Nature-Inspired Computing

Grammatical Evolution

Dr. Şima Uyar
September 2007

Overview

- Michael O'Neill and Conor Ryan
- 1998 – 2001
- binary chromosomes
- grammar defined through BNF
- genotype to phenotype mapping
- regular genetic algorithm operators
- produces syntactically correct programs in any language (originally developed for Lisp)

Backus Naur Form (BNF)

- notation for expressing the grammar of a language in the form of production rules
- BNF grammars consist of:
  - terminals (items which appear in language, e.g. + or -)
  - non-terminals (can be expanded into one or more terminals or non-terminals)

Backus Naur Form (BNF)

- a grammar represented by tuple {N,T,P,S}
  - N: set of non-terminals
  - T: set of terminals
  - P: set of production rules mapping elements of N to T
  - S: a start symbol which is a member of N

Backus Naur Form (BNF)

- when more than one production can be applied to one element of N, they are seperated by the "|" symbol

Backus Naur Form (BNF)

example:

```
N={expr,op, pre-op,var}
T={sin,cos,+,-,/,*,X,1.0,(,)}
S=expr
```

Backus Naur Form (BNF)

example (*cntd.*):

P can be represented as:

```
(A) <expr> ::= <expr> <op> <expr>          (a)
             | (<expr> <op> <expr>)        (b)
             | <pre-op> (<expr>)           (c)
             | <var>                       (d)

(B) <op> ::= +
           (e)
           | -                             (f)
           | /                             (g)
           | *                             (h)

(C) <pre-op> ::= sin                       (i)
               | cos                       (j)

(D) <var> ::= X                            (k)
            | 1.0                          (l)
```

Grammatical Evolution

- 8 bits = codons
- integer values of codons used to select production rule
- wrapping of chromosome is used

Grammatical Evolution

- beginning from left hand side of genome, codon integer values are generated and used to select rules from BNF grammar until one of these arise:
  - a complete program is generated (all non-terminals in expression is transformed into elements from the terminal set)
  - end of genome reached; wrapping operator is invoked; continue from left; repeat until max no of wrapping reached
    - if max no of wrappings reached and individual is not complete, it is assigned the lowest possible fitness

Example Individual

220 40 16 203 101 53 202 203 102 55 220 202 19 130 37 202 203 32 39 202 203 102

Step 1: S=expr $\Rightarrow$ 4 to choose from in rule A

220 mod 4 = 0, so first production is chosen

<expr> replaced with <expr><op><expr>

Example Individual

220 40 16 203 101 53 202 203 102 55 220 202 19 130 37 202 203 32 39 202 203 102

Step 2: continue with first <expr>: 40 mod 4=0, so

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

Step 3: 16 mod 4 = 0 so
<expr><op><expr><op><expr><op><expr>
Step 4: 203 mod 4=3 so
<var><op><expr><op><expr><op><expr>
Step 5: 101 mod 2 =1
<1.0><op><expr><op><expr><op><expr>
Step 6: 53 mod 4=1
<1.0><-><expr><op><expr><op><expr>
Step 6: 202 mod 4=2
<1.0><-><pre-op>(<expr>)<op><expr><op><expr>
Step 7: only one choice for pre-op, no codon used
<1.0><-><sin>(<expr>)<op><expr><op><expr>
Step 8: 203 mod 4=3
<1.0><-><sin>(<var>)<op><expr><op><expr>
Step 9: 102 mod 2=0
1.0-sin(x)<op><expr><op><expr>
...
...
Outcome: 1.0-sin(x)*sin(x)-sin(x)*sin(x)

Grammatical Evolution

- issues:
  - degeneracy (due to modulo operator)
  - neutral mutations
  - disruptive effect of one point crossover
    - work on special crossovers
- many applications
- work still continues