Nature-Inspired Computing

Genetic Programming

Dr. Şima Uyar
September 2006

Overview

- developed in the USA
- typically applied to:
  - machine learning tasks such as prediction, classification ...
- properties
  - very large populations (thousands)
  - slow
  - has non-linear chromosomes: trees, graphs
  - mutation not always used

Typical GP

Representation	Tree structures
Recombination	Exchange subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Tree-based Representation

- trees can represent:
  - an arithmetic formula
  - a logical formula
  - a program

Tree-based Representation

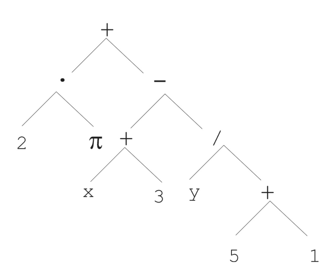
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

```
i =1;
while (i < 20)
{
        i = i +1
}
```

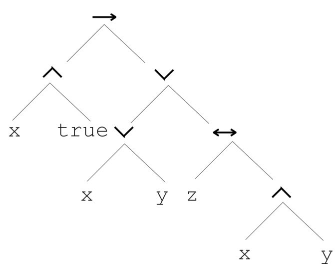
Tree-based Representation

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$



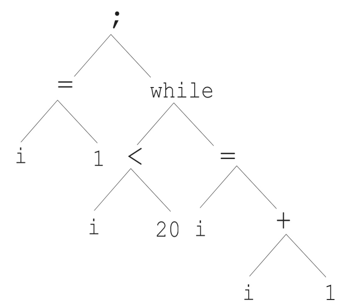
Tree-based Representation

$(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$



Tree-based Representation

```
i =1;
while (i < 20)
{
        i = i +1
}
```



Tree-based Representation

- chromosomes as:
  - bit strings, integer string, real-valued vectors, permutations ⇒ linear structures (GAs and ES)
  - trees ⇒ non-linear structures (GPs)
- in Gas and ES: fixed chromosome size
- in GP: tree (chromosome) depth/width may change

Tree-based Representation

- a symbolic expression can be defined by,
  - a terminal set: T
  - a function set: F
- typically, expressions in GP are not typed
  - closure property: any f ∈ F can take any g ∈ F as argument

Terminal Set

- composed of:
  - inputs (variables)
  - constants
  - zero-argument functions
- are the leaves of the tree
- terminal nodes have an arity of zero

Function Set

- composed of:
  - statements
  - operators
  - functions
- members of set determined based on application

Function Set

- boolean functions (AND, OR, NOT, ...)
- arithmetic functions (+, -, *, /, ...)
- transcendental functions (trigonometric and logarithmic functions)
- variable assignment functions (=)
- conditional statements (if-then-else, switch-case, ...)
- control transfer statements (goto, jump, ...)
- loop statements (while – do, repeat – until, for, ...)
- subroutines

Choosing the Function Set

- not too small or too large
- if too small, cannot solve problem
- if too large, large search space
- good starting point:
  - +, -, *, /, AND, OR, XOR
- must have closure property:
  - division by zero is a problem; closure property violated ⇒ define protected division operator instead

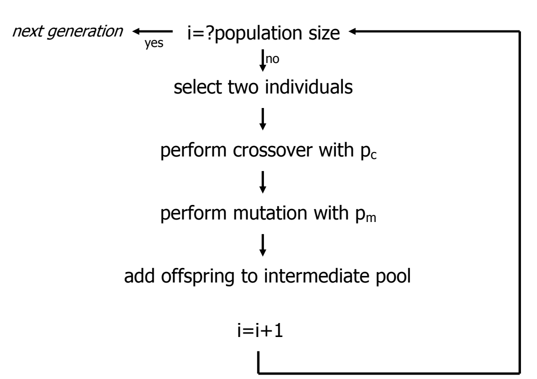
Offspring Generation

- GAs use crossover AND mutation (probabilistically)
- GPs use crossover OR mutation (chosen probabilistically)

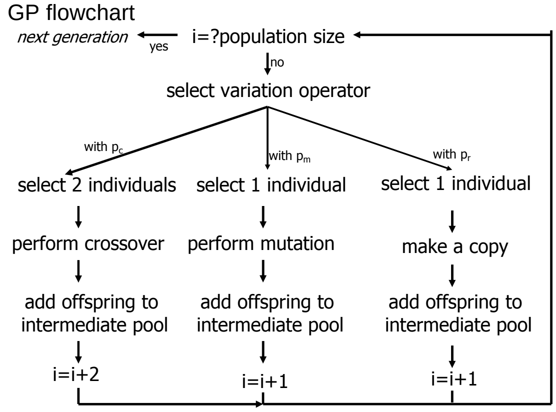
Offspring Generation

- GP operators: crossover, mutation, reproduction
- there is a probability for selection of each operator
  - p_m : probabilitiy of mutation
  - p_c : probabilitiy of crossover
  - p_r : probabilitiy of reproduction
- $p_m + p_c + p_r = 1$

GA flowchart



GP flowchart



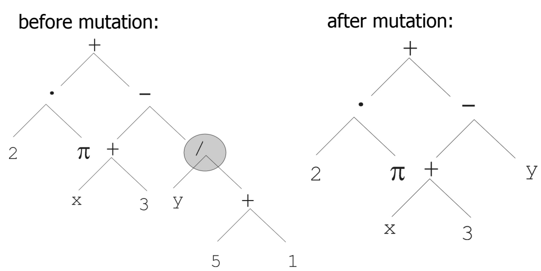
Reproduction

- one individual selected
- copy of individual made
- copy added to offspring pool
- has parameter p_r

Mutation

- typical mutation: replace randomly chosen sub-tree by randomly generated tree

Mutation



Mutation

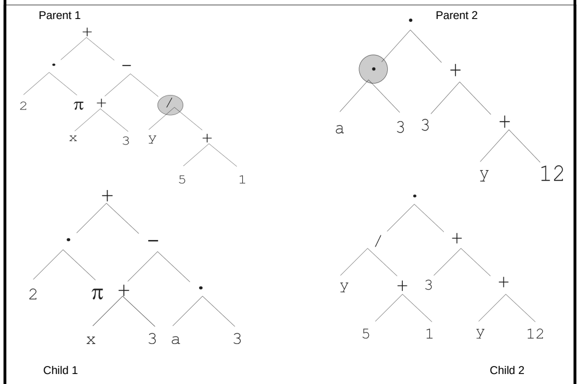
- has two parameters:
  - probability p_m to choose mutation vs. recombination
  - probability to chose an internal point as the root of the sub-tree to be replaced
- p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- size of the child can be larger than the parent

Recombination

- typical recombination: exchange two randomly chosen sub-trees among parents

Recombination

- has two parameters:
  - probability p_c to choose recombination vs. mutation
  - probability to chose an internal point within each parent as crossover point
- size of the children may be larger than the parents



Selection

- typical parent selection: fitness proportionate
- truncation selection also used
  - (μ,λ)
  - $(\mu+\lambda)$
- ranking selection, tournament selection possible

Selection

- typical survivor selection: generational scheme
  - recently steady-state is becoming popular due to its elitism

Initialization

- maximum initial depth of trees: D_{max}
- full method (each branch has depth = D_{max})
  - nodes at depth $d < D_{max}$ randomly chosen from function set F
  - nodes at depth $d = D_{max}$ randomly chosen from terminal set T

Initialization

- grow method (each branch has depth $\leq D_{max}$)
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - nodes at depth $d = D_{max}$ randomly chosen from T

Initialization

- typical GP initialisation: ramped half-and-half
  - grow method and full method each used to generate half of the initial population

Bloat

- bloat = "survival of the fattest", i.e., tree sizes increase over time
- must be prevented
  - do not allow very big children
  - parsimony pressure: apply penalty for being oversized

Introns

- extra code segments which, if removed, will not alter the result
  - e.g. a = a + 0
  - e.g. b= b * 1
- bloat mainly caused by introns

Preparatory Steps

- determining the set of terminals
- determining the set of functions
- determining the fitness measure
- determining the parameters
  - population size
  - maximum tree depth
  - p_c, p_m, p_r
  - number of generations
- determining the method for
  - designating a result and
  - the criteria for termination

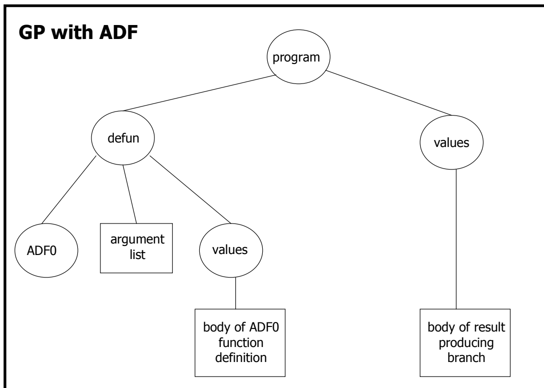
Example application: symbolic regression

- given some points in $\mathbf{R}^2$, (x_1, y_1), …, (x_n, y_n)
- find function f(x) s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- possible GP solution:
  - representation: F={+, -, /, sin, cos}, T=$\mathbf{R} \cup \{x\}$
  - fitness is the error $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - all standard operators used
  - population size: 1000, ramped half-half initialization
  - termination: n "hits" or 50000 fitness evaluations reached (where "hit" is if $| f(x_i) - y_i | < 0.0001$)

Modularization – Automatically Defined Functions (ADFs)

- individual tree consists of two sub-trees:
  - result-producing branch (main)
  - function defining branch (function definitions)

GP with ADF



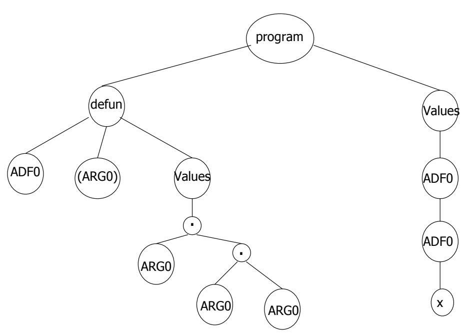
GP with ADF

- a defun node per ADF
- "Values" nodes determine the result (overall or from ADF)
- argument list in ADF, determines the ADF's input variables
  - becomes part of terminal set of ADF
- all evolution takes place in ADF bodies and the result producing branchS
- possible to have hierarchies between ADFs, determining which ADF is able to call which ADF (depends on system set up)

GP with ADF

- first determine *architecture*
  - number of ADFs
  - the number of arguments for each ADF
- this is a weakness since *architecture* must be determined by user
  - adds a new parameter to GP
  - architecture altering operations (type of mutation)
- initialization done accordingly
- function-defining bodies and result-producing bodies generated randomly

Example: ADF for x^3 and result producing branch uses ADF to get x^6



Steps when Applying GP with ADFs

- choose number of function defining branches
- fix number of arguments for each ADF
- determine allowable referencing between ADFs
- determine all function and terminal sets (may be different for all)
- define fitness measure, fix parameters and termination criteria