GRAMMATICAL EVOLUTION-COS314

1 Introduction to Grammatical Evolution

Grammatical evolution which was proposed by Ryan et al. [1] is considered to be an extension of GP. Unlike GP which uses syntax trees to represent individuals, GE uses chromosomes of variable length binary strings. Each gene of the chromosome is an 8-bit binary string and is referred to as a codon. Codons contain information on how to select production rules from a programming language grammar. Grammatical evolution uses the context free Backus Naur Form grammar. Figure 1 is an illustration of two examples of GE individuals where a) is a genome consisting of 6 codons and b) consists of 8 codons.

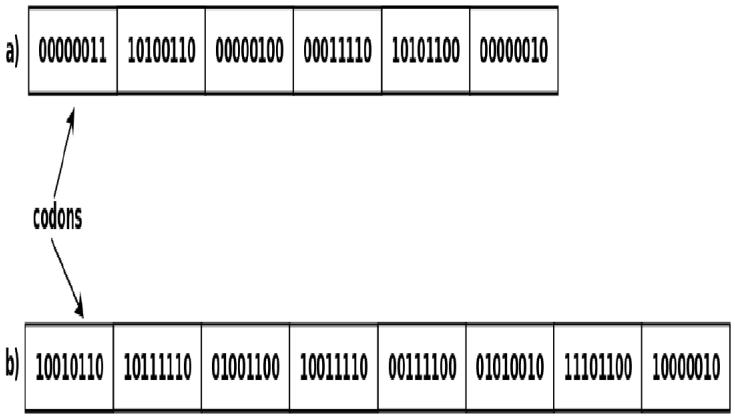

Figure 1: Codon Examples

Grammatical evolution uses a user-defined BNF grammar to map variable length linear genomes to executable programs. Domain knowledge of the problem being solved is incorporated into the grammar. The grammar is used in conjunction with evolution to map a genotype to a phenotype. Grammatical evolution draws inspiration from molecular biology where a sequence of genetic material, deoxyribonucleic acid (DNA)(genotype) is translated into a protein which defines the characteristics of a phenotype. A grammar G can be represented by the four-tuple $<$ N, T, P, S $>$, where N represents a set of non-terminals, T a set of terminals, P a set of production rules that map the elements of N to T and S (a member of N) the start symbol. An individual (genotype) is used to map the start symbol S to terminals by reading and converting each codon to its decimal value from which an appropriate production rule is selected by using the following mapping function:

Rule = (codon decimal value)%(No of production rules)..... (1)

A derivation tree (phenotype) is evolved by iterating and mapping through the sequence of codons. The derivation process is performed from left to right starting with the left-most non-terminal. If the iteration process reaches the

end of the sequence of codons before the derivation tree is evolved the procedure continues by looping to the start of the codon sequence, a process called *wrapping*. The fitness of the phenotype is evaluated by applying it to a problem at hand.

Grammatical Evolution Algorithm

Algorithm 1 Grammatical Evolution

1: Create an initial population of variable length binary strings
2: Map via a BNF grammar
    a) binary strings to expression using production rules
3: Evaluate fitness
4: **do while** {termination condition not met}
5: Select fitter individuals for reproduction
6: Recombine selected individuals
7: Mutate offspring
8: Evaluate fitness of offspring
9: Replace all individuals in the population with offspring
10: **end while**
11: **return** best individual

Algorithm 1 is an outline of the step by step GE algorithm.

Initial Population Generation

The first step of the GE algorithm is to randomly generate a population of variable length binary strings (individuals). The population size and the variable length limits are user specified. The individual lengths are determined randomly from a lower bound and upper bound range of given values eg [8 - 20].

Mapping

After population initialisation the randomly generated genotypes are mapped into phenotypes. The mapping process involves the BNF grammar and the rule specified by equation 1 to produce valid phenotypes. The mapping is deterministic meaning if the grammar remains the same, a particular genotype will always be mapped to the same phenotype.

Selection

The commonly used selection methods in GE are tournament selection and fitness proportionate selection.

Crossover

Single-point crossover is the most widely used crossover operator in GE. This is applied in a similar manner as in genetic algorithms except that in GE it is used on variable length genomes resulting in variable length offspring. Two parents are selected using a chosen selection method and a crossover probability rate is

used to determine if the crossover operator should be applied. If crossover is to be applied a random crossover point r in the range [1, l -1] where l is the maximum length of the shortest parent is generated and used as the crossover point where the tails of each parent are exchanged.

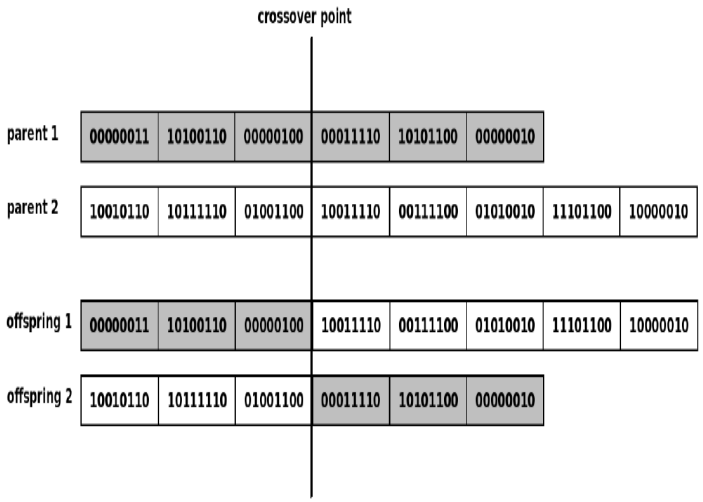

Figure 2: Single-point Crossover

Mutation

Mutation is applied in the same manner as in genetic algorithm, e.g bit mutation.

Population Replacement

Grammatical evolution follows a similar approach as the other EAs when it comes to population replacement. Population replacement can either be generational or steady-state replacement. Elitism may also be applied in GE.

Termination

Like the other EAs termination can occur either after a certain number of generations or if a problem specific solution is obtained.

2 Applications of Grammatical Evolution

There are several examples of the use of Grammatical Evolution in various domains, including:

1. Circuit Design: Grammatical Evolution has been used to evolve circuits that perform specific functions, such as logic gates or filters. The grammar specifies the allowable circuit components and their connections, and the evolutionary process searches for the best circuit that meets the design requirements.

2. Image Processing: GE has been used to evolve image filters that can enhance or modify images in specific ways. The grammar specifies the operations that can be applied to the image, such as blurring, sharpening, or edge detection, and the evolutionary process searches for the best filter that meets the desired criteria.

3. Game AI: GE has been used to evolve game AI agents that can play games such as chess or poker. The grammar specifies the allowable moves and strategies, and the evolutionary process searches for the best agent that can win the game against opponents.

4. Language Processing: GE has been used to evolve natural language processing models that can generate coherent sentences or classify text. The grammar specifies the allowable sentence structures and word choices, and the evolutionary process searches for the best model that can generate or classify text.

Overall, Grammatical Evolution has shown promising results in a wide range of applications, where it can evolve complex solutions that meet specific requirements and are difficult to design manually.

3 Pin Generation Example

Given the following individual where GE is evolving pin numbers the mapping proceeds as follows.

00000011	10100110	00000100	00011110	10101100	00000010

Figure 3: GE Individual

A grammar for generating pin numbers is required and is shown in Figure 4. The grammar is problem dependant and contains problem domain information.

<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>
<W> ::= <Y><X> | <Y><Z>
<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<Y> ::= a | b | c | d | e | f | g | h | i | j
<Z> ::= @ | ~ | # | & | % | * |) | - | (| +

Figure 4: GE Grammar

A strong password must contain a combination of alpha-numeric and special characters. Any symbol enclosed in <> is considered to be a non-terminal meaning is can be further simplified by substitution. A terminal is a final stand

alone value. The first line of the given grammar can be interpreted as follows:

< S > can be replaced by rule 0 whose value is < X > < W > < W >

OR(signified by |)

by rule 1 whose value is < W > < X > < Y >

OR(signified by |)

by rule 2 whose value is < X > < W > < Y >

Because the rules for <S> contain non terminals it means they can be further simplified. The rest of the lines of the grammar follow the same format with < W > having 2 rules (rule 0 and rule 1), < X > not only has 10 rules but they are also terminals i.e final values. Similarly < Y > and < Z > have 10 rules containing terminals The rule selected is determined by processing the codon values with equation 1 as follows: we start by obtaining the decimal values of the codons of an individual. So the decimal values of the codons of the individual depicted in Figure 3 is given as : 5,166,4,30,172,2
Pin = <S> keeping in mind S is the starting point. We proceed as follows:

5,166,4,30,172,2

and

$$Rule = (codon\ \ decimal\ \ value)\%(N^o\ \ of\ \ production\ \ rules) \tag{1}$$

1. <S> = 5 % 3 = 2 (5 codon decimal value and 3 is the number of production rules. We use equation 1 to select rule). This means we select rule 2. Rule 2 which is < X > < W > < Y > replaces <S>. Since the production rule contains non-terminals we proceed with the mapping taking the left most non-terminal with the next codon value.

2. < X > = 166 % 10 = 6 (166 second codon decimal value and 10 is the number of rules associated with X.) This means we select production rule 6 resulting in the following mapping 7 < W > < Y >. following the same procedure we simplify < W >.

3. 4% 2 = 0 production rule 0 → 7<Y> <X> <Y>

4. 30%10 = 0 production rule 0 → 7 a <X><Y>

5. 172%10 = 2 production rule 2 → 7 a 3 <Y>

6. 2%10 = 0 production rule 7 → 7 a 3 a

The evolved phenotype (pin) is **7a3a**. The fitness of the phenotype is measured using a problem dependent fitness function. For pin numbers, this can be standard password metrics such as a numeric evaluation of complexity. The derivation tree for this example is shown below.

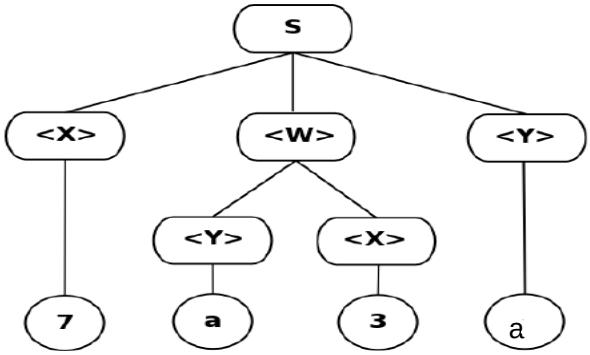

Figure 5: GE Derivation Tree

Reference

[1] O'Neill, M., & Ryan, C. (2001). Grammatical evolution. IEEE Transactions on Evolutionary Computation, 5(4), 349-358.