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

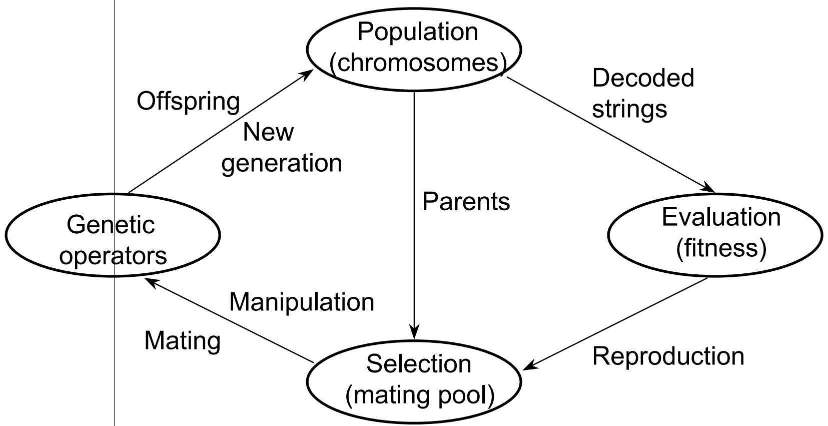
Evolutionary Algorithms (EAs)

- 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 selelection
- 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)

  1 0 0 0 0 1 0 0 ⟶ 1 0 0 0 0 1 0 0

  1 0 0 0 0 1 0 0

- **Crossover:** exchange subparts of two chromosomes

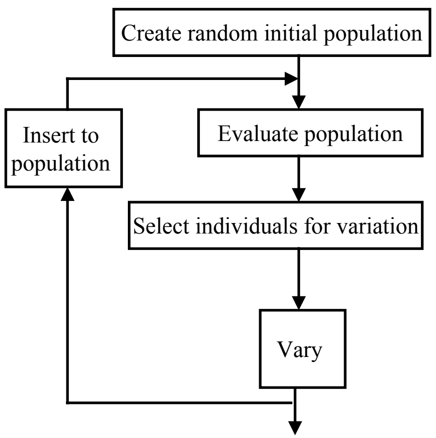
  1 0 0 | 0 0 1 0 0 ⟶ 1 0 0 1 1 1 1 1

  1 1 1 | 1 1 1 1 1 ⟶ 1 1 1 0 0 1 0 0

- **Mutation:** randomly flip some bits

  0 0 0 0 0 1 0 0 ⟶ 0 0 0 0 0 0 0 0

Selection



Selection Schemes

- Proportionate selection
  - Reproduce offspring in proportion to fitness *fi.*
- Ranking selection
  - Select individuals according to *rank*(*fi*).
- 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

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

$m(H,t)$ Number of members of *H*

$p_c\ ,p_m$ 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)

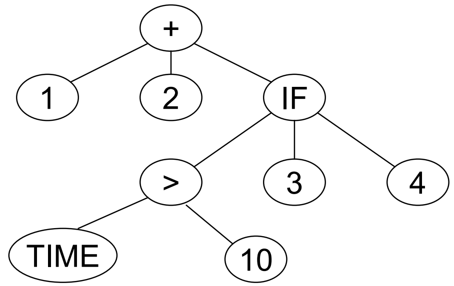
Genetic Programming (GP)

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}

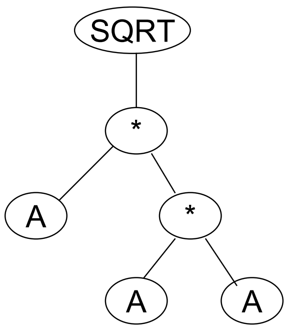


A GP Tree for Kepler's Law

- GP-tree representation of Kepler's third law: $P^2 = cA^3$

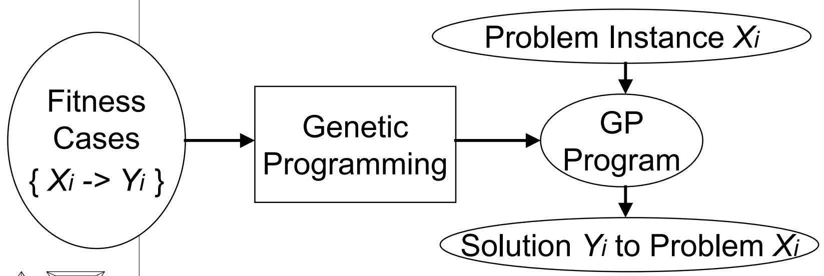
```
PROGRAM ORBITAL_PERIOD
C          # Mars #
           A = 1.52
           P = SQRT(A * A * A)
           PRINT P
END ORBITAL_PERIORD
```

```
(defun orbital_period ()
           ; Mars ;
           (setf A 1.52)
           (sqrt (* A (* A A))))
```



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.



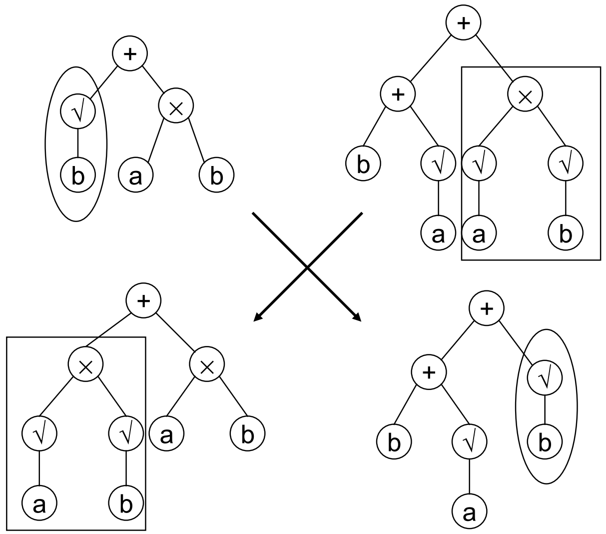
Setting Up for a GP Run

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.

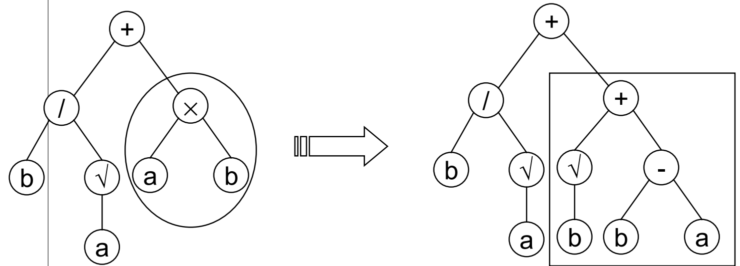
Genetic Programming Procedure

1. Choose a set of possible functions and terminals for the program: $F = \{+, -, *, /, \surd\}$, $T = \{A\}$.
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

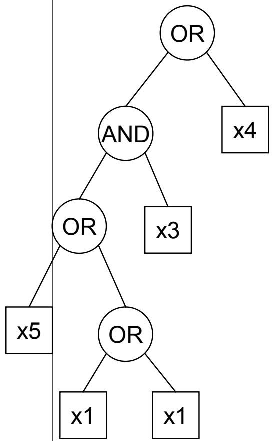


Example GP Run: Majority

- Problem: Given five binary inputs x1, x2, …, x5, return y = 1 if three or more of xi are 1 and output y=0 otherwise.
- Fitness cases given (20 out of 32):

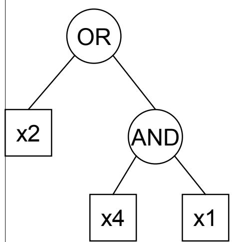
```
x1 x2 x3 x4  x5 ==> y
-------------------
 0  0  0  0  0  ==> 0
 0  0  0  0  1  ==> 0
 0  0  1  0  1  ==> 0
 0  0  1  1  0  ==> 0
 0  0  1  1  1  ==> 1
          …..
 1  1  0  0  0  ==> 0
 1  1  0  0  1  ==> 1
 1  1  1  0  1  ==> 1
```


Majority: Best Program at Generation 0



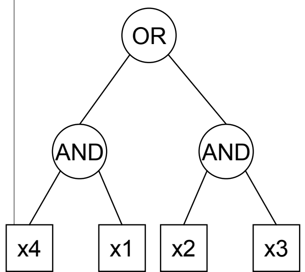
- Training error = 4/20
- Generalization error = 8/32

Majority: Best Program at Generation 1



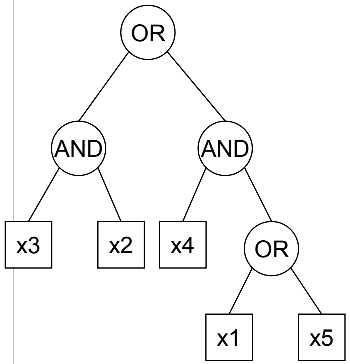
- Training error = 3/20
- Generalization error = 8/32

Majority: Best Program at Generation 11



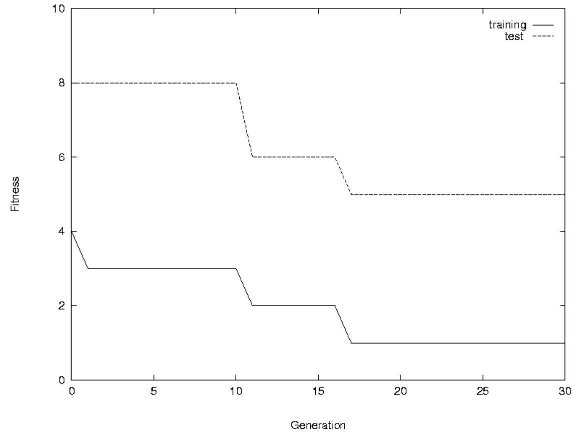
- Training error = 2/20
- Generalization error = 6/32

Majority: Best Program at Generation 17



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

Majority: Evolution of Fitness Values



Genetic Programming Applications

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

$$D = \{(x_i, y_i) \mid i=1,\dots,N\}$$

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]

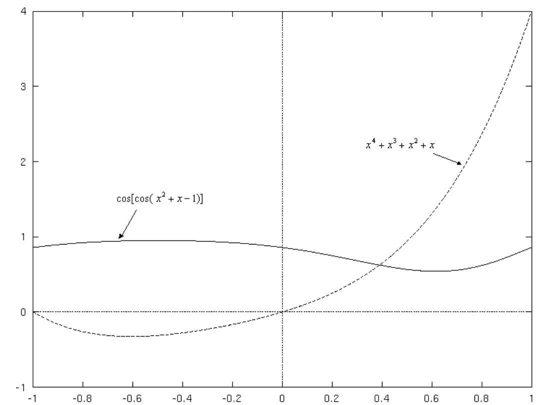
Independent Variable X	Dependent Variable Y
-1.0	0.0
-0.9	-0.1629
-0.8	-0.2624
-0.7	-0.3129
-0.6	-0.3264
-0.5	-0.3125
-0.4	-0.2784
-0.3	-0.2289
-0.2	-0.1664
-0.1	-0.0909
0	0.0
0.1	0.1111
0.2	0.2494
0.3	0.4251
0.4	0.6496
0.5	0.9375
0.6	1.3056
0.7	1.7731
0.8	2.3616
0.9	3.0951
1.0	4.0000

Symbolic Regression: Experimental Setup

Objective:	Find a function of one independent variable, in symbolic form, that fit a given sample of 20 (x_i, y_i) 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.

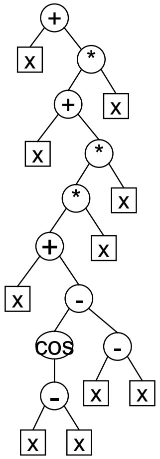
Symbolic Regression: Generation 0

- Median individual with raw fitness of 23.67
  - (COS (COS (+ (- (* x x) (% x x)) x)))



Symbolic Regression: Generation 34

- Best-of-run individual with raw fitness of 0.00 (100% correct)
  - `(+ x (* (+ x (* (* (+ x (- (COS (- x x) ) (- x x) ) ) x) x) ) x) )`
  - Equivalent to $x^4 + x^3 + x^2 + x$

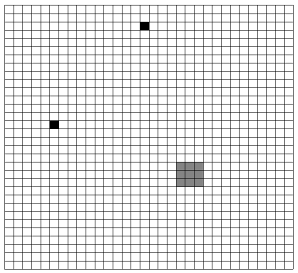


Symbolic Regression: Observations

- 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, RLOG`, and (effectively) `RCOS` are all absent in the best individual of generation 34.

Multi-Agent Strategies [Benenett III, 1996]

- The Foraging Problem
  - 32×32 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

$$\frac{\sum^{n}(t_{food} * f_{food}) + \sum^{m}(t_{max} * f_{max} * d_{food})}{1,000,000}$$

n = Number of food pellets transported to the nest

t_{food} = Number of time steps elapsed when the food pellet arrived at the nest

f_{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_{max} = Maximum allotted time step = 4,000

f_{max} = Maximum possible value of f_{food} = 400,000

d_{food} = Manhattan distance between food pellet and nest

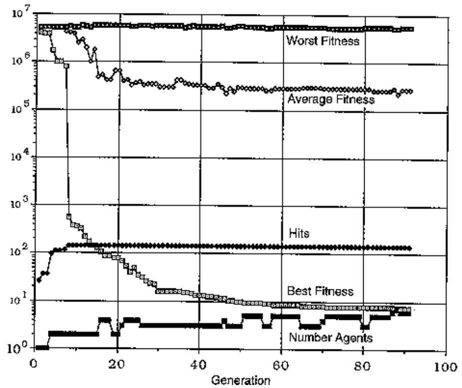
p_{max} = Maximum number of points per agent = 100

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

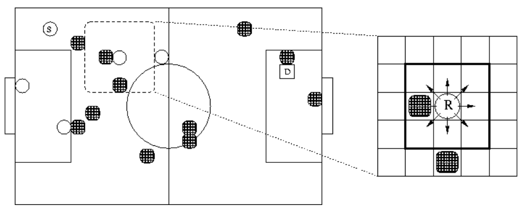
Multi-Agent Strategies:
Results

- 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

$$f_1 = \sum_{r=1}^{4} \{c_1 \max(X_r, Y_r) + c_2 S_r + c_3 C_r - c_4 M_r + K\}$$

- For Dribbling behavior to the target position

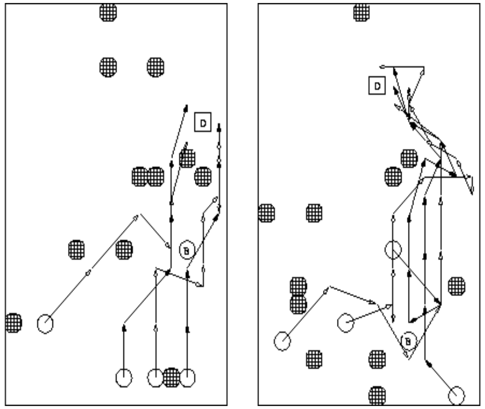
$$f_2 = \sum_{r=1}^{4} \{c_1 \max(X_r, Y_r) + c_2 S_r + c_3 C_r - c_4 M_r + c_5 A_r + K\}$$

Symbol	Description
X_r	*x*-axis distance between target and robot *r*
Y_r	*y*-axis distance between target and robot *r*
S_r	number of steps moved by robot *r*
C_r	number of collisions made by robot *r*
M_r	distance between starting and final position of robot *r*
A_r	penalty for moving away from other robots
c_i	coefficient for factor *i*
K	positive constant

Robot Soccer: Experimental Setup

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

Robot Soccer: Cooperative Behaviors of Robots



Training case Test case

Time Series Prediction

[Oakley, 1996]

- **Given**: τ previous values in a time series

$$\mathbf{x}(t) = (x(t), x(t-1), \ldots, x(t-\tau))$$

Find: a function f which predicts the next value of the series

$$x(t+1) = f(\mathbf{x}(t)) = f(x(t), x(t-1), \ldots, x(t-\tau))$$

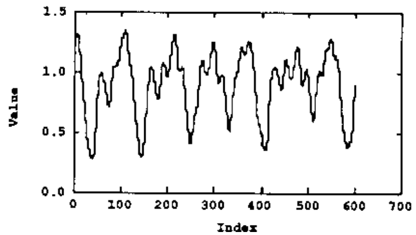
- Examples: Logistic map, sun-spots, stock price index, currency exchange rate

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 Δ = 30.0

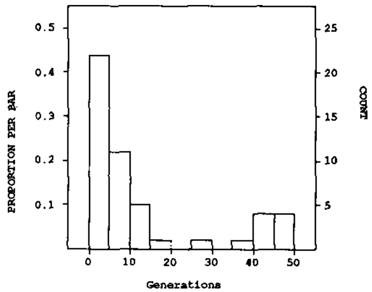


Time Series Prediction:
Experimental Setup

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
Success predicate	None
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

Time Series Prediction: Results

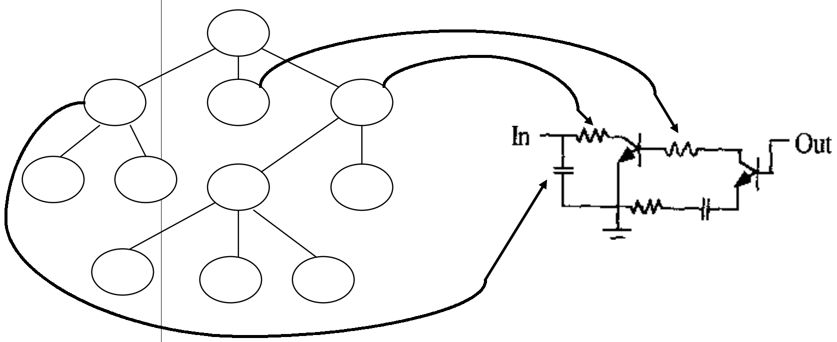
Frequency distribution of generations at which fittest S-expression was found:



Summary of the fittest S-expressions

Series	Mackey-Glass
Number	50
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

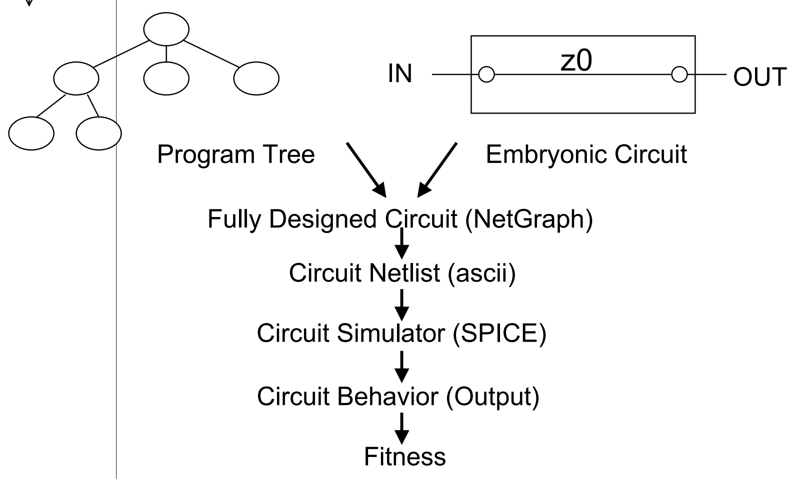
Circuit Design [Koza et al., 1997]



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



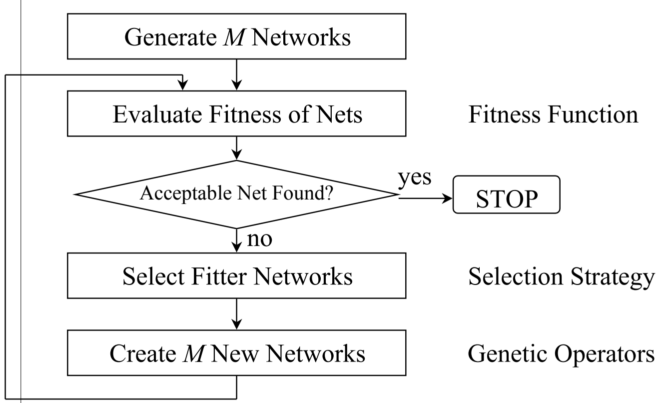
Evolving Neural Networks

[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

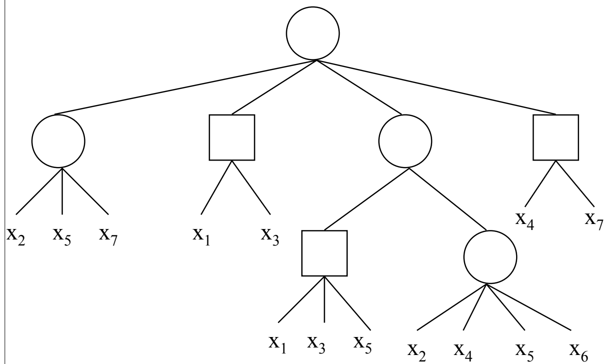


Evolving Neural Networks: Neural Tree Representation

- 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



Evolving Neural Networks:

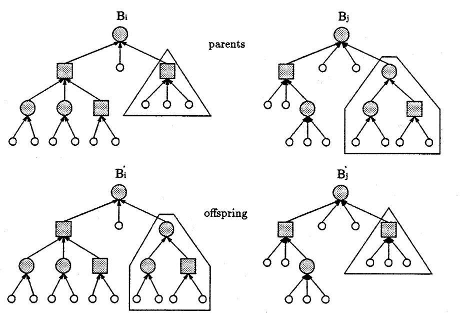
Features of Neural Trees

- **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:

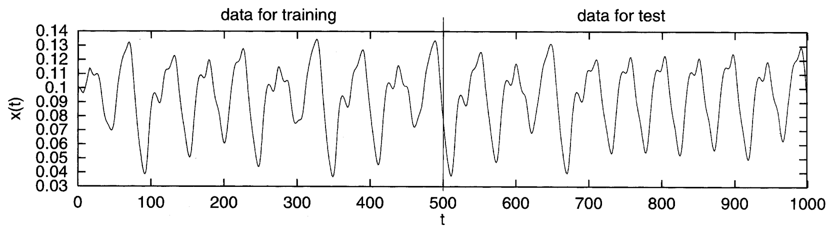
Structural Adaptation by Crossover

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



Evolutionary Neural Trees:

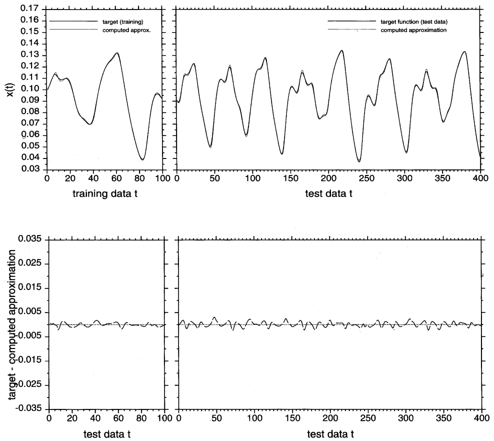
Results for Mackey-Glass Time Series



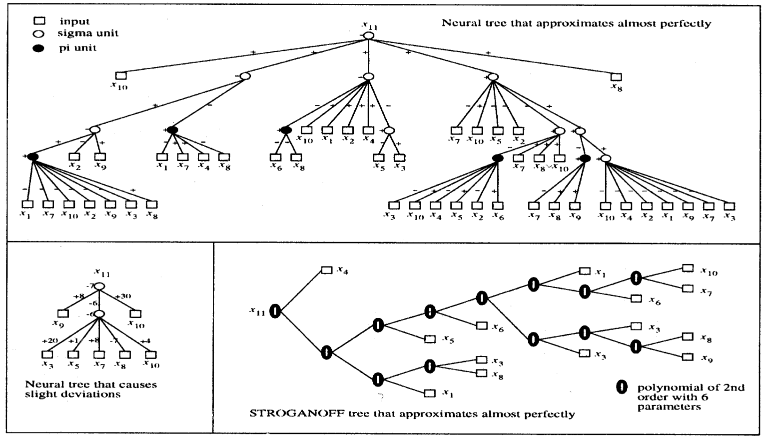
$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t)$$

$$x(t+10) = \rho(x(t), x(t+1), \ldots, x(t+9))$$

Evolutionary Neural Trees:
Results for Mackey-Glass Data



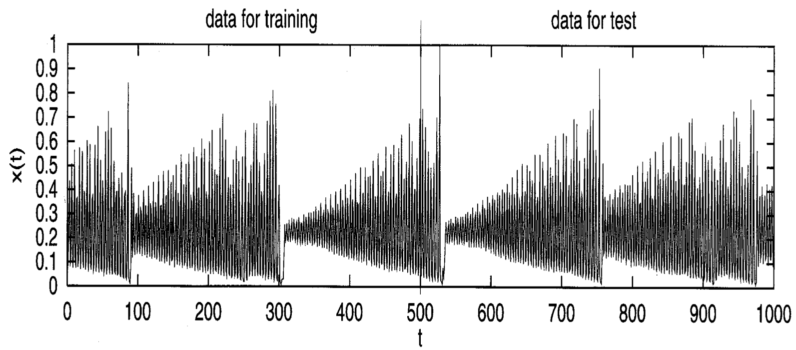
Evolutionary Neural Trees:
Neural Trees Evolved



Evolutionary Neural Trees:
Comparison to Back-Propagation Networks

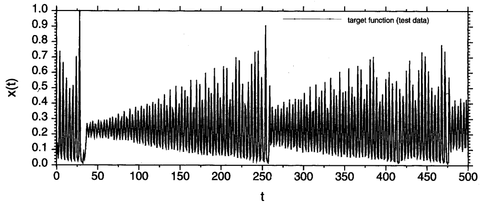
Method	Hidden Units	Num. Weights	Training Error	Prediction Error
Neural trees	30	153	0.52	0.58
Backpropagation 1	100	601	0.53	0.56
Backpropagation 2	300	1801	0.69	0.84

Evolutionary Neural Trees:
Results for Far-Infrared NH_3 Laser

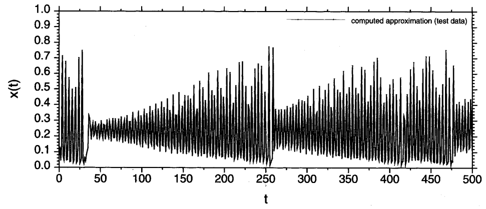


Evolutionary Neural Trees:

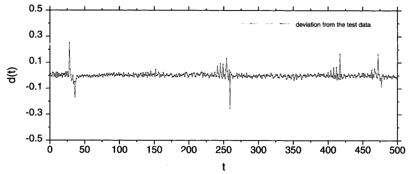
Performance for Test Data



Target Function



Computed Approximation



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]

- Expressions in trees can be rewritten in postfix notation.

Y COS 5 * X X Y - / +

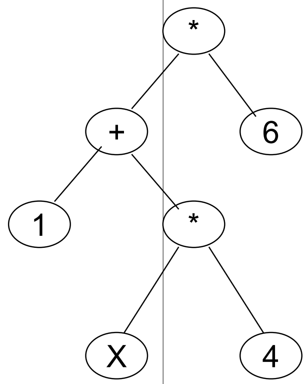
Y → COS → 5 → * → X → X → Y → - → / → +

Strongly-Typed GP
[Montana, 1995]

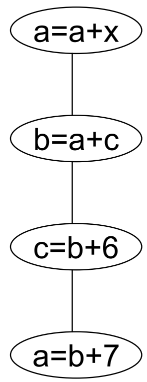
- STGP = Strongly Typed Genetic Programming
- Motivation
  - Don't create and evaluate trees that are syntactically illegal (or at least silly) with respect to the data.
  - Provide a good way to specify constraints from the input space.
- STGP only really makes sense if the input data is typical
- Mutation 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

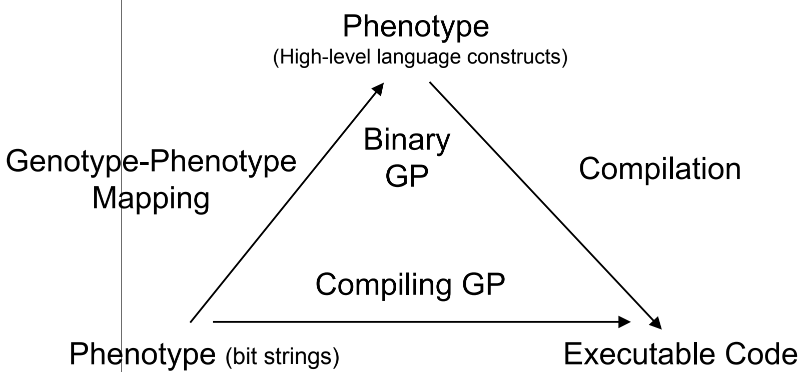


Linear Genome

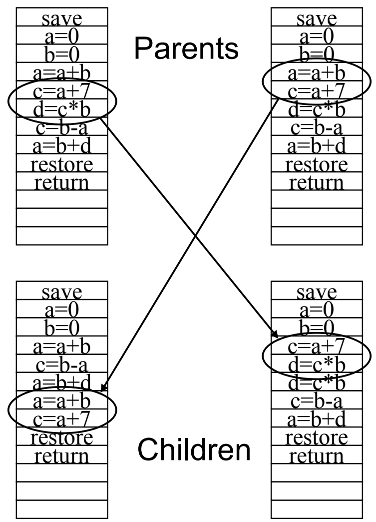


01100
00101
11100
11110
00111
11011
10101
00111
01011
11001
10100

Linear GP: Binary GP & Compiling GP



Linear GP: Crossover in CGP



Linear GP: CGP Crossover in Bitstrings

- Crossover between instructions

crossover point

32 bit instruction

01110010011000100101111011101010|01110110011000100101011011101010|01110011011000100101011011101010 1

- Crossover within instructions

crossover point

= protected field

32 bit instruction

01110010011000100101111011101010|01110110011000100101011011101010|011100110110001001010110111010101

Linear GP: Mutation in CGP

- Mutation in operands

point mutation

0111001001	10001001011	10111010101
Op-code	Operand1	Operand2

32 bit instruction

- Mutation in op-code

point mutation

0111001001	10001001011	10111010101
Op-code	Operand1	Operand2

32 bit instruction

Register address allowed?
Constant value allowed
Op-code allowed?

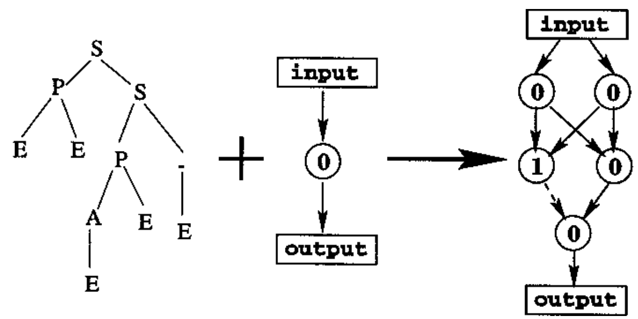
Ontogenetic GP
[Spector and Stoffel, 1996]

- 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

Push-x % - shift-left push-x noop * * dup % - + push-x % dup %
shift-right dup shift-left push-x shift-right * + shift-right - - push-x

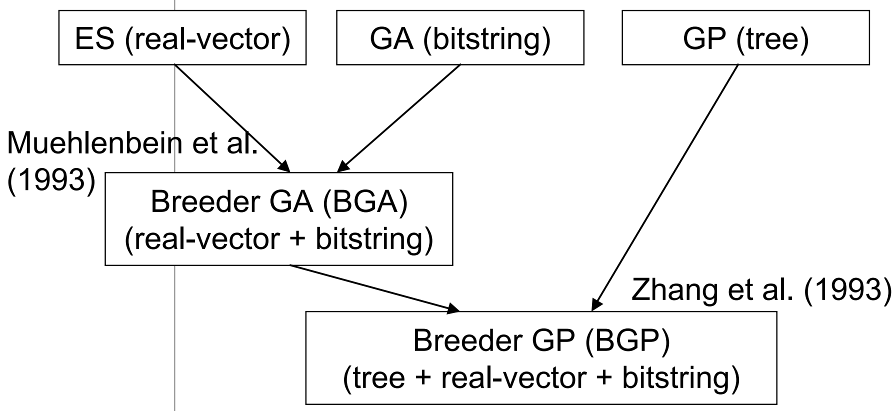
Cellular GP [Gruau, 1992]

- 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.



Breeder GP (BGP)

[Zhang and Muehlenbein, 1993, 1995]



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

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

$E(D \mid A)$ Training error of program *A* for data set *D*

$C(A)$ Structural complexity of program *A*

α, β Relative importance to be controlled

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

$$F_i(t) = E_i(t) + \alpha(t) C_i(t)$$

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

ε Desired performance level in error

$E_{best}(t-1)$ Training error of best progr. at gen *t-1*

$C_{best}(t)$ Complexity of best progr. at gen. *t*

Guidelines

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 apposed 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).

Promising Application Areas Cont'd

- 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 bolat [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/
- Second European Conference on Genetic Programming (EuroGP-99)
  - http://www.cs.bham.ac.uk/~rmp/eebic/eurogp99
- IEEE Congress on Evolutionary Computation (CEC-99)
  - http://garage.cps.msu.edu/cec99/

Texts on GP

- *Genetic Programming: On Programming Computers by Means of Natural Selection,* John Koza, MIT Press, 1992.
- *Genetic Programming II: Automatic Discovery of Reusable Programs,* John Koza, MIT Press, 1994.
- *Genetic Programming: An Introduction*, Wolfgang Banzhaf et al., Morgan Kaufmann Publishers, 1998.
- *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming*, William B. Langdon, Kluwer, 1998.

GP Conference/Workshop Proceedings

- Proceedings of GP Conferences
  - J. Koza et al. (Eds.) *Genetic Programming 1996: Proceedings of the First Annual Conference,* July 28-31, 1996, Stanford University, MIT Press, 1996.
  - J. Koza et al. (Eds.) *Genetic Programming 1997: Proceedings of the Second Annual Conference,* July 13-16, 1997, Stanford University, Morgan Kaufmann, 1997.
  - J. Koza et al. (Eds.) *Genetic Programming 1998: Proceedings of the Third Annual Conference,* July 22-25, 1998, University of Wisconsin, Madison, Morgan Kaufmann, 1998.
- Proceedings of EuroGP Workshops
  - W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Forgaty (Eds.) *Genetic Programming: First European Workshop. EuroGP'98,* April, 1998, Paris, France, Lecture Notes in Computer Science, Volume 1391, Springer-Verlag, 1998.

AiGP Series and Journals

- Advances in Genetic Programming (AiGP) Series
  - K. E. Kinnear Jr. (Ed.) *Advances in Genetic Programming*, MIT Press, 1994.
  - P. J. Angeline and K. E. Kinnear Jr. (Eds.) *Advances in Genetic Programming 2*, MIT Press, 1996.
  - L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. Angeline (Eds.) *Advances in Genetic Programming 3*, MIT Press, 1999.
- Selected journals for GP and EC in general
  - *Genetic Programming and Evolvable Machines*, Kluwer (in preparation)
  - *Evolutionary Computation*, MIT Press.
  - *IEEE Transactions on Evolutionary Computation*, IEEE Press.