Genetic Programming Tutorial

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
btzhang@scai.snu.ac.kr
http://scai.snu.ac.kr/~btzhang/

The Second Asia Pacific Conference on Simulated Evolution and Learning (SEAL98)
Canberra, Australia
Tuesday, 24 November 1998
2:00 - 5:30 PM

This material available online at
http://scai.snu.ac.kr/~btzhang/

Outline

Introduction
- Background on Evolutionary Algorithms (EAs)

Genetic Programming (GP)
- Representation, Genetic Operators, Running a GP

GP Applications
- AI, Alife, Engineering, Science

Advanced Topics
- Variants of Genetic Programming
- Techniques for Enhancing GP Performance

Guidelines
- Promising Application Areas
- Research Issues

Further Information on GP

Introduction
Evolutionary Algorithms (EAs)

- A computational model inspired by natural evolution and genetics
- Proved useful for search, machine learning and optimization
- Population-based search (vs. point-based search)
- Probabilistic search (vs. deterministic search)
- Collective learning (vs. individual learning)
- Balance of exploration (global search) and exploitation (local search)

Analogy to Evolutionary Biology

- Individual (Chromosome) = Possible solution
- Population = A collection of possible solutions
- Fitness = Goodness of solutions
- Selection (Reproduction) = Survival of the fittest
- Crossover = Recombination of partial solutions
- Mutation = Alteration of an existing solution

Simulated Evolution

Canonical Evolutionary Algorithm

begin
  t = 0 /* generation */
  initialize P(t) /* population */
  evaluate P(t)
while (not termination-condition) do
begin
  t = t + 1
  select P(t) from P(t-1) /* selection */
  crossover-mutate P(t) /* genetic operators */
  evaluate P(t) /* fitness function */
end
end
Variants of Evolutionary Algorithms

- Evolutionary Programming (EP)
  - Fogel et al., 1960’s
  - FSMs, mutation only, tournament selection

- Evolution Strategy (ES)
  - Rechenberg and Schwefel, 1960’s
  - Real values, mainly mutation, ranking selection

- Genetic Algorithm (GA)
  - Holland et al., 1970’s
  - Bitstrings, mainly crossover, proportionate selection

- Genetic Programming (GP)
  - Koza, 1992
  - Trees, mainly crossover, proportionate selection

- Others

Genetic Operators for Bitstring Chromosomes

- **Reproduction**: make copies of chromosome (the fitter the chromosome, the more copies)
  
<table>
<thead>
<tr>
<th>Original</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 0</td>
<td>1 0 0 0 1 0 0</td>
</tr>
<tr>
<td>1 0 0 0 1 0 0</td>
<td>1 0 0 0 1 0 0</td>
</tr>
</tbody>
</table>

- **Crossover**: exchange subparts of two chromosomes
  
<table>
<thead>
<tr>
<th>Original</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0</td>
<td>0 0 1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1 1 1</td>
</tr>
</tbody>
</table>

- **Mutation**: randomly flip some bits
  
<table>
<thead>
<tr>
<th>Original</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Selection

Create random initial population

Evaluate population

Select individuals for variation

Vary

Insert to population

Selection Schemes

- **Proportionate selection**
  - Reproduce offspring in proportion to fitness $f_i$.

- **Ranking selection**
  - Select individuals according to $rank(f_i)$.

- **Tournament selection**
  - Choose $q$ individuals at random, the best of which survives.

- **Generational vs. steady-state**
Theory of Bitstring EAs

- **Assumptions**
  - Bitstrings of fixed size
  - Proportionate selection

- **Definitions**
  - Schema $H$: A set of substrings (e.g., $H = 1**0$)
  - Order $o$: number of fixed positions (FP) (e.g., $o(H) = 2$)
  - Defining length $d$: distance between leftmost FP and rightmost FP (e.g., $d(H) = 3$)

- [Holland, 1975]

Schema Theorem

\[
m(H,t+1) \geq m(H,t) \frac{f(H,t)}{\tilde{f}(t)} \left(1 - p_c \frac{d(H)}{n-1}\right)^{o(H)} (1 - p_m)^{o(H)}
\]

- Number of members of $H$
- Probability of crossover and mutation, respectively

Interpretation: Fit, short, low-order schemata (or building blocks) exponentially grow.

Some Applications of EAs

- **Optimization** (e.g., numerical optimization, VLSI circuit design, gas turbine design, factory scheduling)
- **Automatic Programming** (e.g., automatic induction of LISP programs, evolving optimal sorting algorithms)
- **Complex Data Analysis and Time-Series Prediction** (e.g., prediction of “chaotic” technical systems, financial market prediction, protein-structure analysis)
- **Machine and Robot Learning** (e.g., rule induction for expert systems, evolutionary learning of neural networks, cooperation of multiple mobile agents, robot navigation)

Genetic Programming (GP)
GP Trees

- Genetic programming uses variable-size tree-representations rather than fixed-length strings of binary values.
- Program tree = S-expression = LISP parse tree
- Tree = Functions (Nonterminals) + Terminals

GP Tree: An Example

S-expression: \((+ 1 2 (IF (> TIME 10) 3 4))\)
Terminals = \{1, 2, 3, 4, 10, TIME\}
Functions = \{+, >, IF\}

A GP Tree for Kepler’s Law

- GP-tree representation of Kepler’s third law: 
  \[ P^2 = cA^3 \]

```
PROGRAM ORBITAL_PERIOD
C # Mars #
A = 1.52
P = SQRT(A * A * A)
PRINT P
END ORBITAL_PERIOD
```

GP as Automatic Programming

- GP evolves a program for solving a class of problem instances. The solution found by GP is a program that solves many problem instances.
- GP is an automatic programming method.
Setting Up for a GP Run

1. The set of terminals
2. The set of functions
3. The fitness measure
4. The algorithm parameters
   * population size, maximum number of generations
   * crossover rate and mutation rate
   * maximum depth of GP trees etc.
5. The method for designating a result and the criterion for terminating a run.

Genetic Programming Procedure

1. Choose a set of possible functions and terminals for the program: \( F = \{ +, -, *, /, \sqrt{\cdot} \}, \ T = \{ \cdot \} \).
2. Generate an initial population of random trees (programs) using the set of possible functions and terminals.
3. Calculate the fitness of each program in the population by running it on a set of “fitness cases” (a set of input for which the correct output is known).
4. Apply selection, crossover, and mutation to the population to form a new population.
5. Repeat steps 3 and 4 for some number of generations.

Crossover: Subtree Exchange

Mutation
Example GP Run: Majority

Problem: Given five binary inputs $x_1, x_2, \ldots, x_5$, return $y = 1$ if three or more of $x_i$ are 1 and output $y = 0$ otherwise.

Fitness cases given (20 out of 32):

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Majority: Best Program at Generation 0

- Training error = 4/20
- Generalization error = 8/32

OR

Majority: Best Program at Generation 1

- Training error = 3/20
- Generalization error = 8/32

OR

Majority: Best Program at Generation 11

- Training error = 2/20
- Generalization error = 6/32

OR
### Majority: Best Program at Generation 17

- Training error = $1/20$
- Generalization error = $5/32$

### Majority: Evolution of Fitness Values

![Fitness Evolution Graph](image)

### A List of GP Applications

- Genetic programming has been applied to a wide range of problems in artificial intelligence, artificial life, engineering, and science, including the following:
  - Symbolic Regression
  - Multi-Agent Strategies
  - Simulated Robotic Soccer
  - Time Series Prediction
  - Circuit Design
  - Evolving Neural Networks
Symbolic Regression

**Given:** a set of $N$ data points

\[ D = \{(x_i, y_i) \mid i=1,...,N\} \]

**Find:** a symbolic expression of the function $f$ that minimizes the error measure:

\[
E_f(D) = \sum_{i=1}^{N} (y_i - f(x_i))^2
\]

**Useful for** system identification, model building, empirical discovery, data mining, and time series prediction.

Symbolic Regression: Fitness Cases [Koza, 1998]

<table>
<thead>
<tr>
<th>Independent Variable X</th>
<th>Dependent Variable Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>1.00</td>
</tr>
<tr>
<td>-0.9</td>
<td>-0.1629</td>
</tr>
<tr>
<td>-0.8</td>
<td>-0.2624</td>
</tr>
<tr>
<td>-0.7</td>
<td>-0.3129</td>
</tr>
<tr>
<td>-0.6</td>
<td>-0.3264</td>
</tr>
<tr>
<td>-0.5</td>
<td>-0.3125</td>
</tr>
<tr>
<td>-0.4</td>
<td>-0.2784</td>
</tr>
<tr>
<td>-0.3</td>
<td>-0.2289</td>
</tr>
<tr>
<td>-0.2</td>
<td>-0.1664</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.0909</td>
</tr>
<tr>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1111</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2494</td>
</tr>
<tr>
<td>0.3</td>
<td>0.4251</td>
</tr>
<tr>
<td>0.4</td>
<td>0.6496</td>
</tr>
<tr>
<td>0.5</td>
<td>0.9375</td>
</tr>
<tr>
<td>0.6</td>
<td>1.3056</td>
</tr>
<tr>
<td>0.7</td>
<td>1.7731</td>
</tr>
<tr>
<td>0.8</td>
<td>2.3616</td>
</tr>
<tr>
<td>0.9</td>
<td>3.0951</td>
</tr>
<tr>
<td>1.0</td>
<td>4.0000</td>
</tr>
</tbody>
</table>

Symbolic Regression: Experimental Setup

**Objective:** Find a function of one independent variable, in symbolic form, that fit a given sample of 20 $(x, y)$ data point.

**Terminal set:** $x$ (the independent variable).

**Function set:** +, -, *, %, SIN, COS, EXP, RLOG

**Fitness cases:** The given samples of 21 data points $(x_i, y_i)$ where the $x_i$ come from the interval $[-1, +1]$.

**Raw fitness:** The sum, taken over the 21 fitness cases, of the absolute value of difference between value of produced by the individual program and the target values $y_i$ of the dependent variable.

**Standardized fitness:** Equals raw fitness.

**Hits:** Number of fitness cases (0-21) for which the value of the dependent variable produced by the individual program comes within 0.01 of the target value $y_i$ of the dependent variable.

**Wrapper:** None.

**Parameters:** $m = 500$, $G = 51$

**Success Predicate:** An individual program scores 21 hits.
Symbolic Regression:
Generation 34

- Best-of-run individual with raw fitness of 0.00 (100% correct)

- Equivalent to $x^4 + x^3 + x^2 + x$

Symbolic Regression:
Observations

- GP works on this problem.
- The answer is algebraically correct (hence no further cross validation is needed).
- It’s not how a human programmer would have written it.
  - Not parsimonious
  - $\cos x - x$
- The extraneous functions - $\sin$, $\exp$, $\text{RLog}$, and (effectively) $\text{RCos}$ are all absent in the best individual of generation 34.

Multi-Agent Strategies
[Benenett III, 1996]

- The Foraging Problem
  - $32 \times 32$ grid for the ant colony world
  - Two food locations with 72 food pellets (black)
  - The nine grid locations of the nest (gray)

- Objective
  - Find a multi-agent parallel algorithm that causes efficient central-place foraging behavior in the ant colony.

Multi-Agent Strategies: Fitness Function

\[
\sum (t_{\text{food}} \cdot f_{\text{food}}) + \sum (t_{\text{max}} \cdot f_{\text{max}} \cdot d_{\text{food}}) / 1,000,000
\]

- $n =$ Number of food pellets transported to the nest
- $t_{\text{food}} =$ Number of time steps elapsed when the food pellet arrived at the nest
- $f_{\text{food}} =$ Number of sequential IF functions executed by the ant who transported the food pellet
- $m =$ Number of food pellets not transported to nest.
- $t_{\text{max}} =$ Maximum allotted time step = 4,000
- $f_{\text{max}} =$ Maximum possible value of $f_{\text{food}} = 400,000$
- $d_{\text{food}} =$ Manhattan distance between food pellet and nest
- $p_{\text{max}} =$ Maximum number of points per agent = 100
Multi-Agent Strategies: Experimental Setup

- **Function set**
  - IF_FOOD_HERE, IF_FOOD_FORWARD, IF_CARRYING_FOOD,
  - IF_NEST_HERE, IF_FACING_NEST, IF_SMELL_FOOD,
  - IF_SMELL_PHEROMONE, IF_PHEROMONE_FORWARD

- **Terminal set**
  - MOVE_FORWARD, TURN_RIGHT, TURN_LEFT,
  - MOVE_RANDOM, GRAB_FOOD,
  - UNCONDITIONAL_DROP_PHEROMONE, NO_ACTION

- **Parameters**
  - Population size: $M = 64,000$
  - Maximum number of generations: $G = 100$

---

Multi-Agent Strategies: Results

- The best individual of the run appeared in generation 90, had a fitness value of 7.4, and scored 144 hits.

---

Simulated Robotic Soccer

[Cho and Zhang, 1998]

- **Environment for Dash-and-Dribble Behavior**
  - 22×14 grid soccer field
  - a ball and a target position
  - 4 offensive robots (moving in 8 directions)
  - 11 opponent robots (obstacles)

---

Robot Soccer: Fitness Function

- **For Dashing behavior to the ball**
  $$f_i = \sum_{r=1}^{4} \{ c_1 \text{ max}(X_r - X_t, Y_r - Y_t) + c_2 S_r + c_3 C_r - c_4 M_r + K \}$$

- **For Dribbling behavior to the target position**
  $$f_i = \sum_{r=1}^{4} \{ c_1 \text{ max}(X_r - X_t, Y_r - Y_t) + c_2 S_r + c_3 C_r - c_4 M_r + c_5 A_r + K \}$$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_r$</td>
<td>$x$-axis distance between target and robot $r$</td>
</tr>
<tr>
<td>$Y_r$</td>
<td>$y$-axis distance between target and robot $r$</td>
</tr>
<tr>
<td>$S_r$</td>
<td>number of steps moved by robot $r$</td>
</tr>
<tr>
<td>$C_r$</td>
<td>number of collisions made by robot $r$</td>
</tr>
<tr>
<td>$M_r$</td>
<td>distance between starting and final position of robot $r$</td>
</tr>
<tr>
<td>$A_r$</td>
<td>penalty for moving away from other robots</td>
</tr>
<tr>
<td>$c_i$</td>
<td>coefficient for factor $i$</td>
</tr>
<tr>
<td>$K$</td>
<td>positive constant</td>
</tr>
</tbody>
</table>
Robot Soccer: Experimental Setup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal set</td>
<td>FORWARD, AVOID, RANDOM-MOVE, STOP, TURN-TARGET TURN-BALL</td>
</tr>
<tr>
<td>Function set</td>
<td>IF-BALL, IF-ROBOT, IF-TARGET, IF-OPPONENT, PROG2, PROG3</td>
</tr>
<tr>
<td>Fitness cases</td>
<td>20 training worlds, 20 test worlds</td>
</tr>
<tr>
<td>Robot world</td>
<td>32 by 32 grid, 64 obstacles, 1 ball to dribble</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Max generation</td>
<td>200</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>1.0</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.1</td>
</tr>
<tr>
<td>Max tree depth</td>
<td>10</td>
</tr>
<tr>
<td>Selection scheme</td>
<td>truncation selection with elitism</td>
</tr>
</tbody>
</table>

Robot Soccer: Cooperative Behaviors of Robots

Time Series Prediction

[Oakley, 1996]

- **Given:** \(\tau\) previous values in a time series

\[ x(t) = (x(t), x(t-1), \ldots, x(t-\tau)) \]

- **Find:** a function \(f\) which predicts the next value of the series

\[ x(t+1) = f(x(t)) = f(x(t), x(t-1), \ldots, x(t-\tau)) \]

- **Examples:** Logistic map, sun-spots, stock price index, currency exchange rate

Time Series Prediction: Example

- The Mackey-Glass delay differential series

\[ \frac{dx_t}{dt} = \frac{bx_{t-\Delta}}{1 + x_{t-\Delta}^c} - ax_t \]

- \(a = 0.1, b = 0.2, c = 10.0, \) and \(\Delta = 30.0\)
Time Series Prediction: Experimental Setup

Objective: Predict next 65 points at 5 places in series

Terminal set: Embedded data at \( t = 1, 2, 3, 4, 5, 6, 11, 16, 21, 31, R \)

Function set: +, -, %, *

Fitness cases: Actual members of the data series

Raw fitness: Sum over the 325 fitness cases of squared error between predicted and actual points

Standardized fitness: Same as raw fitness

Hits: Predicted and actual points are within 0.001 of each other

Wrapper: None

Parameters:
- \( M = 500, G = 51 \)
- Max. depth of new individuals: 6
- Max. depth of new subtrees for mutants: 4
- Max. depth of individuals after crossover: 17
- Fitness-proportionate reproduction fraction: 0.1
- Crossover at any point fraction: 0.2
- Crossover at function points fraction: 0.7
- Selection method: Fitness-proportionate (by normalized fitness)
- Generation method: Ramped half-and-half

Series: Mackey-Glass

Mean generations to fittest: 13.38
Std. dev. of generation: 16.46
Median generations: 5
No. of generations ≥ 25: 10
No. of generations ≥ 40: 9
Mean best fitness: 10.22
Std. dev. of best fitness: 3.371
Median best fitness: 10.51
No. of duplicate fitnesses: 3
Overall best fitness: 3.851
Typical linear fitness: 11.44
Mean left parentheses: 9.120
Std. dev. of left parens.: 17.64
Median left parens.: 3
No. left parens. ≥ 10: 8
No. left parens. ≥ 20: 6

Summary of the fittest S-expressions

Circuit Design [Koza et al., 1997]

Circuit Design: Functions

- **Component-creating functions**
  - Resistor R, capacitor C, inductor L
  - Diode D, transistor QT0,
  - Logical AND0 function

- **Connection-creating functions**
  - SERIES division function
  - PSS and PSL parallel division function
  - STAR1 division function
  - VIA0 function
Circuit Design: Fitness Evaluation

- Program Tree
- Embryonic Circuit
- Fully Designed Circuit (NetGraph)
- Circuit Netlist (ascii)
- Circuit Simulator (SPICE)
- Circuit Behavior (Output)
- Fitness

Evolving Neural Networks

- [Zhang et al., 1993, 1995]
- Genetic operators are used to adapt:
  - Connection weights
  - Network topology
  - Network size
  - Neuron types
- using the neural tree representation scheme

Evolving Neural Networks: Method

- Generate $M$ Networks
- Evaluate Fitness of Nets
  - Acceptable Net Found?
    - Yes: STOP
    - No: Select Fitter Networks
- Create $M$ New Networks

Evolving Neural Networks: Neural Tree Representation

- Neural trees are used as genotype for the evolution of neural networks.
- **Nonterminal** nodes: neural units
- **Terminal** nodes: input units
- **Root** node: output unit
- **Links**: connection weights $w_{ij}$ from $j$ to $i$
- **Layer** of node $i$: path length of the longest path to a terminal node of the substrees of $i$. 
Evolving Neural Networks: A Neural Tree

- **Expressiveness**: arbitrary feedforward networks of heterogeneous neurons can be represented by neural trees.
- **Parsimony**: sparse networks with partial connectivity
- **En/decoding**: genotype and phenotype equivalent in functionality
- **Examples**: sigma-pi neural networks.

Evolving Neural Trees: Structural Adaptation by Crossover

- Neuron type, topology, size and shape of networks are adapted by crossover.

Evolutionary Neural Trees: Results for Mackey-Glass Time Series

\[
\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1 + x^{10}(t-\tau)} - bx(t)
\]

\[x(t+10) = \rho(x(t), x(t+1), \ldots, x(t+9))\]
Evolutionary Neural Trees: Results for Mackey-Glass Data

Evolutionary Neural Trees: Neural Trees Evolved

Evolutionary Neural Trees: Comparison to Back-Propagation Networks

<table>
<thead>
<tr>
<th>Method</th>
<th>Hidden Units</th>
<th>Num. Weights</th>
<th>Training Error</th>
<th>Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural trees</td>
<td>30</td>
<td>153</td>
<td>0.52</td>
<td>0.58</td>
</tr>
<tr>
<td>Backpropagation 1</td>
<td>100</td>
<td>601</td>
<td>0.53</td>
<td>0.56</td>
</tr>
<tr>
<td>Backpropagation 2</td>
<td>300</td>
<td>1801</td>
<td>0.69</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Evolutionary Neural Trees: Results for Far-Infrared NH₃ Laser
Evolutionary Neural Trees: Performance for Test Data

Target Function

Difference between True Values and Predicted Values for the Test Data

Advanced Topics

Variants of Genetic Programming

- Stack-based GP
- Strongly-typed GP
- Linear GP
- Ontogenetic GP
- Cellular GP
- Breeder GP

Stack-Based GP

[Perkis, 1994]

\[ Y \ cos \ 5 \ * \ X \ X \ Y \ - \ / \ + \]

\[ Y \xrightarrow{\cos} \xrightarrow{5} \xrightarrow{*} \xrightarrow{X} \xrightarrow{X} \xrightarrow{Y} \xrightarrow{-} \xrightarrow{/} \xrightarrow{+} \]
Strongly-Typed GP
[Montana, 1995]

- STGP = Strongly Typed Genetic Programming
- Motivation
  - Don't create and evaluate trees that are syntactically illegal (or at least silly) with respect to the data.
  - Provide a good way to specify constraints from the input space.
- STGP only really makes sense if the input data is typical
- Mutation and Crossover must now respect the type constraints.
- *Generic* functions: Argument types determine return type
- *Generic* data-types: e.g. “List-of-?” where “?” is instantiated at runtime.

Linear GP [Nordin and Banzhaf, 1993]

- **Tree-based Genome**
  - a = a + x
  - b = a + c
  - c = b + 6
  - a = b + 7

- **Linear Genome**
  - 01100
  - 00101
  - 11100
  - 11110
  - 00111
  - 11011
  - 10101
  - 00111
  - 01011
  - 11001
  - 10100

Linear GP: Binary GP & Compiling GP

- **Genotype-Phenotype Mapping**
- **Compilation**
- **Binary GP**
- **Compiling GP**
- **Executable Code**

Linear GP: Crossover in CGP

- **Parents**
- **Children**
- **Save**
- **Restore**
- **Return**
- **Children**
- **Save**
- **Restore**
- **Return**
**Linear GP: CGP Crossover in Bitstrings**

- Crossover between instructions
  - crossover point
  - 32 bit instruction
  - Op-code   Operand1   Operand2
  - 0111001001 0001001011 1011101010

- Crossover within instructions
  - crossover point
  - 32 bit instruction
  - = protected field
  - Register address allowed?
  - Constant value allowed
  - Op-code allowed?

---

**Linear GP: Mutation in CGP**

- Mutation in operands
  - point mutation
  - 32 bit instruction
  - Op-code   Operand1   Operand2
  - 0111001001 0001001011 1011101010

- Mutation in op-code
  - point mutation
  - 32 bit instruction
  - Op-code   Operand1   Operand2
  - 0111001001 0001001011 1011101010

---

**Ontogenetic GP**

[Spector and Stoffel, 1996]

- Phylogeny: Development of a population over evolutionary time
- Ontogeny: Development of an individual over its lifetime
- Linear genome of GP terminals and non-terminals
- Addition of ontogenetic operators
  - *segment-copy* copies part of the linear program over another part of the program
  - *shift-left* rotates the program to the left
  - *shift-right* rotates the program to the right

**Cellular GP**

[Gruau, 1992]

- GP trees (genotype) are used to construct neural networks (phenotype).
- The fitness of the genotype is measured through the performance of the phenotype on the desired task.

---

Push-x % - shift-left push-x noop * * dup % - + push-x % dup % shift-right dup shift-left push-x shift-right * + shift-right - - push-x
Breeder GP (BGP) [Zhang and Muehlenbein, 1993, 1995]

ES (real-vector)  GA (bitstring)  GP (tree)

Muehlenbein et al. (1993)

Breeder GA (BGA) (real-vector + bitstring)

Zhang et al. (1993)

Breeder GP (BGP) (tree + real-vector + bitstring)

Breeder GP: Motivation for GP Theory

- In GP, parse trees of Lisp-like programs are used as chromosomes.
- Performance of programs are evaluated by training error and the program size tends to grow as training error decreases.
- Eventual goal of learning is to get small generalization error and the generalization error tends to increase as program size grows.
- How to control the program growth?

Breeder GP: MDL-Based Fitness Functions

\[ F(A \mid D) = F_D + F_A = \beta E(D \mid A) + \alpha C(A) \]

- \( F_D \): Training error of program \( A \) for data set \( D \)
- \( F_A \): Structural complexity of program \( A \)
- \( \alpha, \beta \): Relative importance to be controlled

Breeder GP: Adaptive Occam Method [Zhang et al., 1995]

\[ F_i(t) = E_i(t) + \alpha(t)C_i(t) \]

\[ \alpha(t) = \begin{cases} 
N^{-2} \frac{E_{\text{best}}(t-1)}{C_{\text{best}}(t)} & \text{if } E_{\text{best}}(t-1) > \varepsilon \\
1 & \text{otherwise} 
\end{cases} \]

- \( E_{\text{best}}(t-1) \): Training error of best progr. at gen \( t-1 \)
- \( C_{\text{best}}(t) \): Complexity of best progr. at gen. \( t \)
- \( \varepsilon \): Desired performance level in error
Promising Application Areas

- Problem areas where a good approximate solution (but not necessarily optimal solution) is satisfactory (e.g., AI and AL applications).
- Problem areas where discovery of functional structure (as opposed to parameter estimation) is a major part of the problem (e.g., symbolic regression).
- Problem areas involving many variables whose inter-relationship is not well understood (e.g., structural design).

- Problem areas where data are observable but underlying structure is not known (e.g., discovery of rules in data).
- Problem areas where primitive functions can be guessed but their combinations are not well understood (e.g., circuit design).
- Problem areas where programming by hand is difficult (e.g., multi-agent strategies)

Research Issues (1/3)

- Speed-up methods for GP runs
  - Parallel implementation of GP [Koza et al. 96] [Stoffel & Spector 96]
  - Training subset selection [Gathercole & Ross 97] [Zhang & Cho 98] [Zhang & Joung 98]
- Issues of introns and program growth control
  - Introns and bloat [Langdon 97] [Rosca & Ballard 97] [Soule & Foster 97] [Banzhaf 97]
  - Fixed complexity penalty [Iba et al. 94] [Rosca et al. 97]
  - Adaptive Occam method for controlling bloat [Zhang & Muehlenbein 93, 95]
Research Issues (2/3)

- Finding and exploiting parameterizable submodules
  - ADF [Koza 94] [O’Reilly 96]
  - GLiB [Angeline 93]
  - AR [Rosca 94], ARL [Rosca & Ballard 96]
  - Libraries [Teller & Veloso 95] [Zhang et al. 97]
  - ADM [Spector 96]
  - Architecture Altering Operations [Koza 95]

Research Issues (3/3)

- Intelligent crossover and mutation [Luke and Spector 97] [Angeline 97] [Poli and Langdon 98]
- Handling vectors and complex data structures [Langdon 98]
- Automatic setup of GP parameters [Angeline 96]
- Employing more general program constructs, such as recursion, iteration, and internal states.

Further Information

Web Sites and E-mail Lists

- Web sites
  - Genetic programming home page: http://www.genetic-programming.org/
- Genetic programming (GP) list
  - To subscribe, send e-mail message to: Genetic-Programming-Request@CS.Stanford.Edu
    - The body of the message must consist of exactly the words: subscribe genetic-programming
- GP bibliography
  - William Langdon of the University of Birmingham maintains a bibliography on GP at http://www.cs.bham.ac.uk/~wbl
Upcoming GP-Related Conferences

- Genetic and Evolutionary Computation Conference (GECCO-99)
  - [http://www-illigal.ge.uiuc.edu/gecco/](http://www-illigal.ge.uiuc.edu/gecco/)
- Second European Conference on Genetic Programming (EuroGP-99)
- IEEE Congress on Evolutionary Computation (CEC-99)
  - [http://garage.cps.msu.edu/cec99/](http://garage.cps.msu.edu/cec99/)

Texts on GP


GP Conference/Workshop Proceedings

- Proceedings of GP Conferences

- Proceedings of EuroGP Workshops

AiGP Series and Journals

- Advances in Genetic Programming (AiGP) Series

- Selected journals for GP and EC in general
  - Genetic Programming and Evolvable Machines, Kluwer (in preparation)
  - Evolutionary Computation, MIT Press.