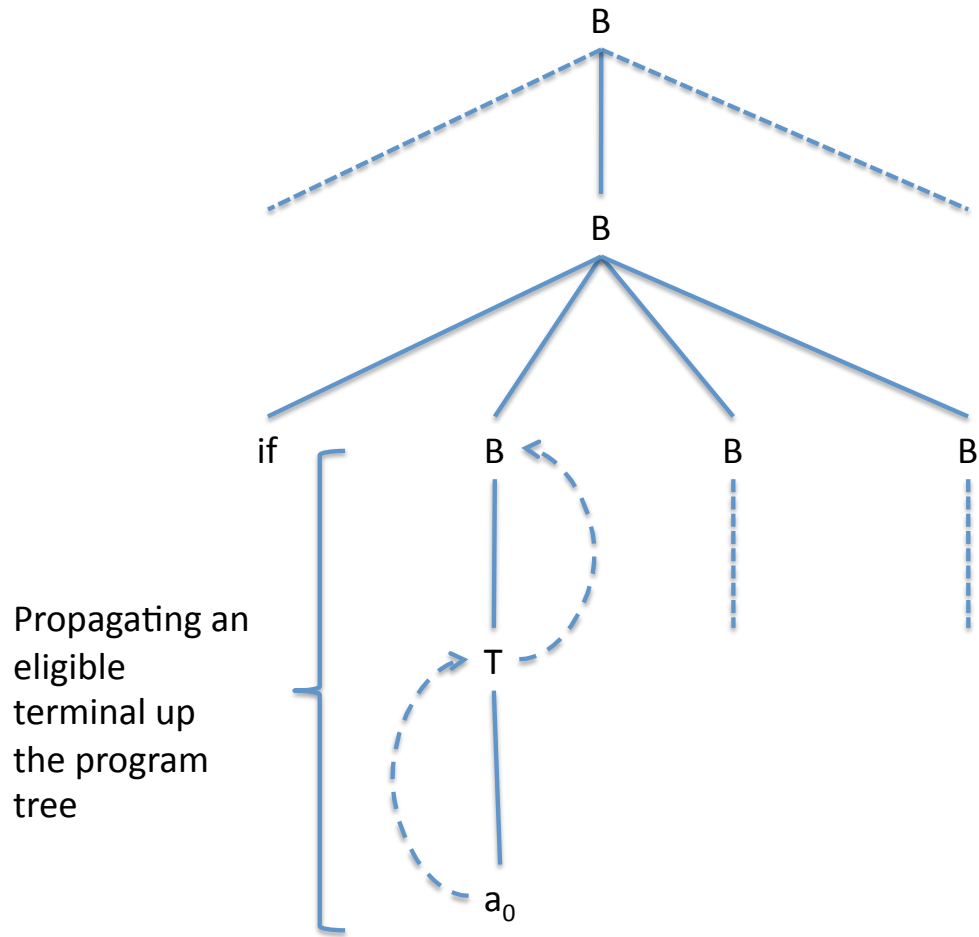


Figure 2: Identifying useful productions: a ‘terminal propagation’ approach.

C) Identification of ‘encapsulated functions’

- Case of all terminals at the deepest node of a program and no equal argument non-terminals (figure 3).
- Represents a candidate for explicit encapsulation through the replacement by a single new terminal

$$\circ B \rightarrow IF (a_0 \ d_0 \ d_1) \Rightarrow B \rightarrow t_1$$

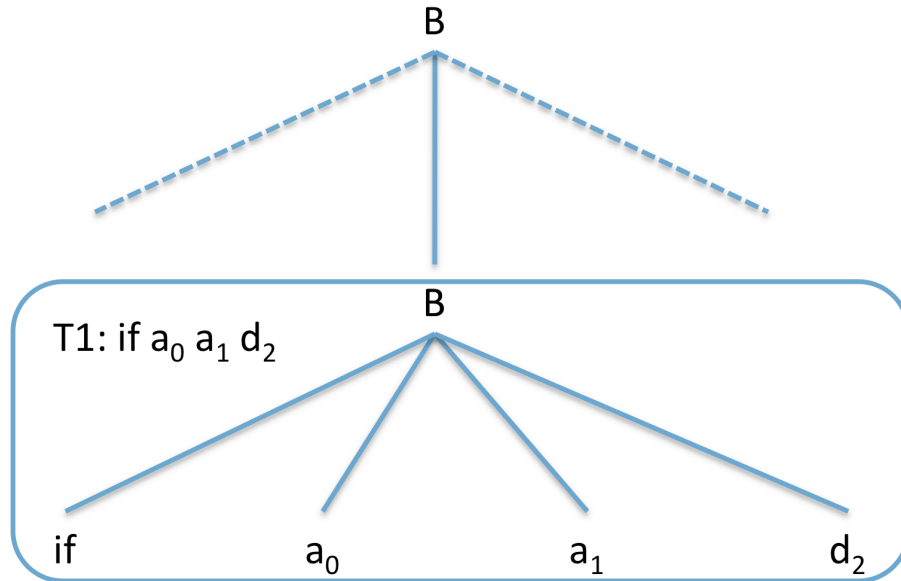


Figure 3: Encapsulation of a production.

6-bit multiplexer benchmark revisited

Configuration	Percent of initializations converging
Base CFG-GP	22%
CFG-GP + Identification of productions	52%
CFG-GP + Merit proportional selection of productions	66%
CFG-GP + Encapsulation	63%

Closing comments:

- Likely that ‘Encapsulation’ as defined here was too specific, actually constraining programs to local minima
 - Consider the ‘difference of volume’ task
- Merit proportional selection of productions does clearly provide a mechanism for identifying productions of utility (albeit to this particular task)

Grammatical Evolution

Basic Motivation

- To “*evolve complete programs in an arbitrary language using a variable length binary string.*” [1]
- As such a binary genome is used to define which production rules (from a context free grammar) are applied during the genotypic to phenotypic mapping.

Biological Motivation

- Genotypic material – the DNA – provides an encoded definition for proteins in the form of a sequence of building blocks, nucleotides or amino acids.
- A codon is the building block sequence defining components for a protein.
- All functional properties for a set of proteins appear from the same set of codons, but applied in different combinations.
- Proteins are then reflected phenotypically in terms of different measurable traits – eye colour, height etc.
- The biological process behind this is based on an intermediate representation in which the DNA is transcribed into the RNA, amino acid sequences identified and proteins configured accordingly, Figure 4.

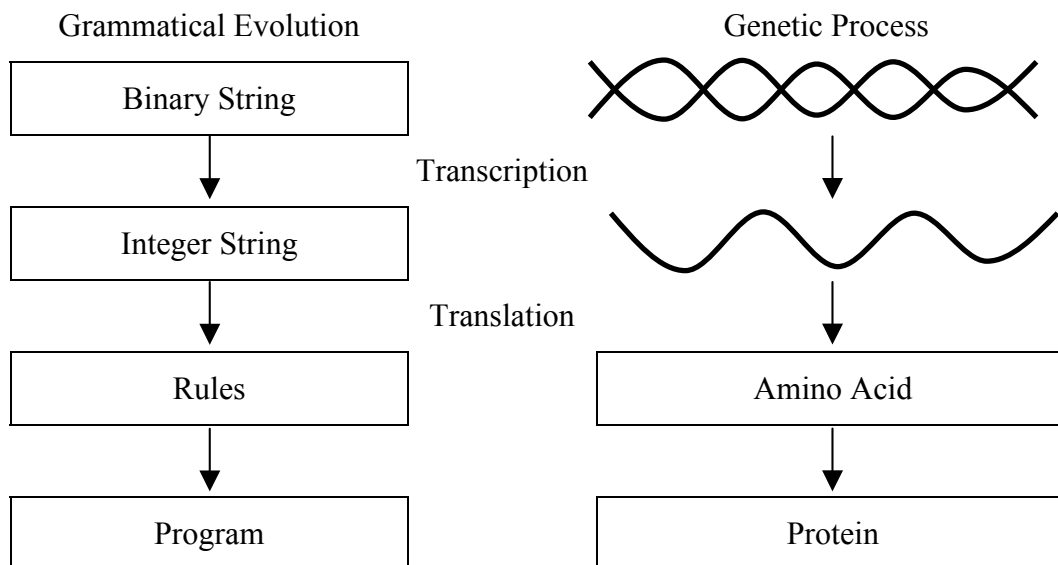


Figure 4 – Comparison between Grammatical Evolution and basic Genetic Process

Context Free Grammar under Grammatical Evolution

- As with Grammar-based GP, a BNF CFG represents the mechanism for decoding a genotype into a phenotype.
- Under GE the separation of genotype and phenotype will be more explicit.
- Consider the following CFG (O’Neill and Ryan, 2001),

$N =$	{ <i>expr</i> , <i>op</i> , <i>pre_op</i> }	
$T =$	{ <i>sin</i> , <i>+</i> , <i>-</i> , <i>/</i> , <i>x</i> , <i>x</i> , <i>1.0</i> , <i>(</i> , <i>)</i> }	
$S =$	< <i>expr</i> >	
P		
1.	< <i>expr</i> > ::= < <i>expr</i> > < <i>op</i> > < <i>expr</i> >	(0)
	(< <i>expr</i> > < <i>op</i> > < <i>expr</i> >)	(1)
	< <i>pre-op</i> > (< <i>expr</i> >)	(2)
	< <i>var</i> >	(3)
2.	< <i>op</i> > ::= +	(0)
	-	(1)
	/	(2)
	*	(3)
3.	< <i>pre-op</i> > ::= Sin	(0)
4.	< <i>var</i> > ::= x	(0)
	1.0	(1)

- Notes,
 - Terminals under the grammar clearly include both the “Terminal Set” and “Functional Set” defined for GP.
 - The phenotype is synonymous with a solution expressed purely in terms of the symbol set, T .
 - Each production rule has a corresponding number of ‘instances’, Table 1.

Table 1 – Instances per Production Rule for the above BNF language definition.

Rule ID	Instances
1.	4
2.	4
3.	1
4.	2

Grammatical Evolution

- Our starting point is a binary string consisting of codon sequences of ‘ b ’ bits [$b = 8$ in the following], where this is synonymous with the DNA or genotypic encoding in biology, Figure 4.
- Genetic search operators are applied to the binary string, thus the search process takes the form of a GA.
- Transcription (decoding) between DNA and RNA is synonymous with the decoding of binary codons into their equivalent integer equivalents, Figure 4.
 - Result is then a string of integers.
- Application of production rules to the integer string produces the resulting program or phenotype, where this is analogous to the construction of proteins from amino acid sequences, Figure 4.

Mapping of Integer Strings into equivalent Program

- Integer codons are mapped to non-terminals or terminals through the iterative application of the mapping function to the left most non-terminal,
 - Rule = (codon integer) MOD (rule instance)
 - Where Table 1 defines rule instance and S defines the first non-terminal.

Example

- Consider the following integer string:
 - 220, 240, 220, 203, 101, 53, 202, 203, 102, 55, 220, 241, 130, 37, 202, 203, 140, 39, 202, 203, 102
- Using the above BNF definition, we may now proceed to perform the genotype to phenotype mapping.

Integer	N	Rule	P
220	<expr>	220 MOD 4 \rightarrow 0	<expr> <op> <expr>
<expr> <op> <expr>			
240	<expr>	240 MOD 4 \rightarrow 0	<expr> <op> <expr>
<expr> <op> <expr> <op> <expr>			
220	<expr>	220 MOD 4 \rightarrow 0	<expr> <op> <expr>
<expr> <op> <expr> <op> <expr> <op> <expr>			
203	<expr>	203 MOD 4 \rightarrow 3	<var>
< var > <op> <expr> <op> <expr> <op> <expr>			
101	< var >	101 MOD 2 \rightarrow 1	1.0
1.0 <op> <expr> <op> <expr> <op> <expr>			
53	< op >	53 MOD 4 \rightarrow 1	-
1.0 - <expr> <op> <expr> <op> <expr>			
202	<expr>	202 MOD 4 \rightarrow 2	Sin(<expr>)
1.0 - Sin(<expr>) <op> <expr> <op> <expr>			
203	<expr>	203 MOD 4 \rightarrow 3	<var>
1.0 - Sin(< var >) <op> <expr> <op> <expr>			
102	< var >	102 MOD 4 \rightarrow 0	x
1.0 - Sin(x) <op> <expr> <op> <expr>			
... etc ...			
1.0 - Sin(x) \times Sin(x) - Sin(x) \times Sin(x)			

Notes,

- The above case provides exactly the right number of codons to map genotype into phenotype. However, this need not be the case.

- How might you deal with an individual which maps without employing all the codons?
- How might you deal with an individual which does not map after employing all the codons?

Code Degeneracy

- The mapping between codon values and production rule is many-to-one.
- Phenomenon also readily observed in biological organisms and referred to as *code degeneracy*.
 - E.g. On average 3 codons per amino acid
- From the perspective of mutation code degeneracy may result in no change at the phenotypic level although codon at the genotypic level has changed.

Genetic Algorithm

- At the genotypic level, crossover and mutation follow the standard GA algorithm.
- Codon duplication operator introduced,
 - Copies a random selection of codons to the end of the current individual.

Empirical Evaluation

- Benchmark Problems
 - Symbolic Regression #1: $f(x) = x^4 + x^3 + x^2 + x$; $x \in [-1 \dots 1]$
 - Symbolic Regression #2: $f(x) = \sin(x) + x + x^2$; $x \in [0 \dots 2\pi]$
 - Santa Fe Ant Trail:
 - Locate 89 food items on a discontinuous trail within 600 operations.
 - Trail defined on a 32 by 32 toroidal grid.
- Figures 4 and 5 of (O’Neill and Ryan, 2001) detail the cumulative frequency of solutions found, over generation for 100 different initializations.
 - Two experiments performed for the case of GP
 - With and without a solution depth constraint.
- Not clear what the GP alternative was!
 - Probably Koza’s standard Tree Structured GP.
- Results sufficient to demonstrate a “proof of concept”

Comments

- In a Genetic Programming context ‘functional’ and ‘terminal’ set should possess closure.

- Return type of any function or argument may represent the argument for any other function.
- Enables crossover to produce syntactically closed or well-formed individuals.
 - Strongly Typed GP requires specialist crossover types (Montana, 1995).
- Implication – Only one type in classical GP.

Search Operators for Grammatical Evolution

- Above study merely demonstrated the concept of Grammatical Evolution, it does nothing to demonstrate the competitiveness of the results.
- What would the significance of a single gene change be on an individual if the change were made at the beginning, middle, or end of a genotype?
 - What is the impact of the corresponding operation on tree or linearly structured GP?
- What impact would (two point) crossover operation have on the gene sequence following the second crossover point?
 - Again what the impact of the corresponding operation on tree or linearly structured GP?
- Mutation and Crossover in GE have multiple levels.
 - Search operators have the potential to impact either the entire individual or just the region local to the search operator.
 - Search operators impacting the entire individual will potentially introduce a lot of new behaviours into the population, at the expense of convergence.
 - Local search operators use information from genotype to phenotype mapping to guide/ constrain the application of appropriate mutation and crossover locations (Harper and Blair, 2005).
 - Restricted to boundaries set by genes that correspond to a non-terminals in the grammar once mapped to the phenotype.
 - Results in better-behaved convergence characteristics (Harper and Blair, 2005)

References

- Harper R., Blair A. (2005) A Structure Preserving Crossover in Grammatical Evolution. *IEEE Congress on Evolutionary Computation*, vol. 3, 2537-2544.
- Koza J.R. (1992) *Genetic Programming: On the programming of computers by means of Natural Selection*. MIT Press.
- Koza J.R. (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.
- Montana D. J. (1995) “Strongly Typed Genetic Programming,” *Evolutionary Computation*, 3(2), pp 199-230, 1995.
- O’Neill M., Ryan C. (2001) “Grammatical Evolution,” *IEEE Transactions on Evolutionary Computation*. 5(4), pp 349-358.
- O’Neill M., Ryan C., Keijzer M., Cattolico M. (2003) “Crossover in Grammatical Evolution,” *Genetic Programming and Evolvable Machines*. 4(1), Kluwer Academic, pp 67-93. March.
- Whigham P.A (1995) *Grammatically-based Genetic Programming*. Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. J.P. Rosca (ed) pp 33-41.