MIS 214 / CS 274 LECTURE

GENETIC ALGORITHMS, GENETIC PROGRAMMING

JOHN KOZA

TUESDAY, APRIL 27, 1999

OUTLINE

- **Genetic Algorithm (GA)**
  - Flowchart, operations, preparatory steps
  - Examples
  - Characteristics, advice, common mistakes
  - Hillclimbing and Simulated Annealing
- **Genetic Programming (GP)**
  - Idea of automatic programming
  - Flowchart, operations, preparatory steps
  - Examples
  - Automatically defined functions (ADFs)
  - Architecture-altering operations based on gene duplication and gene deletion
  - Embryos and developmental GP
- **Sources of additional information**

"[ask] not what mathematics can do for biology, but what biology can do for mathematics."

– Stanislaw Ulam 1976

"...WHAT BIOLOGY CAN DO FORCOMPUTER SCIENCE..."

9 BIOLOGICAL IDEAS USED IN GENETIC ALGORITHMS (GA) AND GENETIC PROGRAMMING (GP)

- **The Darwinian principle of survival of the fittest**
- **asexual mutation operation**
- **sexual recombination (crossover) operation**
- **inversion operation**
- **gene regulation**
- **gene duplication**
- **gene deletion**
- **embryos**
- **development of embryo into organism**

PROMISING GA/GP APPLICATION AREAS

• Problem areas involving many variables that are interrelated in highly non-linear ways

• Problem areas involving many variables whose inter-relationship is not well understood

• Problem areas where a good approximate solution is satisfactory (and no one is expecting a perfect solution)

- design
- control
- classification, pattern recognition, inage processing
- forecasting
- model building and data mining

PROMISING GA/GP APPLICATION AREAS – CONTINUED

- **Problem areas where discovery of the size and shape of the solution is a major part of the problem**

- **Problem areas where large computerized databases are accumulating and computerized techniques are needed to analyze the data**
    - genome and protein sequences
    - satellite data
    - astronomy
    - petroleum
    - financial databases
    - marketing databases
    - World Wide Web

PROMISING GA/GP APPLICATION AREAS – CONTINUED

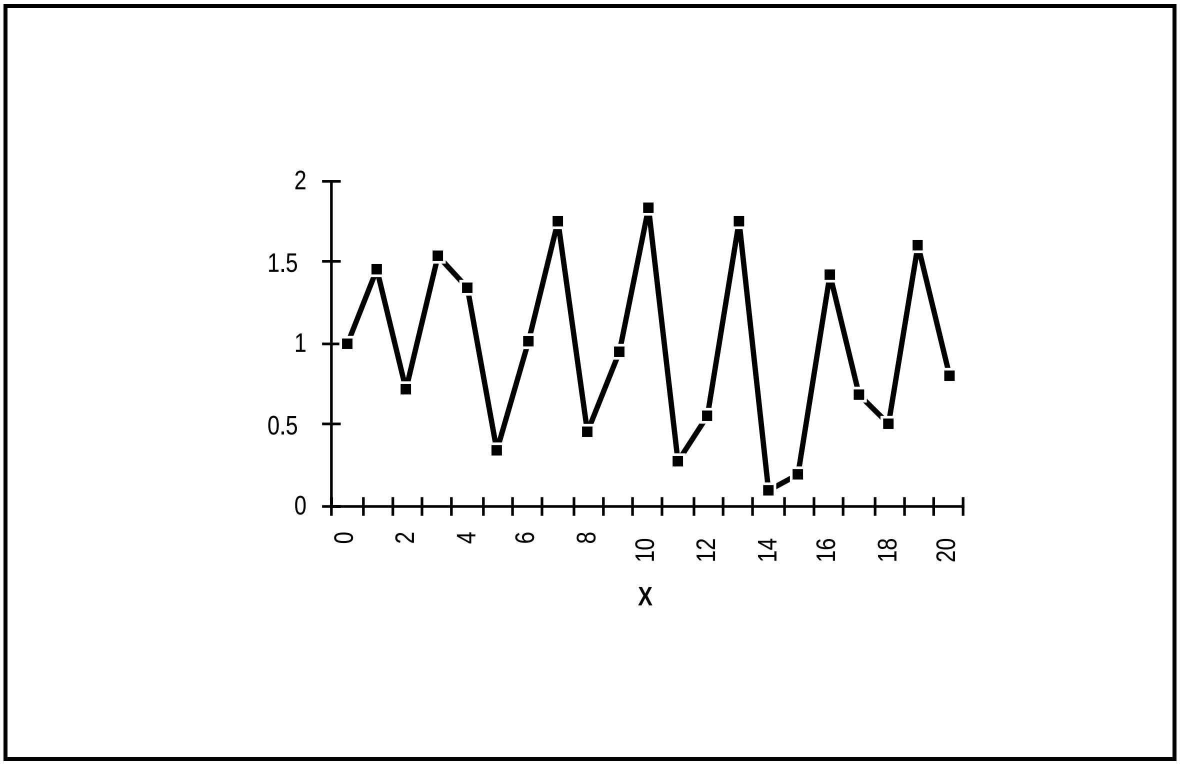
- **Problem areas for which humans find it very difficult to write good programs**
    - parallel computers
    - cellular automata
    - multi-agent strategies
    - distributed AI
    - FPGAs

SEARCH METHODS IN GENERAL

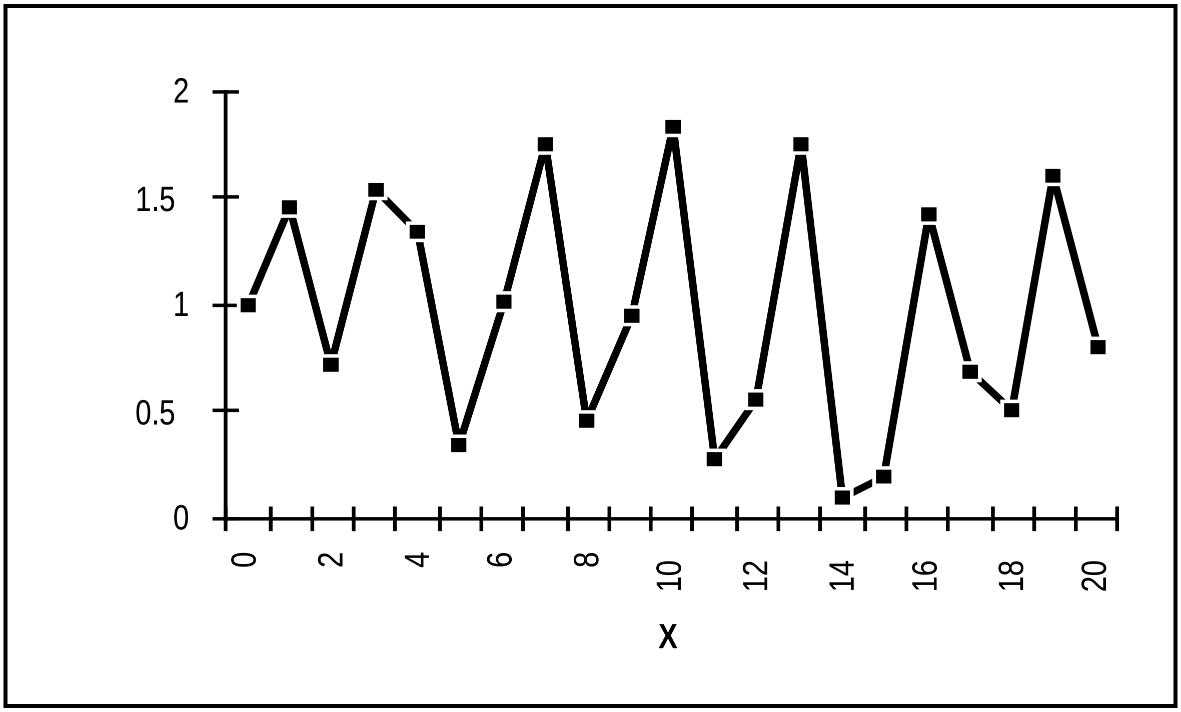
- INITIAL STRUCTURE (E.G., A POINT OR POINTS IN THE SEACH SPACE OF THE PROBLEM)
- FITNESS MEASURE
- METHOD OF CREATING NEW STRUCTURE
- PARAMETERS
- TERMINATION CRITERION AND METHOD OF DESIGNATING THE RESULT

9

SEARCHING A SPACE WITH ONE GLOBAL OPTIMUM POINT AND MANY LOCAL OPTIMA



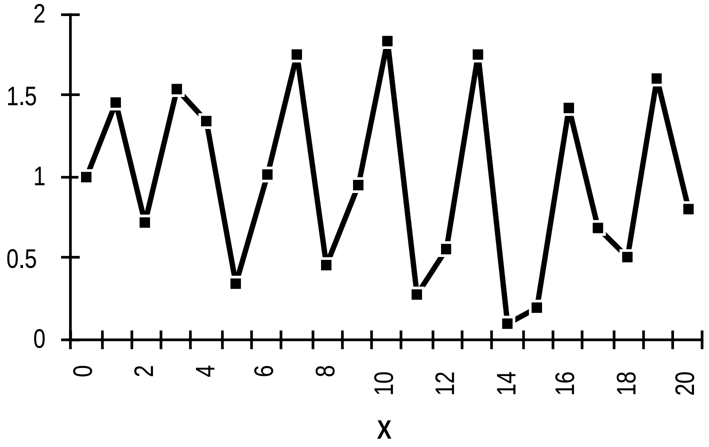
ENUMERATIVE RANDOM OR BLIND RANDOM SEARCH



NEITHER ENUMERATIVE RANDOM NOR BLIND RANDOM SEARCH USES ACQUIRED INFORMATION IN DIRECTING THE SEARCH

11

HILL CLIMBING



HILL CLIMBING — CONTINUED

- HILL CLIMBING AND GRADIENT DESCENT (ASCENT) USE ACQUIRED INFORMATION IN DIRECTING THE SEARCH

- HOWEVER, HILL CLIMBING AND GRADIENT DESCENT (ASCENT) ARE VERY PRONE TO GETTING TRAPPED ON LOCAL OPTIMA AND THEREBY MISSING THE GLOBAL OPTIMUM

- **If problem can be solved by hill-climbing, it is probably trivial to begin with.**

- **One broad approach to problem-solving is to recast original problem (e.g., “by changing the representation”) so that it becomes solvable by hill-climbing**

GIRAFFE

- **Long neck**
- **Long tongue**
- **Vegetable-digesting enzymes in stomach**
- **Long legs**
- **Brown coloration**

THE DESIGN

Neck length	Tongue length	Carnivorous?	Leg length	Coloration
15.11 feet	14 inches	No	9.96 feet	Brown
Numerical	Numerical	Boolean	Numerical	Categorical

CHARACTERISTICS

- **NUMERICAL VARIABLES – floating-point (such as 15.11 and 9.96) and integer (such as 14)**
- **BOOLEAN VARIABLES (two alternatives)**
- **CATEGORICAL VARIABLES (many alternatives)**

NON-LINEARITY

- **Taken one-by-one, the 5 design variables**
  - Long neck contributes negatively to fitness (requires considerable material to build, requires considerable energy to maintain, prone to injury, etc.)
  - Same for long tongue
- **Taken in pairs, the 10 possible pairs**
  - Long neck and long tongue - doubly detrimental
- **But, all 5 taken together are "co-adapted sets of alleles" and make a very fit animal for the jungle environment**

THE FALLACY OF HILL CLIMBING

Freely-made concession	Action
Of course, we all know that the variables are all inter-related in a highly non-linear way	Nonetheless, use hill climbing
Of course, we all know that hill climbing gets stuck on local optima (non-global) optima	Nonetheless, use hill climbing

THE GENETIC ALGORITHM (GA)

- **The *genetic algorithm* is a mathematical algorithm that transforms a set (population) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new set (new generation of the population) of offspring objects, using operations patterned after naturally-occurring genetic operations and the Darwinian principle of reproduction and survival of the fittest.**

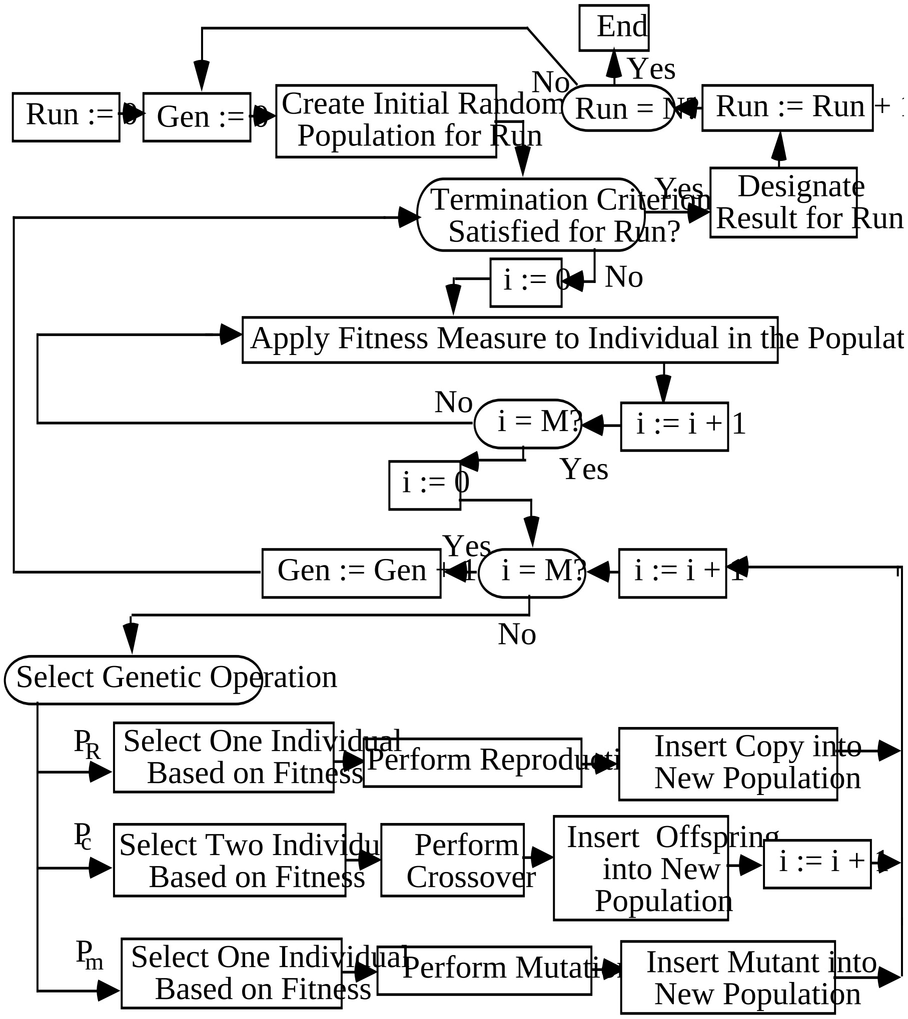
EXAMPLE

Generation 0			Generation 1
Individuals	Fitness		Offspring
in Population	Measure		Population
011	$3		111
001	$1	——>	010
110	$6		110
010	$2		010

GENETIC OPERATIONS USED IN THE BASIC GENETIC ALGORITHM

- **Darwinian reproduction**
- **Crossover (sexual recombination)**
- **Mutation (very occasional)**

FLOWCHART FOR THE BASIC GENETIC ALGORITHM


End
Yes
No
Run := 0
Gen := 0
Create Initial Random Population for Run
Run = N?
Run := Run + 1
Termination Criterion Satisfied for Run?
Yes
Designate Result for Run
i := 0
No
Apply Fitness Measure to Individual in the Populat
No
i = M?
i := i + 1
Yes
i := 0
Yes
Gen := Gen + 1
i = M?
i := i + 1
No
Select Genetic Operation
P_R
Select One Individual Based on Fitness
Perform Reproducti
Insert Copy into New Population
P_c
Select Two Individu Based on Fitness
Perform Crossover
Insert Offspring into New Population
i := i + 1
P_m
Select One Individual Based on Fitness
Perform Mutati
Insert Mutant into New Population

PSEUDO-CODE FOR THE BASIC GENETIC ALGORITHM

```
PROCEDURE GA:

 BEGIN
     t=0
     Initialize population P(t)

   REPEAT
     t=t+1
     Evaluate individuals in P(t-1) for
           fitness
     Select P(t) from P(t-1) using FPR
     Perform Crossover on P(t)
     Perform small amount of Mutation
   UNTIL (TERMINATION CONDITION)

END
```

PROBABILISTIC SELECTION

INDIVIDUALS ARE SELECTED FOR REPRODUCTION TO PARTICIPATE IN CROSSOVER AND MUTATION BASED ON FITNESS

- **Better individuals are usually chosen**
  - The best individual is not necessarily chosen
  - The worst individual is not necessarily excluded
- **Thus, there is SOME greedy hill-climbing**
- **But, there is considerable selection of individuals that are the INFERIOR nased on the current evidence of the search**
- **Resembles simulated annealing**
- **A population is used (i.e., the search is not merely point-to-point)**

"I think it would be a most extraordinary fact if no variation ever had occurred useful to each being's own welfare ... . But if variations useful to any organic being do occur, assuredly individuals thus characterised will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance they will tend to produce offspring similarly characterised. This principle of preservation, I have called, for the sake of brevity, Natural Selection."

--- Charles Darwin in*On the Origin of Species by Means of Natural Selection* (1859)

CROSSOVER OPERATION

- THE predominant operation with GAs

- Two parental strings chosen based on fitness

- Pick interstitial point from 1 to L–1 (using a uniform random distribution)

Parent 1	Parent 2
011	110

Two crossover fragments(if point 2 is chosen)

Crossover fragment 1	Crossover fragment 2
01–	11–

Two remainders (if point 2 is chosen)

Remainder 1	Remainder 2
– – 1	– – 0

Two offspring produced by crossover

Offspring 1	Offspring 2
111	010

MUTATION OPERATION

- VERY occasional – Maybe 1 bit per generation

- One parental string chosen based on fitness.

- Pick point from 1 to L (using a uniform random distribution)

Parent 1
010

One Offspring (Point 3 chosen and mutated)

Offspring
011

EXAMPLE RUN OF THE GENETIC ALGORITHM WITH A POPULATION OF SIZE 4 BETWEEN GENERATION 0 AND 1 FOR SIMPLE 3-DIMENSIONAL OPTIMIZATION PROBLEM

	Gen 0			Mating pool		Gen 1		
1	011	3	.25	011	3	2	111	7
2	001	1	.08	110	6	2	010	2
3	110	6	.50	110	6	---	110	6
4	010	2	.17	010	2	---	011	3
Total		12			17			18
Worst		1			2			2
Aver		3.00			4.25			4.5
Best		6			6			7

GENETIC ALGORITHMS ARE PROBABILISTIC

• Creation of the initial random population (generation 0) (uniform distribution)

• Probabilistic selection of operation (uniform distribution)

• Probabilistic selection of participant(s) for the operation (distribution based on fitness)

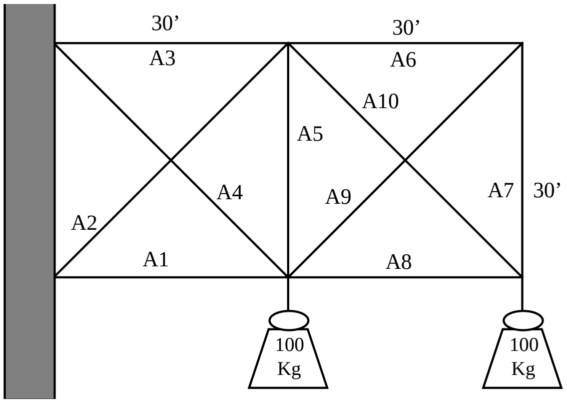
• Probabilistic selection of crossover or mutation point (uniform distribution)

• (Often) probabilistic selection of fitness cases (uniform distribution)

FOUR MAJOR PREPARATORY STEPS FOR THE GENETIC ALGORITHM

- **Determining the representation scheme**
  - structure (e.g., fixed length string, variable length string, data structure, etc.)
  - if the structure is fixed length string, then determine the alphabet size K and the string length L
  - mapping from points in search space of the problem to the structure, and vice versa
- **Determining the fitness measure**
  - May involve numerous fitness cases
- **Determining the parameters**
  - population size M
  - number of generations G
  - other control parameters
- **Determining the method for designating a result (e.g., best-so-far) and the criterion for terminating a run (e.g., maximum number of generations to be run or achievement of some satisfactory level of performance)**

10-MEMBER TRUSS PROBLEM (GOLDBERG AND SAMTANI 1986)



- **Find the 10 cross-sectional areas A_1, A_2, ..., A_{10} that minimize the total weight (dollar cost) of the 10 members of the truss while supporting the load (i.e., satisfying stress constraints).**

Goldberg, David E. and Samtani, M.P. Engineering optimization via genetic algorithms. In *Proceedings of the Ninth Conference on Electronic Computation*. 1986. Pages 471-482.

10-MEMBER TRUSS PROBLEM – CONTINUED

- **The weight (dollar cost) of each member of the truss is based on its volume.**

- **The volume of a member is its cross-sectional area, A_i, times its length.**

- **The length are either 30 feet or 42.4 feet**

- **The volume is smaller for a member with smaller cross-sectional area.**

- **However, the members must be large enough to support the loads (i.e., satisfy stress constraints).**

- **Hence, competition between small cross-sectional area and strength.**

FOUR MAJOR PREPARATORY STEPS
10-MEMBER TRUSS PROBLEM

1. REPRESENTATION SCHEME
40-BIT CHROMOSOME (GENOME)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
0010	**1110**	**0001**	**0011**	**1011**	**0011**	**1111**	**0011**	**0011**	**1010**

FOUR MAJOR PREPARATORY STEPS 10-MEMBER TRUSS PROBLEM

2. FITNESS MEASURE

- Decode the 40-bit chromosome into the 10 cross-sectional areas: A_1, A_2, ..., A_{10}.
- Compute the volume of each member of the truss as its cross-sectional area times its length (30 feet or 30•2 = 42 feet)
- Compute cost of each member
- Compute the sum, over the 10 members, the cost to get the total cost.
- The smaller the total cost the better. The minimal cost is not known in advance.
- Penalize violations of stress constraints. For example, a stress that is 10% above the maximum set by the constraint for that member might be penalized (e.g., 110%).

FOUR MAJOR PREPARATORY STEPS 10-MEMBER TRUSS PROBLEM – CONTINUED

3. MAJOR PARAMETERS

- **Population size, *M* = 200**
- **Maximum number of generations to be run, *G* = 50**

4. TERMINATION

- **The minimal cost is not known in advance. The criterion for terminating a run (e.g., maximum number of generations to be run or plateau in fitness of best-of-generation individuals**
- **Method for designating a result is "best-so-far" individual**

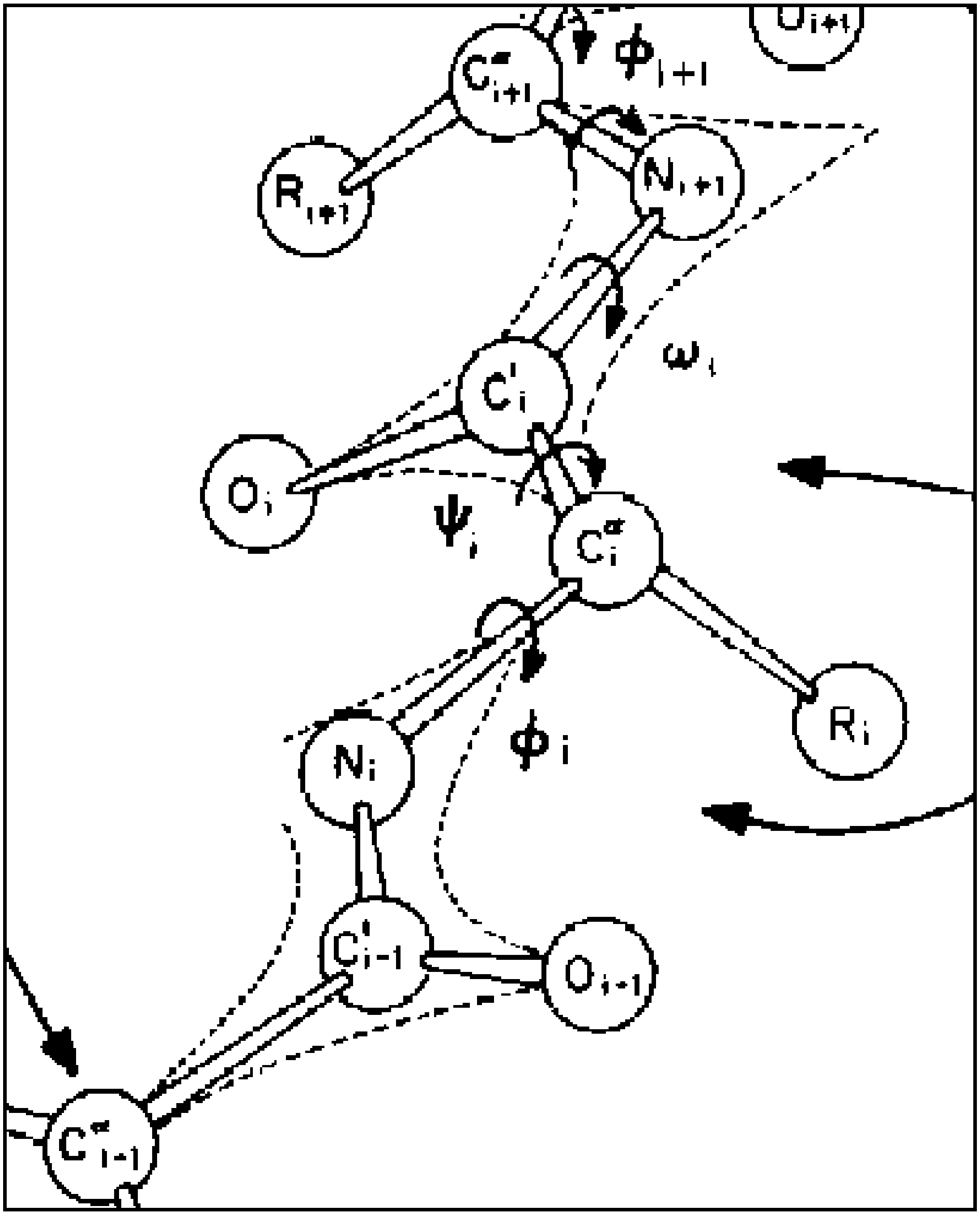
GA TABLEAU FOR 10-MEMBER TRUSS

Objective:	**Find the globally optimum combination of cross-sectional areas for the 10 members of the truss.**
Representation scheme:	**• structure = fixed length string** **• alphabet size *K* = 2 (binary)** **• string length *L* = 40** **• mapping = each 4-bit group of the 40-bit string corresponds to the cross-sectional area (with granularity of 16) of one of the 10 members of the truss.**
Fitness cases:	**Only one.**

Raw fitness:	**Raw fitness = cost (weight) of 10 members with penalty for constraint violations (uses packaged evaluation program).**
Parameters:	**• Population size *M* = 200.** **• Maximum number of generations to be run *G* = 50.**
Termination criteria:	**The GA has run for *G* generations.**
Result designation:	**The best-so-far individual in the population.**

GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

- **Find the ϕ_i (phi) and ψ_i (psi) angles for each amino acid residue *i* and the 0-8 additional angles χ_{i1}, ..., χ_{i8} for each residue *i*.**


O_{i+1}
C^{α}_{i+1}
ϕ_{i+1}
R_{i+1}
N_{i+1}
C'_{i}
ω_{i}
O_{i}
ψ_{i}
C^{α}_{i}
R_{i}
ϕ_{i}
N_{i}
C'_{i-1}
O_{i-1}
C^{α}_{i-1}

GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

The representation of the sequence of ϕ_i and ψ_i angles for each amino acid residue *i* and the 0-8 additional angles χ_{i1}, ..., χ_{i8} for each residue *i*

BINARY ENCODING

<----RESIDUE #1--------------------------><----RESIDUE #2----

ϕ_1	ψ_1	χ_{11}	χ_{12}	ϕ_2	ψ_2 • • •
101010110	011010110	110100101	001110101	101101101	001001010

REAL-VALUED GENES

<----RESIDUE #1--------------------------><----RESIDUE #2----

ϕ_1	ψ_1	χ_{11}	χ_{12}	ϕ_2	ψ_2 • • •
+45.6	–22.7	+156.9	–5.2	+29.8	–122.7

LE GRAND’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

"AMBER" POTENTIAL ENERGY FUNCTION

Approximates N-body problem with 2-body terms by measuring all N^2 pairwise interactions of N atoms

- **<u>VAN DER WAALS</u>: Repulsion and attraction inversely depends on 12th and 6th powers of distance between each pair of non-bonded atoms. (Important at short range; irrelevant at a distance).**

- **<u>COULOMB:</u> Electrostatic attraction and repulsion inversely depends on distance between each pair of non-bonded atoms.**

LE GRAND’s USE OF GA FOR Protein Tertiary Structure Prediction

"AMBER" POTENTIAL ENERGY FUNCTION

- **Force (depending on square of deviation) to hold each <u>2-ATOM BOND DISTANCE</u> at a constant equilibrium value.**

- **Force (depending on square of deviation) to hold each <u>3-ATOM BOND ANGLE</u> at a constant equilibrium value.**

- **Force is Fourier series with frequency and phase dependent on <u>4-ATOM DIHEDRAL ANGLE</u>.**

LE GRAND’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

- **Fitness is AMBER plus additional van der Waals and Coulomb contributions for 1st and 4th atoms of 4-dihedrally-bound atoms AND additional van der Waals contribution for polar hydrogens and non-bonded oxygen and nitrogen.**
- **3 kinds of crossover (single-point, two-point, and uniform)**
- **Steady-state GA is used. (Tends to be greedy).**
- **High (and changing) mutation rate.**
- **Child only replaces parent if it is better than most similar existing individual in the population (a variation of phenotypic sharing)**
- **Population size*M* = 200.**

LE GRAND’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

Objective:	**Given the primary sequence of a protein, the ϕ_i (phi) and ψ_i (psi) angles for each amino acid residue *i* and the 0-8 additional angles χ_{i1}, ..., χ_{i8} for each residue *i*.**
Representation scheme:	**• Structure = fixed length string (for a particular protein)** **• Alphabet of real-valued genes** **• String length *L* varies** **• Mapping: See above.**
Fitness cases:	**Only one (for a given protein).**
Raw fitness:	**Modified AMBER potential energy function.**

Parameters:	**• Population size*M* = 200.** **• Maximum number of generations to be run specified as 100,000 (200 x 500) iterations.** **• Variation of phenotypic sharing.**
Termination criteria:	**100,000 (200 x 500) iterations OR variance of population is less than 0.1 OR average distance between 200 randomly selected pairs is less than 0.1.**
Result designation:	**• Best-so-far individual**

LE GRAND’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

- **Tried on 4 proteins**
  - 46-residue crambin
  - 26-residue melittin
  - 36-residue avian pancreatic polypeptide inhibitor
  - 106-residue cytochrome b562 (4 helix bundle)

- **Tried on 3 polypeptides**
  - Polyalanine A9 (Alanine – 9 times)
  - AGAGAGAGA (9 amino acid residues)
  - {Met}-enkephalin

LE GRAND'S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION

• Le Grand, Scott and Merz, Kenneth M., Jr. 1993. The application of the genetic algorithm to the minimization of potential energy functions." *Journal of Global Optimization* 3(1) 49–66.

• Le Grand, Scott. 1993. *The Application of the Genetic Algorithm to Protein Tertiary Structure Prediction*. PhD Dissertation. Department of Biochemistry, The Pennsylvania State University.

SUN’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION USING REDUCED REPRESENTATION MODEL

- **Sun, Shaojian. 1993. Reduced representation model of protein structure prediction: Statistical potential and genetic algorithms. *Protein Science*. Volume 2. Pages 762-785.**

- **Reduced representation**
  - Only backbone atoms
  - Ideal fixed bond lengths and angles
  - Single virtual united-atom as side chain

- **Goal is to find the ϕ (phi) and ψ (psi) angles (2 per amino acid residue)**

SUN’S USE OF GA FOR PROTEIN TERTIARY STRUCTURE PREDICTION USING REDUCED REPRESENTATION MODEL

- **Results in folded versions of**
  - 26-residue melittin – RMS error of 1.6 Å
  - 36-residue avian pancreatic polypeptide inhibitor (APPI)
  - 18-residue apamin (with 2 disulfide bonds) from bee venom

SUN’S PROTEIN TERTIARY STRUCTURE PREDICTION USING REDUCED REPRESENTATION MODEL – CONTINUED

- **Fitness was a statistical interatomic potential function of his own design**
  - Based on 110 proteins (with less than 50% identity)
  - melittin and avian pancreatic polypeptide inhibitor (APPI) were in the 110

- **Fitness - 2 components**
  - Local (NOTE: possible computer savings)
  - Non-local

- **Apparently floating-point gene values. 2 x 26 = 52 for melittin. Values are integers from –180 to +180. Equivalent to 52 x 9 = 468 bits.**

- **Population size*M* = 90**

GA TABLEAU FOR SUN’S PROTEIN TERTIARY STRUCTURE PREDICTION USING REDUCED REPRESENTATION MODEL

Objective:	**Given the primary sequence of a protein, find the three-dimensional conformation of the protein in the form of the dihedral ϕ and ψ angles using a reduced representation model of protein.**
Representation scheme:	**• structure = fixed length string (for a particular protein)** **• alphabet size *K* = 2 (in binary equivalent)** **• string length *L* = 468 (in binary equivalent)** **• mapping.**
Fitness cases:	**Only one (for a given protein).**

Raw fitness:	**Statistical fitness function.**
Parameters:	**• Population size*M* = 90.** **• Maximum number of generations to be run*G* = ???.** **• Special (???) mutation operation at ??? frequency**
Termination criteria:	**??? (Reports convergence of all 90!!!).**
Result designation:	**Best-so-far individual**

SUN’S PROTEIN TERTIARY STRUCTURE PREDICTION USING REDUCED REPRESENTATION MODEL – CONTINUED

- **Reproduction NOT based on fitness. Creates 2M individuals.**
- **Crossover NOT based on fitness. Creates M individuals.**
- **Special mutation operation (sometimes changing several values at once). Creates 2M individuals.**
- **Selects the best M out of 5M new individuals.**
- **On generation 0, initial energy of 90 individual ranges from 1,440.08 to 15,746.34 units (with mean of 2912.00 and standard deviation of 1,960.75)**
- **On generation X, mean of the 90 individuals "converged" to 1,290.50 (with a standard deviation of 0.31 -– i.e., one part in about 4,000).**

GA’S AND PROTEIN FOLDING WITH SELF-AVOIDING GRAPHS

- **Unger, Ron and Moult, John. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology* 231 (1993): 75–81.**
- **Unger, Ron and Moult, John. On the applicability of genetic algorithms to protein folding. *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on Systems Science 1993*. In Mudge, Trevor N., Milutinovic, Veljko, and Hunter, Lawrence (editors). *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on Systems Science 1993*. Los Alamitos, CA: IEEE Computer Society Press. 1993. Volume I. Pages 715-725.**
- **Unger, Ron and Moult, John. A genetic algorithm for 3D protein folding simulations. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Ed.**

Stephanie Forrest. San Mateo, CA: Morgan Kaufmann Publishers, 1993. 581–588.

UNGER AND MOULT’S SELF-AVOIDING GRAPHS

- **Individuals in the population are self-avoiding point-labeled (2 colors) graphs embedded in a 2-dimensional checkerboard lattice**
- **That is, individual in the population are the actual structures that the GA operates on**
  - Phenotype (the individual) = Genotype
- **2 psuedo-amino-acids:**
  - Black (Hydrophobic)
  - White (Other)
- **Fitness is decremented by -1 for each adjacent BLACK point along backbone that is not diagonally adjacent or adjacent along backbone**
  - The 2 termini can contribute up to -3
  - Ordinary points can contribute up to -2

- There are 83,779,155 20-long self-avoiding graphs of the sequence -----------. Fitness ranges from 0 to -9 (best) and there are only 4 9-scoring best conformations out of 83,779,155

UNGER AND MOULT'S SELF-AVOIDING GRAPHS

- **Mutation operation**
  - Pick point
  - Keep trying random rotations that create self-avoiding graph as a result
- **Crossover**
  - Pick point
  - Keep trying random rotations that create self-avoiding graph as a result

UNGER AND MOULT'S SELF-AVOIDING GRAPHS

- **Population size*M* = 200**
- **Initialization: All alike (flat = 180 degrees)**
- **Accept result of mutation with Metropolis algorithm**
- **Accept result of crossover with Metropolis algorithm**
- **Global minimum of -9 found in all 5 runs after 8,800,000; 7,400,000; 3,200,000; 470,000; and 292,000 fitness evaluations. That is, between 9:1 and 284:1.**

PROTEIN FOLDING WITH EXTENSIVELY MODIFIED GENETIC ALGORITHM (PEDERSEN AND MOULT 1997)

Pedersen, Jan T. and Moult, John. 1997. Protein folding simulations with gentic algorithms and a detailed molecular description. *Journal of Molecular Biology*. 269: 240 – 259.

• Dihedral angle library: For each residue type, a set of observed Φ, ψ, and χ angles was compiled

• Conformations were not generated at random, but, instead, were drawn from the Φ, ψ, and χ dihedral angle library. Angles were randomized sequentially residue-by-residue. If can det Walls overlap exceeds 0.5 Å, rechoose. Bactrack to previous residue if necessary. In practice, this is linear (while worst case is exponential for worst case)

PROTEIN FOLDING WITH EXTENSIVELY MODIFIED GENETIC ALGORITHM (PEDERSEN AND MOULT 1997)

• Initial random population for genetic algorithm (GA) comes from 20,000-step Monte Carlo (MC) (i.e., simulated annealing (SA)) run (with initial random conformation as starting point for the SA run). A population of is created every 100-th point toward the end of the SA run.

• No mutations in GA.

• 10% elitism. Typical run is 100 generations and converges in 40 to 60 generations.

PROTEIN FOLDING WITH EXTENSIVELY MODIFIED GENETIC ALGORITHM (PEDERSEN AND MOULT 1997)

- **Crossover sites are not chosen with usual uniform probability. Insteasd, choice of sites is based on conformational diversity.**

- **Crossover operation is extensively modified. For both offspring of normal crossover, 50 Φ / ψ pairs are searched for sterically acceptable outcome (half drawn from the dihedral angle library and half from a representation set of 7 residue conformtions). For each trial, 100 rouns of Monte Carolo simulation are made (based on small 5 degree moves in angles). The lowest free energy conformation is accepted using Metropolis test.**

TERITIARY STRUCTURE PREDICTION

- **Schulze-Kremer, Steffen. Genetic algorithms for protein tertiary structure prediction. *Parallel Genetic Algorithms*. Ed. Joachim Stender. Amsterdam: IOS Press, 1992. 129–149.**

- **Schulze-Kremer, Steffen. Genetic algorithms for protein tertiary structure prediction. *Parallel Problem Solving from Nature 2*. Ed. Reinhard Maenner and Manderick, Bernard. Amsterdam: North-Holland, 1992. 391-400.**

- **Schulze-Kremer, Steffen. Genetic algorithms for protein tertiary structure prediction. *Machine Learning: European Conference on Machine Learning, Vienna, Austria, April 5–7, 1993, Proceedings.* Ed. Pavel B. Brazdil. Berlin: Springer-Verlag, 1993. 262–279.**

SCHULZE-KREMER’S TERITIARY STRUCTURE PREDICTION

- **Tried on 46-residue crambin - 1.86 Å RMS**
- **Reduced representation model using the ϕ (phi) and ψ (psi) angles and 8 additional angles χ_1, ..., χ_8 per each amino acid residue.**

- **Chromosome for GA consists of floating-point numbers (i.e., 10 x 46 = 460 floating-point numbers for crambin) (NOTE: If as few as 10 bits were assigned to a floating point number, that's 4,600 bits).**

- **Initial population was either to all-180-degrees or used list of 10 most frequent angles appearing in 129 proteins from PDB.**

SCHULZE-KREMER’S TERITIARY STRUCTURE PREDICTION – FITNESS MEASURE

- **Simplified version of CHARMM potential energy function**
  - van der Waals, E_{vdW}
  - Coulomb electrostatic, E_{el}
  - bond-length potential, E_{bond} (CONSTANT)
  - bond-angle potential,E_{phi} (CONSTANT)
  - torsion-angle potential, E_{tor}
  - improper torsion-angle potential, E_{impr} (CONSTANT)
  - hydrogen bonds E_H (EXCLUDED)
  - solvent interaction, E_{cr} and E_{chpi} (CONSTANT)
- **That is, $E = E_{vdW} + E_{el} + E_{tor}$**

SCHULZE-KREMER’S TERITIARY STRUCTURE PREDICTION – PARAMETERS FOR RUN

- **Population size*M* = 10.**
- **Maximum number of generations*G* = 1,000.**
- **Mutation changes an angle by plus or minus 1, 5, or 10 degress**
- **20% mutation rate at start of run (with 10-30-60 weighting among 1, 5, and 10 degree changes) and 70% at end (with 80-20-0 weighting).**
- **Crossover 70% rate at start of run (with 90-10 weighting between uniform and two-point) and 10% at end of run (with 10-90 weighting).**
- **Selection 80% at beginning and 20% at end.**

SCHULZE-KREMER'S TERITIARY STRUCTURE PREDICTION

Objective:	**Given the primary sequence of a protein, find the three-dimensional conformation of the protein in the form of the ϕ (phi) and ψ (psi) angles and 8 additional angles χ_1, ..., χ_8 per each amino acid residue.**
Representation scheme:	**• structure = fixed length string** **• alphabet of floating-point genes** **• string length *L* = 460 floating-point numbers for crambin.** **• mapping: There are 10 angles for each amino acid residue for a total of 460 floating-point numbers for crambin.**

Fitness cases:	**One.**
Raw fitness:	**CHARMM potential energy function.**
Parameters:	**• Population size*M* = 10.** **• Maximum number of generations to be run, *G* = 1,000.**
Termination criteria:	***G* = 1,000 generations have been run.**
Result designation:	**Best-so-far individual**

GENETIC PROGRAMMING

How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?

---Attributed to Arthur Samuel - about 1959

AUTOMATIC PROGRAMMING

WYWIWYG – "WHAT YOU WANT IS WHAT YOU GET"

• Starts from a high-level statement of the problem

• Produces an entity that runs on a computer

• Produces the size and shape of the solution (i.e., user doesn't prespecify exact number of primitive steps or exact arrangement of steps

• Can automatically identifies useful groups of steps (i.e., subroutines) and then reuses them (sometimes with different instantiations of parameters)

• Can implement internal memory and data structures such as single variables, vectors, arrays, stacks, queues, lists, and relational memory

A AUTOMATIC PROGRAMMING – CONTINUED

• Can implement iterations and recursions

• Can automatically determine the number of subroutines, the number of arguments that they each possess, how the subroutines hierarchically refer to one another, and whether to employ internal memory, iterations, and recursion

• Can automatically organize groups of steps into a hierarchy

• Can implement the full range of programming constructs that human computer programmers find useful, including macros, libraries, typing, pointers, conditional operations, logical functions, integer functions, floating-point functions, complex-valued functions, multiple inputs, multiple outputs, and machine code instructions

AUTOMATIC PROGRAMMING – CONTINUED

- **Unmistakably distinguishes between what the user must provide and what the system delivers, is problem-independent, and operates in a well-defined way that does not rely on discretionary human intervention or any hidden steps**
- **Can produces satisfactory solutions for a wide variety of problems from many different fields**
- **Scales well to larger versions of the same problem**
- **Can produce results that are competitive with human-produced results**

GENETIC PROGRAMMING (GP)

"Genetic programming *is* automatic programming. For the first time since the idea of automatic programming was first discussed in the late 40’s and early 50’s, we have a set of non-trivial, non-tailored, computer-generated programs that satisfy Samuel’s exhortation: Tell the computer what to do, not how to do it.’ "

– John Holland, University of Michigan, 1997

A ONE-INPUT, ONE-OUTPUT PROGRAM IN C

```
int foo (int time)
{
   int temp1, temp2;
   if (time > 10)
       temp1 = 3;
   else
       temp1 = 4;
   temp2 = temp1 + 1 + 2;
   return (temp2);
}
```

RESULTS OF THE PROGRAM IN C

Independent variable (input) *TIME*	Dependent variable (output)
0	6
1	6
2	6
3	6
4	6
5	6
6	6
7	6
8	6
9	6
10	6
11	7
12	7
13	7
13	7
15	7

16	7
17	7
18	7
19	7
20	7

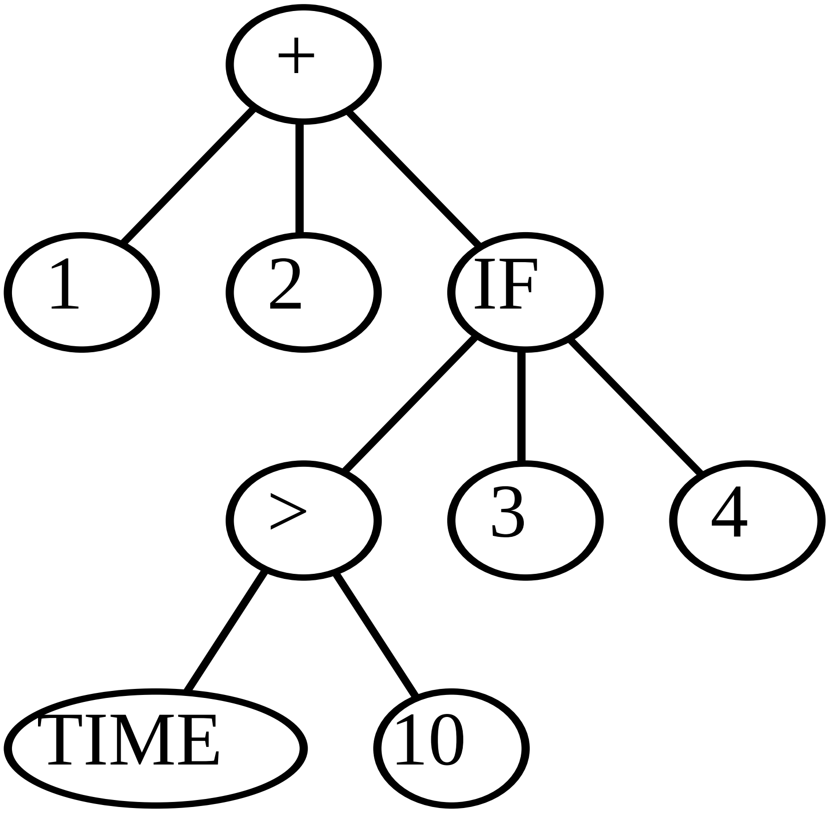
PROGRAM IN LISP = INDIVIDUAL PROGRAM = PARSE TREE = PROGRAM TREE = DATA = LIST

```
(+ 1 2 (IF (> TIME 10) 3 4))
```

- ATOMS = `1, 2, 10, 3, 4, TIME`
- FUNCTIONS = `+, IF, >`

PROGRAM IN LISP = INDIVIDUAL PROGRAM = PARSE TREE = PROGRAM TREE = DATA = LIST

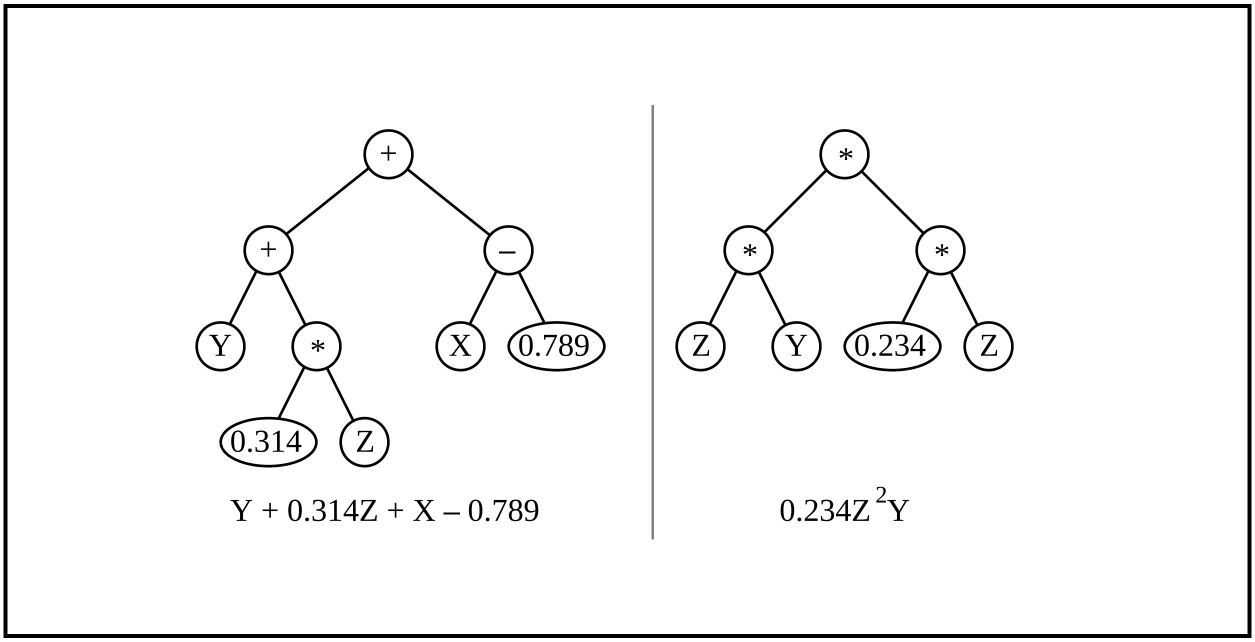
```
(+ 1 2 (IF (> TIME 10) 3 4))
```

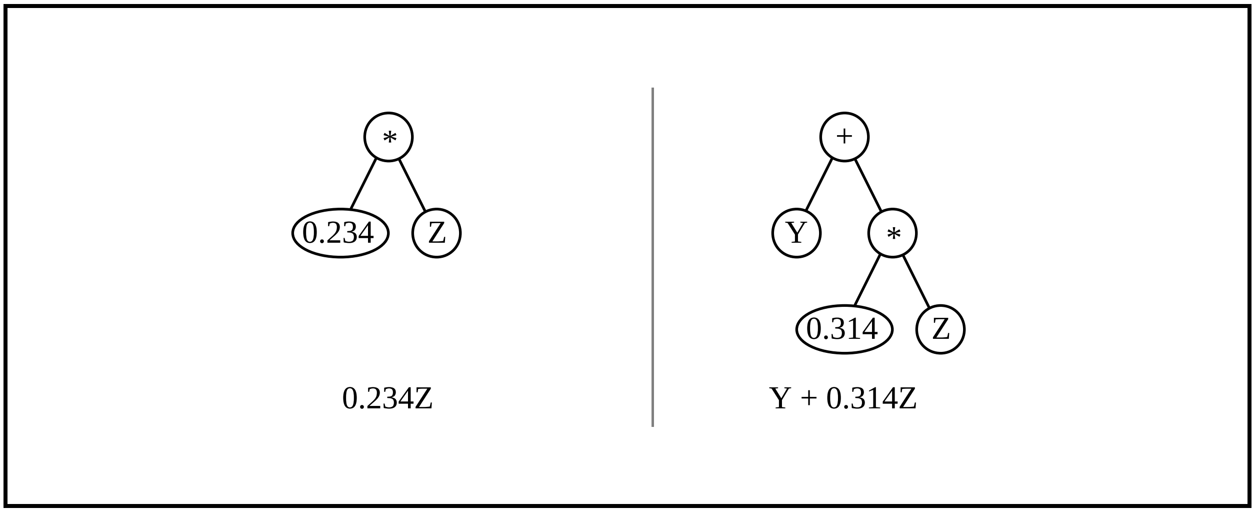


FOUR RANDOM COMPUTER PROGRAMS FOR THE INITIAL RANDOM POPULATION OF A RUN (GENERATION 0)

- **Terminal set T = {X, Y, Z, ←}**
- **Function set F = {+, –, *, %, IFLTE}**

- **The 4 programs are of different size and shape**

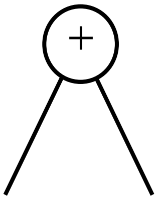



*
0.234
Z
0.234Z
+
Y
*
0.314
Z
Y + 0.314Z

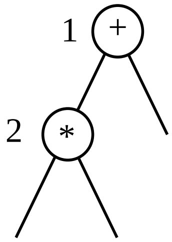
EXAMPLE OF THE CREATION OF A RANDOM PROGRAM TREE

- **Terminal set T = {A, B, C}**
- **Function set F = {+, –, *, %, IFLTE}**

- **Randomly choose a function or terminal from the combined set {+, –, *, %, IFLTE, A, B, C}. Suppose it the two-argument addition (+) function.**

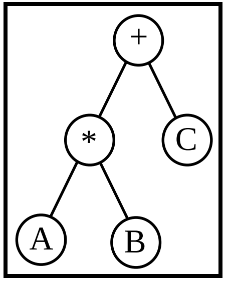


- **Randomly choose another function or terminal from {+, –, *, %, IFLTE, A, B, C}, say the two-argument multiplication (*) function.**



EXAMPLE OF THE CREATION OF A RANDOM PROGRAM TREE – CONTINUED

- **Continue in this manner. Suppose that the next 3 random choices from {`+`, `–`, `*`, `%`, `IFLTE`, `A`, `B`, `C`}, say `A`, `B`, and `C`.**

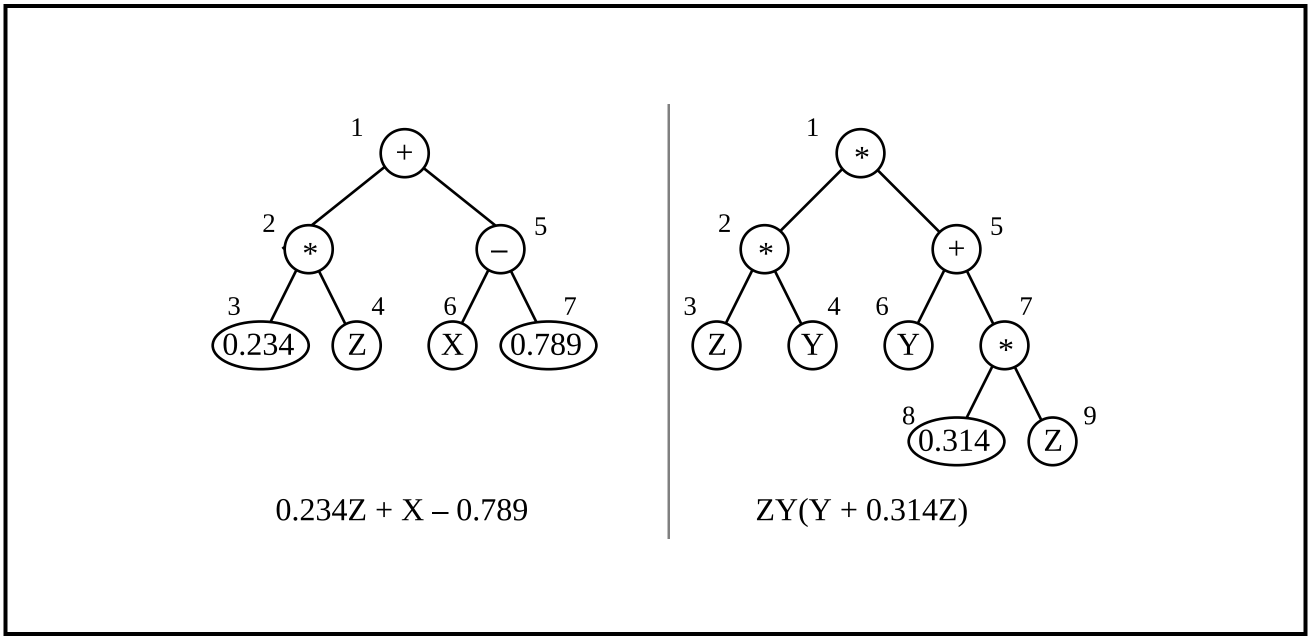


- **The growth process ends when all paths end in a terminal {`A`, `B`, `C`}.**
- **Force a choice from the terminal set (rather than the combined set) if the preestablished maximum size (measured in terms of number of functions and terminals or in terms of depth) is being exceeded.**

CROSSOVER (SEXUAL RECOMBINATION) OPERATION FOR COMPUTER PROGRAMS (TREES)

- **Select two parents probabilistically based on fitness**
- **Randomly pick a number from 1 to `NUMBER-OF-POINTS` – independently for each of the two parental programs**
- **Identify the two subtrees rooted at the two picked points**

CROSSOVER (SEXUAL RECOMBINATION) OPERATION FOR COMPUTER PROGRAMS (TREES)



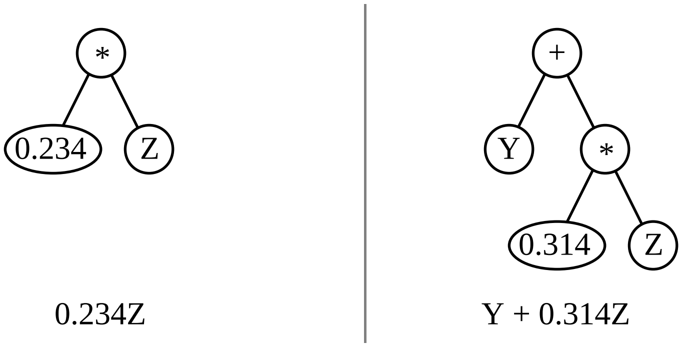
Parent 1:

```
(+ (* 0.234 Z) (- X 0.789))
```

Parent 2:

```
(* (* Z Y) (+ Y (* 0.314 Z)))
```

CROSSOVER FRAGMENTS (THE TWO SUBTREES ROOTED AT THE TWO PICKED POINTS)



Crossover Fragment 1:

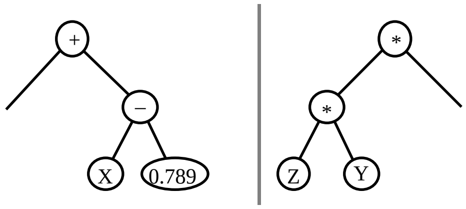
```
(+ (* 0.234 Z) (- X 0.789))
```

Crossover Fragment 2:

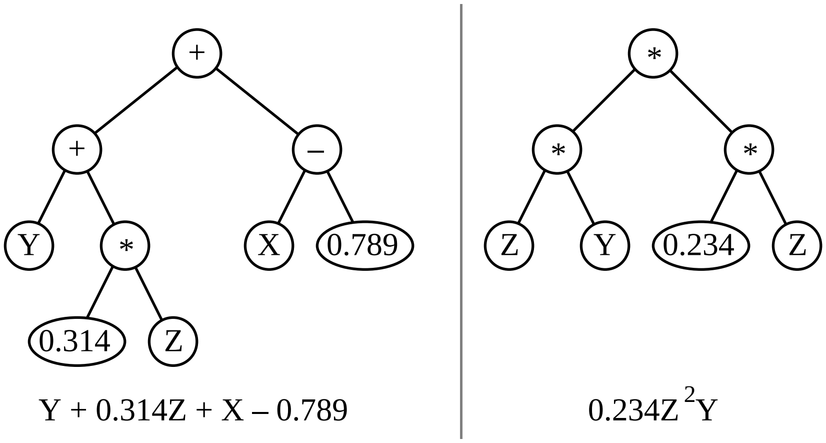
```
(* (* Z Y) (+ Y (* 0.314 Z)))
```

83

TWO REMAINDERS



TWO OFFSPRING
THE CROSSOVER OPERATION PRODUCES TWO OFFSPRING



Offspring 1:

```
(+ (+ Y (* 0.314 Z))
   (- X 0.789))
```

Offspring 2:

```
(* (* Z Y) (* 0.234 Z))
```

THE CROSSOVER OPERATION PRODUCES SYNTACTICALLY VALID, EXECUTABLE COMPUTER PROGRAMS

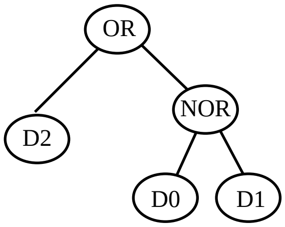
IF SET OF FUNCTIONS AND TERMINALS IS CLOSED (I.E., ANY FUNCTION CAN ACCEPT THE OUTPUT PRODUCED BY ANY OTHER FUNCTION OF TERMINAL

• Protected division % takes two arguments and returns one when division by 0 is attempted (including 0 divided by 0), and, otherwise, returns the normal quotient

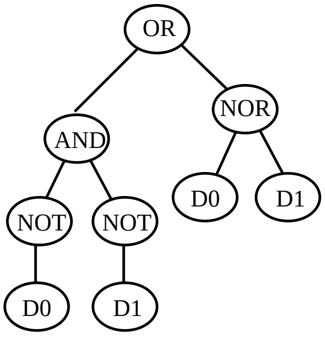
• Protected multiplication, addition, and subtraction

MUTATION OPERATION FOR PROGRAM TREES

- Select one parent probabilistically based on fitness
- Pick point from 1 to NUMBER-OF-POINTS (say the terminal D2 from among the 5 points here)



- Delete the entire subtree rooted at the picked point (i.e., delete the D2)
- Grow new subtree at the mutation point in the same way as used to generate trees for initial random population (generation 0)



FIVE MAJOR PREPARATORY STEPS FOR GP

- **determining the set of terminals**
- **determining the set of functions**
- **determining the fitness measure**
- **determining the parameters**
  - population size
  - number of generations
- **determining the method for designating a result and the criterion for terminating a run**

REGRESSION PROBLEM OF UNKNOWN FUNCTION

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	0.2496
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

REGRESSION PROBLEM OF UNKNOWN FUNCTION

Also Called

- System Identification problem
- the "Black Box" problem
- Model building
- Empirical discovery
- Non-parametric regression
- Datamining
- Forecasting / Time-series prediction

- **We seek**
  - Functional form of a good fit
  - Numerical parameters
  - Size and shape of the mathematical expression
- **Error is the fitness measure**
  - Sum, over fitness cases, of absolute error
  - Sum, over fitness cases, of squared error

TABLEAU FOR SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

Objective:	Find a function of one independent variable, in symbolic form, that fits a given sample of 20 (x_i, y_i) data points
Terminal set:	x (the independent variable).
Function set:	+, -, *, %, SIN, COS, EXP, RLOG
Fitness cases:	The given sample 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 the dependent variable produced by the individual program and the target value 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:	**Population size, *M* = 500.** **Maximum number of generations to be run,*G* = 51.**
Success Predicate:	**An individual program scores 21 hits.**

GENERATION 0 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

WORST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 10^{38}

```
(EXP (- (% X (- X (SIN X)))
(RLOG (RLOG (* X X)))))
```

Equivalent to

$e^{x / (x-\sin x) - \log \log x*x}$

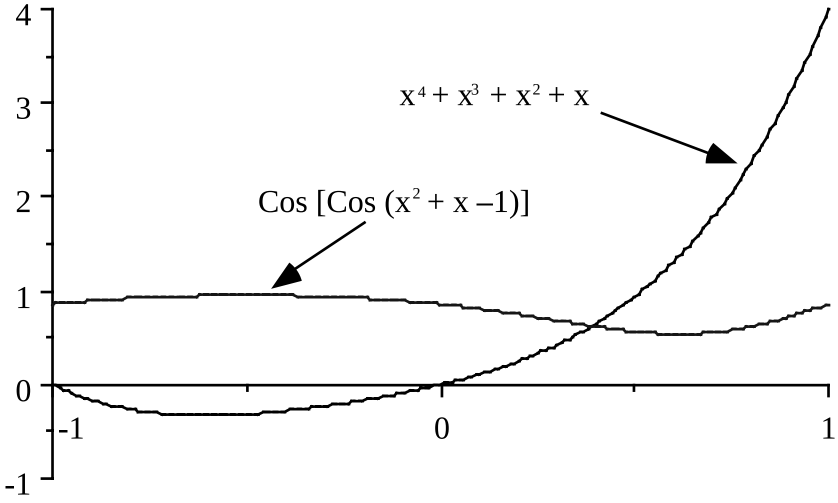
GENERATION 0 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

MEDIAN INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 23.67

```
(COS (COS (+ (- (* X X) (% X
X)) X)))
```

Equivalent to

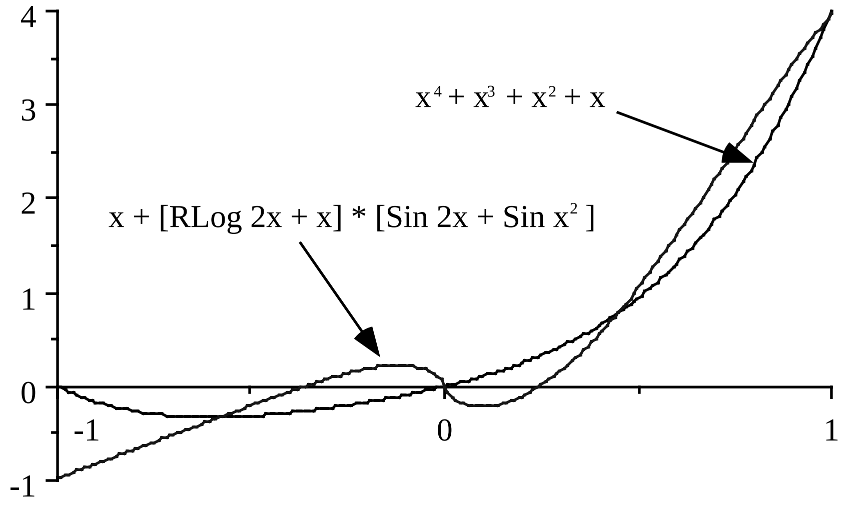
Cos [Cos (x^2 + x – 1)]



GENERATION 0 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

SECOND-BEST INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 6.05

x + [RLog 2x+x][Sin 2x+Sin x2]



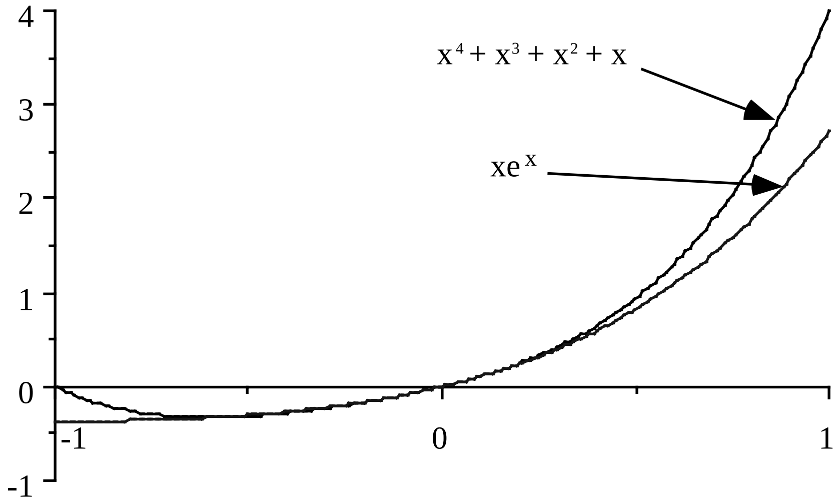
GENERATION 0 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 4.47

```
(* X (+ (+ (- (% X X) (% X X))
(SIN (- X X)))
        (RLOG (EXP (EXP X)))))
```

Equivalent to

xe^x



GENERATION 2 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 2 WITH RAW FITNESS OF 2.57

```
 (+ (* (* (+ X (* X (* X (% (% X
X) (+ X X)))))
               (+ X (* X X))) X) X)
```

Equivalent to...

$x^4 + 1.5x^3 + 0.5x^2 + x$

GENERATION 34 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION

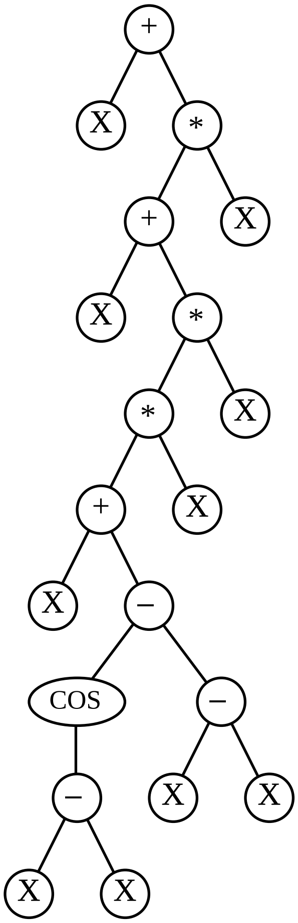
BEST-OF-RUN INDIVIDUAL IN GENERATION 34 WITH RAW FITNESS OF 0.00 (100%-CORRECT)

```
(+ X (* (+ X (* (* (+ X (- (COS
(- X X)) (- X X))) X) X)) X))
```

Equivalent to

$$x^4 + x^3 + x^2 + x$$

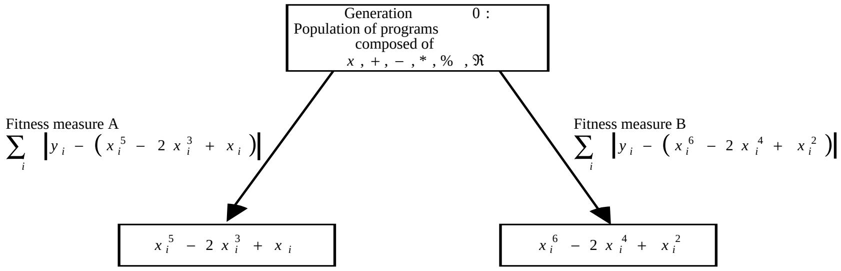
GENERATION 34 – SIMPLE SYMBOLIC REGRESSION OF UNKNOWN FUNCTION



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**

STRUCTURE ARISES FROM FITNESS



Generation 0 :
Population of programs composed of
$x, +, -, *, \%, \Re$

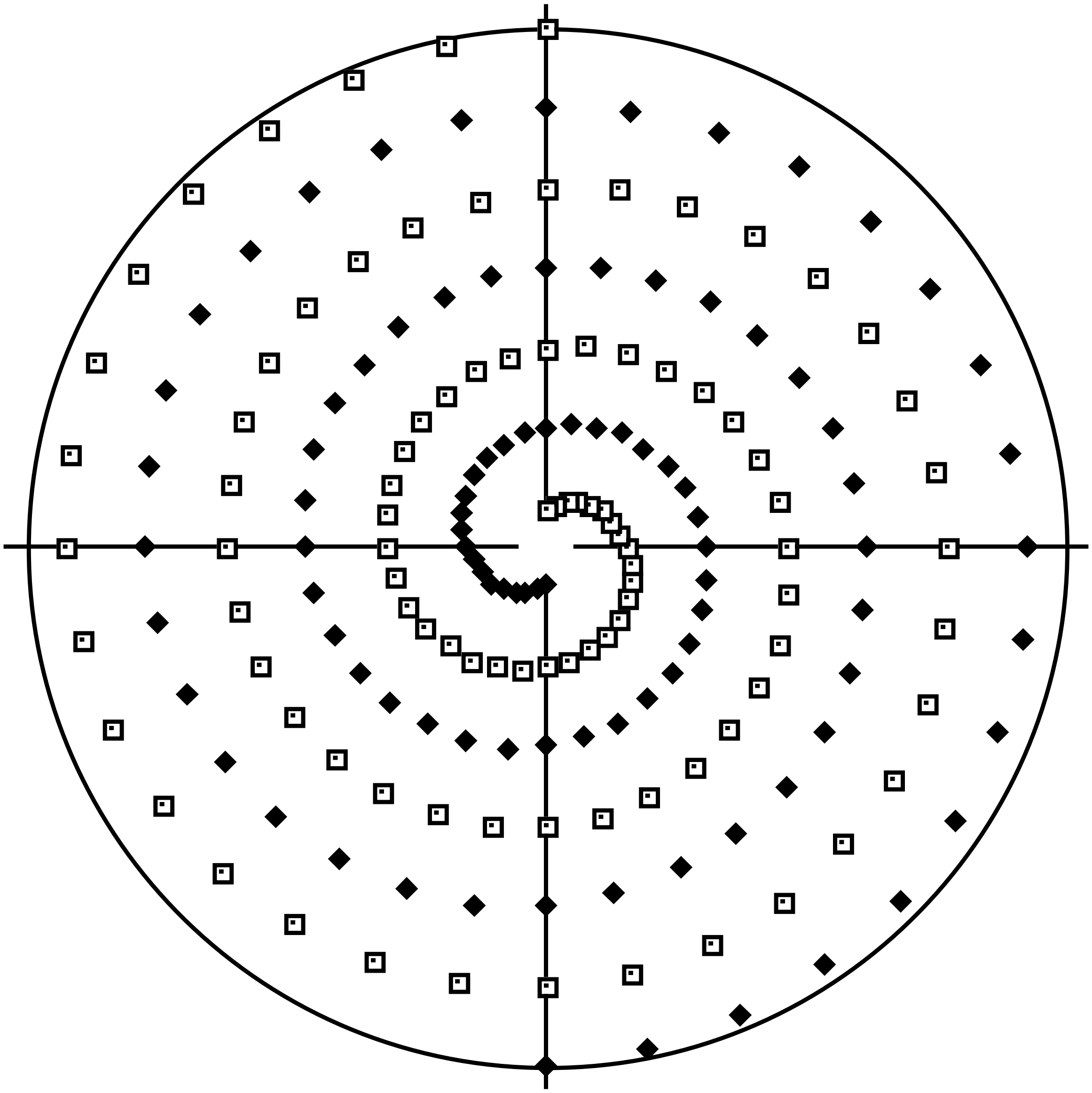
Fitness measure A
$\sum_i \left| y_i - \left(x_i^5 - 2x_i^3 + x_i \right) \right|$

Fitness measure B
$\sum_i \left| y_i - \left(x_i^6 - 2x_i^4 + x_i^2 \right) \right|$

$x_i^5 - 2x_i^3 + x_i$

$x_i^6 - 2x_i^4 + x_i^2$

INTER-TWINED SPIRALS CLASSIFICATION PROBLEM

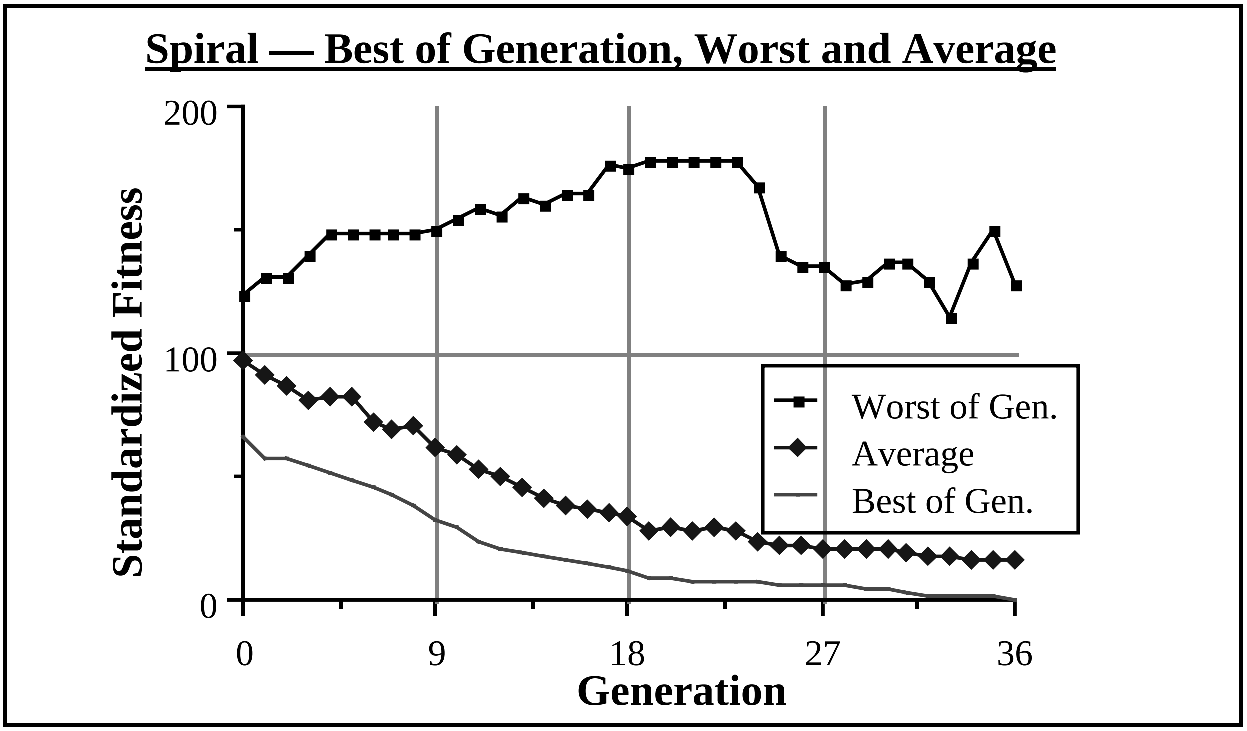


GP TABLEAU – INTERTWINED SPIRALS

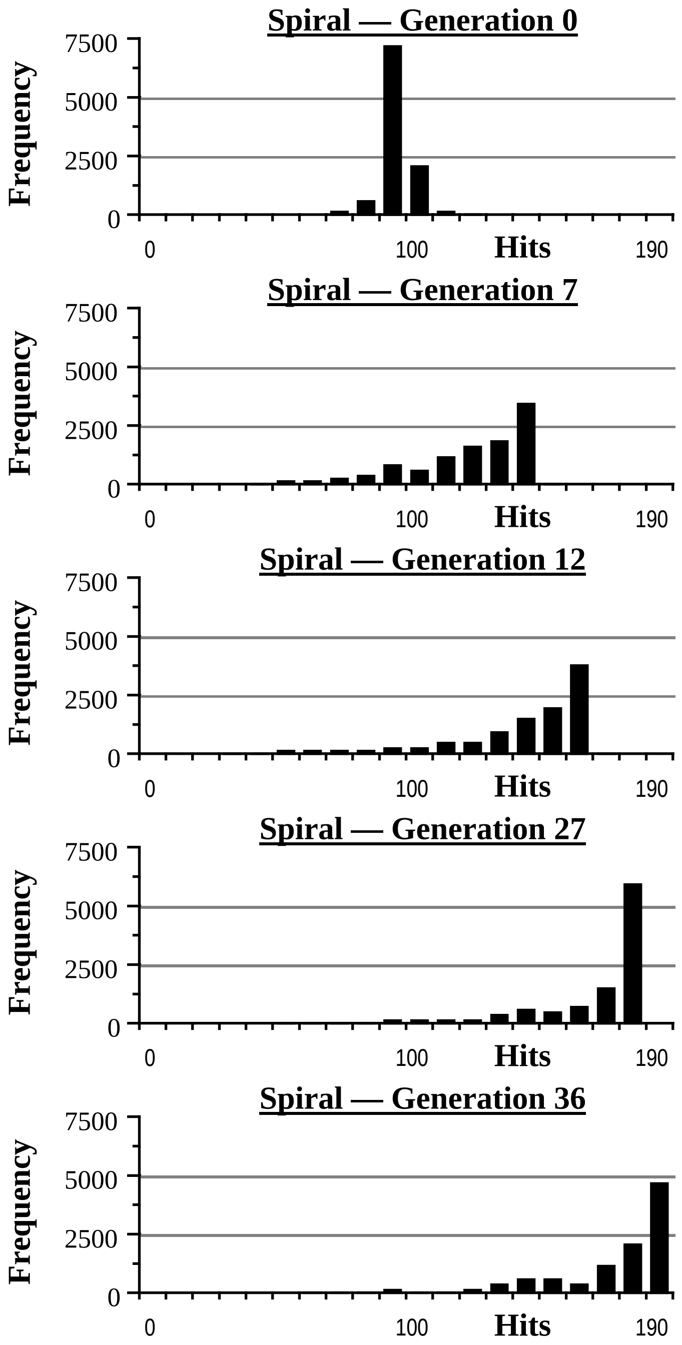
Objective:	**Find a program to classify a given point in the *x*-*y* plane to the red or blue spiral.**
Terminal set:	**`X`, `Y`, ←, where ← is the ephemeral random floating-point constant ranging between –1.000 and +1.000.**
Function set:	**`+, -, *, %, IFLTE, SIN, COS.`**
Fitness cases:	**194 points in the *x*-*y* plane.**
Raw fitness:	**The number of correctly classified points (0 – 194)**
Standardized fitness:	**The maximum raw fitness (i.e., 194) minus the raw fitness.**
Hits:	**Equals raw fitness.**
Wrapper:	**Maps any individual program returning a positive value to class +1 (red) and maps all other values to class –1 (blue).**

Parameters:	***M* = 10,000 (with over-selection). *G* = 51.**
Success predicate:	**An individual program scores 194 hits.**

INTER-TWINED SPIRALS FITNESS CURVES



INTER-TWINED SPIRALS HITS HISTOGRAMS FOR GENERATIONS 0, 7, 12, 27, AND 36

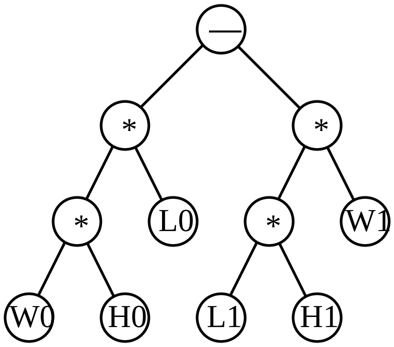


10 FITNESS-CASES SHOWING THE VALUE OF THE DEPENDENT VARIABLE, *D*, ASSOCIATED WITH THE VALUES OF THE SIX INDEPENDENT VARIABLES, L_0, W_0, H_0, L_1, W_1, H_1

Fitness case	L_0	W_0	H_0	L_1	W_1	H_1	D
1	3	4	7	2	5	3	54
2	7	10	9	10	3	1	600
3	10	9	4	8	1	6	312
4	3	9	5	1	6	4	111
5	4	3	2	7	6	1	–18
6	3	3	1	9	5	4	–171
7	5	9	9	1	7	6	363
8	1	2	9	3	9	2	–36
9	2	6	8	2	6	10	–24
10	8	1	10	7	5	1	45

SOLUTION USING GENETIC PROGRAMMING WITHOUT AUTOMATICALLY DEFINED FUNCTIONS (ADF’S)

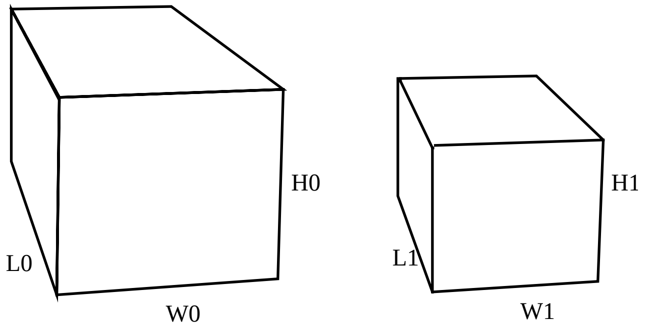
```
(- (* (* W0 L0) H0)
   (* (* W1 L1) H1))
```



DIFFERENCE IN VOLUME OF TWO BOXES

```
(- (* (* W0 L0) H0)
   (* (* W1 L1) H1))
```

```
D = W0*L0*H0 – W1*L1*H1
```

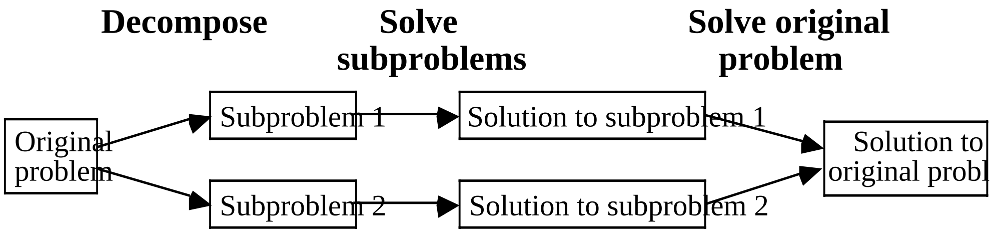


AUTOMATICALLY DEFINED FUNCTIONS (SUBROUTINES - PROCEDURES - SUBFUNCTIONS - DEFUN’S)

```
(progn
  (defun volume(arg0 arg1 arg2)
  (values
          (* arg0 (* arg1 arg2))))


  (values
     (- (volume L0 W0 H0)
        (volume L1 W1 H1))))
```

AUTOMATICALLY DEFINED FUNCTIONS

TOP-DOWN VIEW OF THREE STEP HEIRARCHICAL PROBLEM-SOLVING PROCESS



Decompose

Solve subproblems

Solve original problem

Original problem

Subproblem 1

Subproblem 2

Solution to subproblem 1

Solution to subproblem 2

Solution to original probl

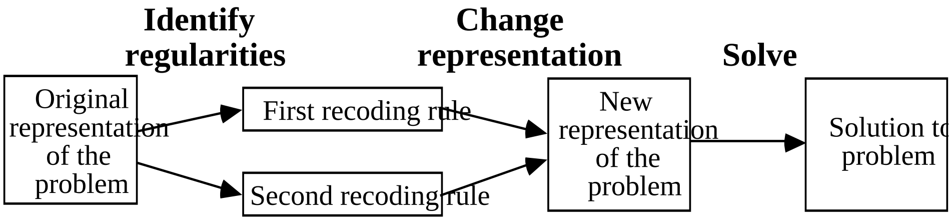
- **Decompose a problem into subproblems**

- **Solve the subproblems**

- **Assemble the solutions of the subproblems into a solution for the overall problem**

AUTOMATICALLY DEFINED FUNCTIONS

BOTTOM-UP VIEW OF THREE STEP HEIRARCHICAL PROBLEM-SOLVING PROCESS



Identify regularities

Change representation

Solve

Original representation of the problem

First recoding rule

Second recoding rule

New representation of the problem

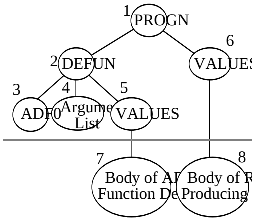
Solution to problem

- **Identify regularities**

- **Change the representation**

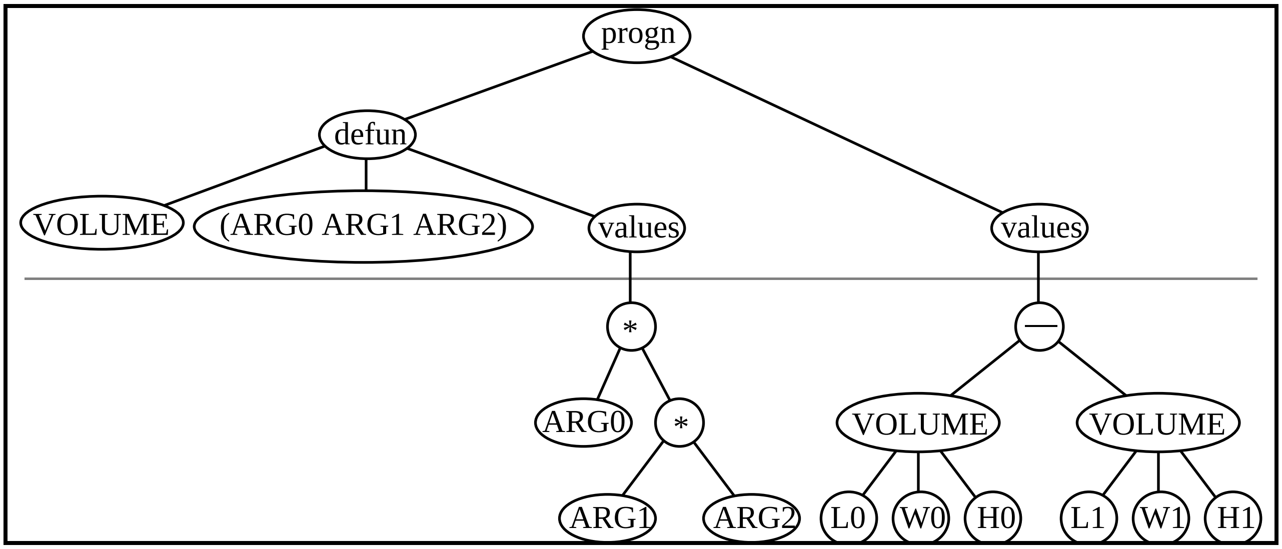
- **Solve the overall problem**

AN OVERALL COMPUTER PROGRAM CONSISTING OF ONE FUNCTION-DEFINING BRANCH AND ONE RESULT-PRODUCING BRANCH



100%-CORRECT PROGRAM FOR THE TWO-BOXES PROBLEM with ADFS

```
(progn
 (defun volume (arg0 arg1 arg2)
   (values
     (* arg0 (* arg1 arg2))))

(values  (- (volume L0 W0 H0)
            (volume L1 W1 H1))))
```



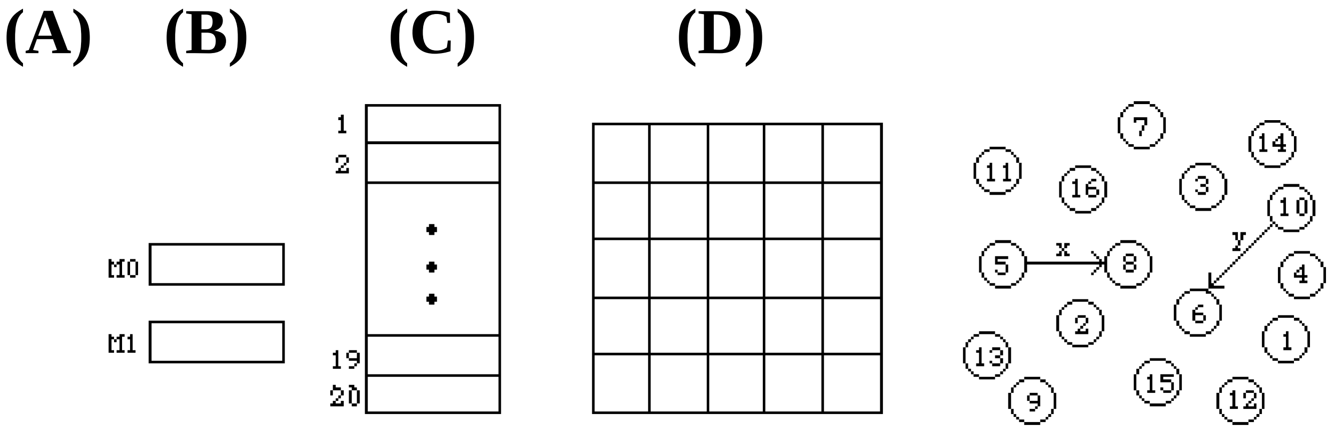
8 MAIN POINTS – *GENETIC PROGRAMMING II* BOOK

- **ADFs work.**
- **ADFs do not solve problems in the style of human programmers.**
- **ADFs reduce the computational effort required to solve a problem.**
- **ADFs usually improve the parsimony of the solutions to a problem.**
- **As the size of a problem is scaled up, the size of solutions increases more slowly with ADFs than without them.**
- **As the size of a problem is scaled up, the computational effort required to solve a problem increases more slowly with ADFs than without them.**
- **The advantages in terms of computational effort and parsimony conferred by ADFs increase as the size of the problem is scaled up.**

8 MAIN POINTS – *GENETIC PROGRAMMING II* BOOK

- **Genetic programming can evolve the architecture of the solution to a problem at the same time that it solves a problem.**

FOUR APPROACHES TO MEMORY AND STATE



- **(A) Settable variables (*Genetic Programming*) using terminals `M0` and `M1` and functions `(SETM0 X)` and `(SETM1 Y)`**

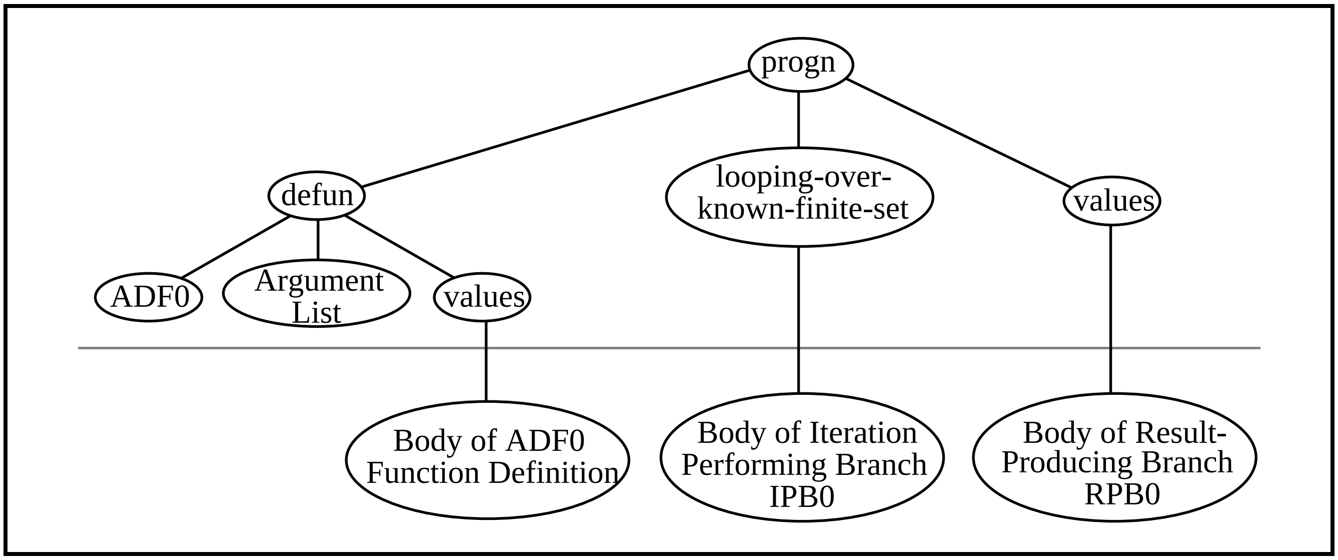
- **(B) Indexed memory (Teller) using `(READ K)` and`(WRITE X K)`**

- **(C) Memory isomorphic to world (Andre)**

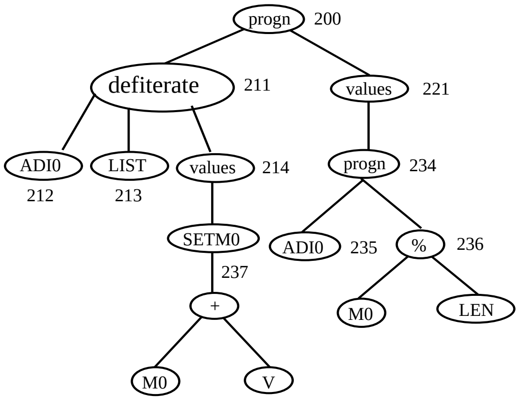
- **(D) Point-labeled, line-labeled directed graph for relational memory (Brave)**

AUTOMATICALLY DEFINED ITERATION (ADI)

- **Uses an iteration-performing branch `IPB0`**
- **Iteration is over a preestablished sequence, vector, list, two-dimensional matrix, etc.**
    - protein or DNA sequence
    - time sequence
    - two-dimensional arrangement of pixels
- **Overall program consisting of automatically defined function(s), iteration-performing branch(es), and a result-producing branch.**


progn
defun
looping-over-known-finite-set
values
ADF0
Argument List
values
Body of ADF0 Function Definition
Body of Iteration Performing Branch IPB0
Body of Result-Producing Branch RPB0

AUTOMATICALLY DEFINED ITERATION (ADI)



RESTRICTED ITERATION

```
(loop initially (progn (setf M0 0.0)
                       (setf.M1 0.0)
                       (setf M2 0.0)
                       (setf M3 0.0))
      for residue-index from 0
       below (length protein-segment)
      for residue =
       (aref protein-segment
             residue-index)
      do (eval IPB0)
      finally (return
              (wrapper (eval RPB))))
```

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

THE 446 RESIDUES OF D3DR_MOUSE

```
MAPLSQISSH INSTCGAENS TGVNRARPHA YYALSYCALI LAIIFGNGLV   50
CAAVLRERAL QTTTNYLVVS LAVADLLVAT LVMPWVVYLE VTGGVWNFSR  100
ICCDVFVTLD VMMCTASILN LCAISIDRYT AVVMPVHYQH GTGQSSCRRV  150
ALMITAVWVL AFAVSCPLLF GFNTTGDPSI CSISNPDFVI YSSVVSFYV   200
FGVTVLVYAR IYMVLRQRRR KRILTRQNSQ CISIRPGFPQ QSSCLRLHPI  250
RQFSIRARFL SDATGQMEHI EDKPYPQKCQ DPLLSHLQPL SPGQTHGELK  300
RYYSICQDTA LRHPNFEGGG GMSQVERTRN SLSPTMAPKL SLEVRKLSNG  350
RLSTSLKLGP LQPRGVPLRE KKATQMVVIV LGAFIVCWLP FFLTHVLNTH  400
CQACHVSPEL YRATTWLGYV NSALNPVIYT TFNIEFRKAF LKILSC      446
```

KYTE-DOOLITTLE HYDROPHOBICITY VALUES FOR THE 20 AMINO ACID RESIDUES

Category	Kyte-Doolittle value	One-letter code for amino acid	Amino acid	Three-letter code
Hydrophobic	+4.5	I	Isoleucine	Ile
Hydrophobic	+4.2	V	Valine	Val
Hydrophobic	+3.8	L	Leucine	Leu
Hydrophobic	+2.8	F	Phenylalanine	Phe
Hydrophobic	+2.5	C	Cysteine	Cys
Hydrophobic	+1.9	M	Methionine	Met
Hydrophobic	+1.8	A	Alanine	Ala
Neutral	–0.4	G	Glycine	Gly
Neutral	–0.7	T	Threonine	Thr
Neutral	–0.8	S	Serine	Ser
Neutral	–0.9	W	Tryptophan	Trp
Neutral	–1.3	Y	Tyrosine	Tyr
Neutral	–1.6	P	Proline	Pro
Hydrophilic	–3.2	H	Histidine	His
Hydrophilic	–3.5	Q	Glutamine	Gln
Hydrophilic	–3.5	N	Asparagine	Asn
Hydrophilic	–3.5	E	Glutamic Acid	Glu
Hydrophilic	–3.5	D	Aspartic Acid	Asp
Hydrophilic	–3.9	K	Lysine	Lys
Hydrophilic	–4.0	R	Arginine	Arg

SOME OF THE 246 IN-SAMPLE FITNESS CASES

Protein	Length	Number of TM domains	Length of chosen TM domain	Location of the chosen TM domain	Length of chosen non-TM segment	Chosen non-tTM area
3BH1_MOUSE	372	2	19	287–305	19	330–348
3BH3_MOUSE	372	2	19	287–305	19	330–348
5HT3_MOUSE	487	4	20	465–484	20	385–404
5HTE_MOUSE	366	7	25	24–48	25	235–259
A2AB_MOUSE	455	7	24	411–434	24	277–300
A4_MOUSE	770	1	24	700–723	24	736–759
ACE_MOUSE	1312	1	17	1265–1281	17	625–641
ACHB_MOUSE	501	4	19	277–295	22	391–412
ACHE_MOUSE	493	4	19	273–291	24	381–404
ACM1_MOUSE	460	7	23	25–47	23	277–299
AG2S_MOUSE	359	7	21	276–296	21	168–188
ANPA_MOUSE	1057	1	21	470–490	21	225–245
ATNC_MOUSE	290	1	28	40–67	28	7–34
AVRB_MOUSE	536	1	26	135–160	26	55–80
B2AR_MOUSE	418	7	23	107–129	24	363–386
B3AT_MOUSE	929	10	24	424–447	18	829–846
BASI_MOUSE	273	1	24	210–233	24	242–265
CADE_MOUSE	884	1	24	710–733	24	798–821
CADP_MOUSE	822	1	23	648–670	23	736–758
CD11_MOUSE	336	1	29	298–326	29	135–163

4 OUTCOMES FOR THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

$$\boxed{N_{fc} = N_{tp} + N_{tn} + N_{fp} + N_{fn}}$$

CORRELATION

$$\boxed{C = \frac{\sum_j \left(S_j - \overline{S}\right)\left(P_j - \overline{P}\right)}{\sqrt{\sum_j \left(S_j - \overline{S}\right)^2 \sum_j \left(P_j - \overline{P}\right)^2}}}$$

$$\boxed{C = \frac{N_{tp}N_{tn} - N_{fn}N_{fp}}{\sqrt{\left(N_{tn} + N_{fn}\right)\left(N_{tn} + N_{fp}\right)\left(N_{tp} + N_{fn}\right)\left(N_{tp} + N_{fp}\right)}}}$$

STANDARDIZED FITNESS

$$\boxed{\frac{1 - C}{2}.}$$

OVERALL PROGRAM FOR THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM CONSISTING OF AN AUTOMATICALLY DEFINED FUNCTION, ADF0, AN ITERATION-PERFORMING BRANCH, IPB0, AND A RESULT-PRODUCING BRANCH, RPB

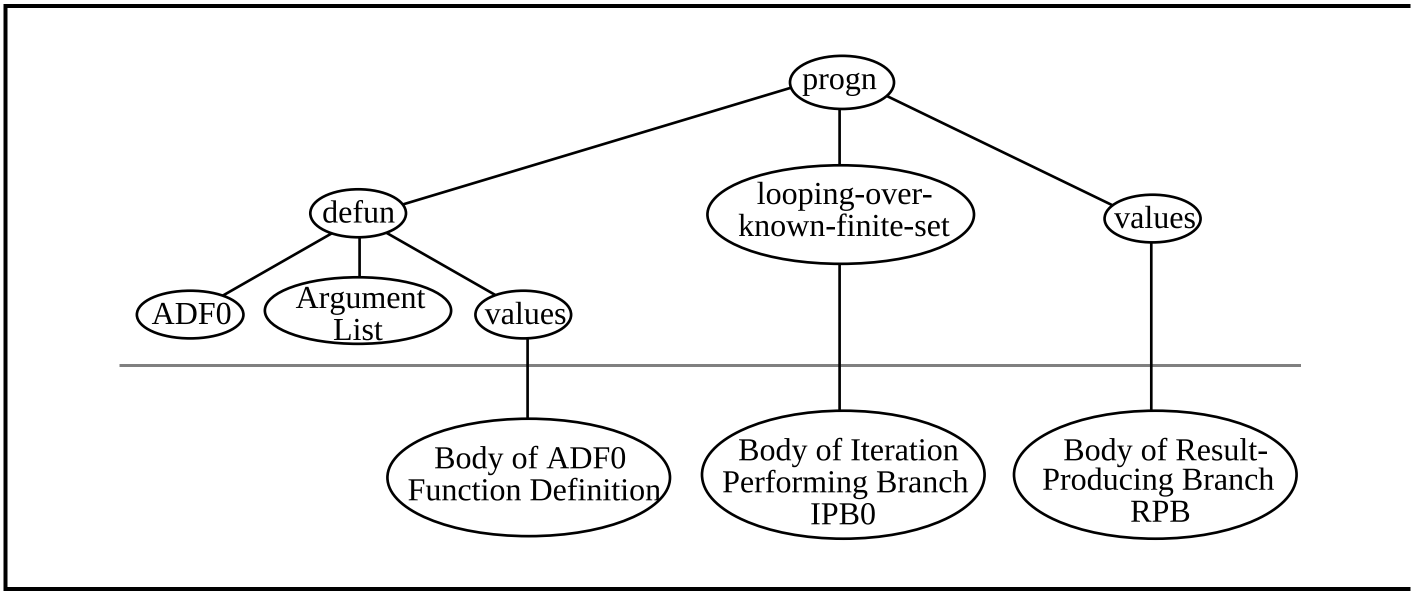

progn
defun
looping-over-known-finite-set
values
ADF0
Argument List
values
Body of ADF0 Function Definition
Body of Iteration Performing Branch IPB0
Body of Result-Producing Branch RPB

TABLEAU WITH ADFS

Objective:	**Find a program to classify whether or not a segment of a protein sequence is a transmembrane domain.**
Architecture of the overall program with ADFs:	**One result-producing branch, one iteration-performing branch, and three zero-argument function-defining branches, with no ADF hierarchically referring to any other ADF.**
Parameters:	**Branch typing for the three ADFs.**
Terminal set for the `IPB`:	**`LEN`, `M0`, `M1`, `M2`, `M3`, and the random constants $\leftarrow_{\text{bigger-reals}}$•**

Function set for the IPB:	ADF0, ADF1, ADF2, SETM0, SETM1, SETM2, SETM3, IFLTE, +, -, *, and %.
Terminal set for the result-producing branch:	LEN, M0, M1, M2, M3, and the random constants $\leftarrow_{\text{bigger-reals}}$.
Function set for the result-producing branch:	IFLTE, +, -, *, and %.
Terminal set for the function-defining branches ADF0, ADF1, and ADF2:	Twenty zero-argument functions (A?), (C?), ..., (Y?).

Function set for the function-defining branches `ADF0`, `ADF1`, and `ADF2`:	**Numerically valued two-argument logical disjunction function `ORN`.**

GENERATION 0 OF RUN 1 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS

- **in-sample correlation of 0.48**
- **a standardized fitness of 0.26**
- **99 true positives**
- **83 true negatives**
- **40 false positives**
- **24 false negatives**
- **out-of-sample correlation of 0.43**

```
(progn (defun ADF0 ()

(values (ORN (ORN (ORN (I?) (M?)) (ORN (V?) (C?)))
(ORN (ORN (W?) (L?)) (ORN (Y?) (A?))))))

(defun ADF1 ()

(values (ORN (ORN (ORN (L?) (L?)) (ORN (R?) (K?)))
(ORN (ORN (I?) (V?)) (ORN (R?) (Q?))))))

(defun ADF2 ()

(values (ORN (ORN (ORN (R?) (S?)) (ORN (F?) (Q?)))
(ORN (ORN (P?) (F?)) (ORN (Y?) (C?))))))

(progn (looping-over-residues
      (SETM0 (SETM3 (SETM0 (ADF0))))

(values (IFLTE (+ (- M3 M0) (+ M1 M3)) (% (IFLTE M0
M3 6.212 M1) (IFLTE M0 M2 M1 L)) (* (% M1 M2) (* M3
0.419)) (+ (% L M2) (- M0 M2)))))))
```

GENERATION 5 OF RUN 1 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS

- **in-sample correlation of 0.764**
- **out-of-sample correlation of 0.784**

```
(progn  (defun ADF0 ()

(values (ORN (ORN (I?) (A?)) (ORN (ORN
(L?) (G?)) (N?)))))

(defun ADF1 ()

(values (ORN (ORN (ORN (ORN (G?) (D?))
(ORN (E?) (V?))) (ORN (ORN (R?) (E?))
(ORN (T?) (P?)))) (ORN (N?) (S?)))))

(defun ADF2 ()

(values (ORN (ORN (ORN (L?) (R?)) (ORN
(V?) (P?))) (ORN (G?) (L?)))))

(progn (looping-over-residues
       (SETM1 (- (+ M1 (ADF0)) (ADF1))))

(values (* (% (+ (% -9.997 M3) M1) 6.602)
(+ 6.738 (% (- M3 L) (+ M3 M2)))))))
```

GENERATION 8 OF RUN 1 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS

- **in-sample correlation of 0.92**
- **out-of-sample correlation of 0.89**

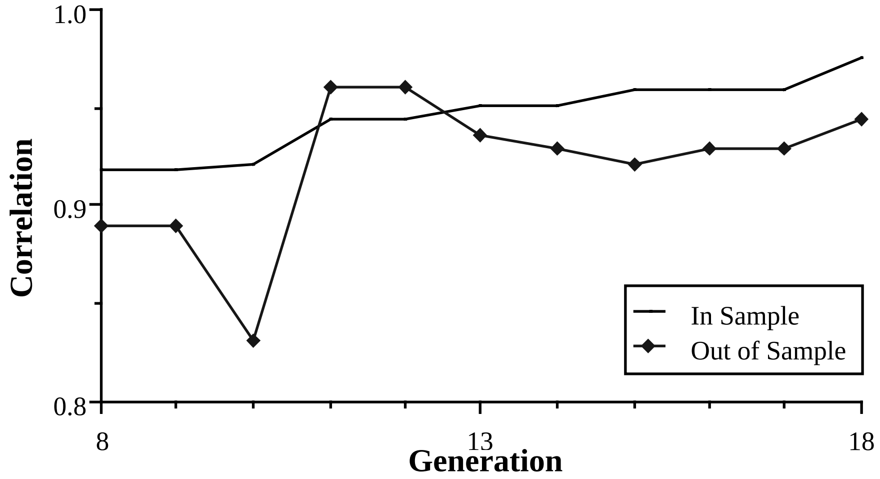
```
(progn  (defun ADF0 ()

(values (ORN (ORN (ORN (I?) (M?)) (ORN
(V?) (C?))) (ORN (ORN (L?) (G?)) (N?)))))

(defun ADF1 ()

(values (ORN (ORN (ORN (ORN (G?) (D?))
(ORN (E?) (V?))) (ORN (ORN (R?) (E?))
(ORN (T?) (P?)))) (ORN (N?) (S?)))))

(defun ADF2 ()

(values (ORN (ORN (ORN (L?) (R?)) (ORN
(V?) (P?))) (ORN (G?) (L?)))))

(progn (looping-over-residues
       (SETM1 (- (+ M1 (ADF0)) (ADF1))))

(values (* (+ M1 M3) (+ 6.738 (% (- M3 L)
(+ M3 M2)))))))
```

GENERATION 11 OF RUN 1 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS

- **in-sample correlation of 0.94**
- **standardized fitness of 0.03**
- **out-of-sample correlation of 0.96**
- **122 true positives — • 123 true negatives**
- **2 false positives — • 3 false negatives**
- **out-of-sample error rate 2.0%**

```
(progn (defun ADF0 ()

(values (ORN (ORN (ORN (I?) (M?)) (ORN (V?) (C?)))
(ORN (ORN (L?) (G?)) (N?)))))

(defun ADF1 ()

(values (ORN (ORN (ORN (ORN (G?) (D?)) (ORN (E?)
(V?))) (ORN (ORN (R?) (E?)) (ORN (ORN (ORN (ORN (G?)
(D?)) (ORN (E?) (V?))) (ORN (ORN (R?) (K?)) (ORN (T?)
(P?)))) (ORN (N?) (S?))))) (ORN (N?) (S?)))))

(defun ADF2 ()

(values (ORN (ORN (ORN (L?) (Y?)) (ORN (V?) (P?)))
(ORN (G?) (L?)))))

(progn (looping-over-residues
       (SETM1 (- (+ M1 (ADF0)) (ADF1))))

(values (* (+ M1 M3) (+ 6.738 (% (- M3 L) (+ M3
M2)))))))
```

COMPARISON OF VALUES OF IN-SAMPLE AND OUT-OF-SAMPLE CORRELATION FOR RUN 1 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS



GENERATION 20 OF RUN 3 FOR THE SUBSET-CREATING VERSION OF THE TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ADFS

- **in-sample correlation of 0.976**
- **out-of-sample correlation of 0.968**
- **out-of-sample error rate 1.6%**

```
(progn (defun ADF0 ()

(values (ORN (ORN (ORN (I?) (H?)) (ORN (P?) (G?)))
(ORN (ORN (ORN (Y?) (N?)) (ORN (T?) (Q?))) (ORN (A?)
(H?))))))

(defun ADF1 ()

(values (ORN (ORN (ORN (A?) (I?)) (ORN (L?) (W?)))
(ORN (ORN (T?) (L?)) (ORN (T?) (W?))))))

(defun ADF2 ()

(values (ORN (ORN (ORN (ORN (ORN (D?) (E?)) (ORN (ORN
(ORN (D?) (E?)) (ORN (ORN (T?) (W?)) (ORN (Q?)
(D?)))) (ORN (K?) (P?)))) (ORN (K?) (P?))) (ORN (T?)
(W?))) (ORN (ORN (E?) (A?)) (ORN (N?) (R?))))))

(progn (loop-over-residues
       (SETM0 (+ (- (ADF1) (ADF2)) (SETM3 M0))))

(values (% (% M3 M0) (% (% (% (- L -0.53) (* M0 M0))
(+ (% (% M3 M0) (% (+ M0 M3) (% M1 M2))) M2)) (% M3
M0))))))
```

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBELM WITH ITERATION CREATION OPERATION

PREPARATORY STEPS

INITIAL FUNCTIONS AND TERMINALS

T_{initial} = {←, **M0**, **M1**, **M2**, **M3**, **M4**, **M5**, **LEN**, **(A?)**, **(C?)**, **…**, **(Y?)**}

F_{initial} = {**+**, **-**, *****, **%**, **IFGTZ**, **ORN**, **SETM0**, **SETM1**, **SETM2**, **SETM3**, **SETM4**, **SETM5**}

POTENTIAL FUNCTIONS AND TERMINALS

T**potential** = {**IPB0**, **IPB1**, **IPB2**, **ARG0**, **ARG1**, **ARG2**, **ARG3**}

The set of potential additional functions, $F_{\text{potential}}$, for this problem consists of

F**potential** = {**ADF0**, **ADF1**, **ADF2**, **ADF3**}

PARAMETERS

- **Populaion size *M* = 64,000**
- **The percentage of operations on each generation after generation 6:**
- **85% crossovers**
- **10% reproductions**
- **0% mutations**
- **1% restricted iteration creations**
- **1% branch duplications**
- **1% argument duplications**
- **0.5% branch deletions**
- **0.5% argument deletions**
- **1% branch creations**
- **0% argument creations**

PARAMETERS

- The percentage of operations on each generation after generation 6:
- 70% crossovers
- 10% reproductions
- 0% mutations
- 6% restricted iteration creations
- 2% branch duplications
- 2% argument duplications
- 2% branch deletions
- 2% argument deletions
- 6% branch creations
- 0% argument creations

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

THE MYOPIC PERFORMANCE OF THE BEST OF GENERATION 0 (CORRELATION OF 0.3108)

```
(setm2 (* (setm5 (setm0 (orn
LEN M0))) (* (* (setm4 LEN)
(setm4 (M?))) (% (setm1 (W?))
(setm4 (V?)))))))
```

A MYOPIC ITERATION-PERFORMING BRANCH FROM GENERATION 1 (CORRELATION OF 0.4702)

- classification of the entire protein segment is myopically done on the basis of just the last residue from the protein segment

AN ITERATION-PERFORMING BRANCH THAT GLOBALLY INTEGRATES INFORMATION

- **Result-producing branch, `RPB`, is**

```
(orn (IPB0) (L?))
```

- **Iteration-performing branch, `IPB0`, is**

```
(% (setm3 (orn (K?) M3)) (E?))
```

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

AN ITERATION-PERFORMING BRANCH THAT COMPUTES A CONVENTIONAL RUNNING SUM

- **Result-producing branch of first pace-setting program from generation 2 (correlation of 0.7224) is just `(IPB0)`**
- **Iteration-performing branch, `IPB0`, is**

```
(setm3 (+ (* (H?) (E?)) (+ (V?)
M3)))
```

- **+1 in contributed by each hydrophobic `V` residue (+4.2 on the Kyte-Dolittle scale), +1 is contributed by each residue that is neither `E` (–3.5 on the Kyte-Dolittle scale) nor `H` (–3.2 on the Kyte-Dolittle scale), and -1 is contributed by either an `E` or a `H`**

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

EMERGENCE OF AUTOMATICALLY DEFINED FUNCTIONS

- Pace-setting program from generation 6 contains both a one-argument automatically defined function as well as an iteration-performing branch

Emergence of Multiple Iteration-Performing Branches

- First pace-setting program from generation 8 has multiple iteration-performing branches. One of these iteration-performing branches globally integrates information over the entire protein segment.

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

EMERGENCE OF COOPERATIVITY AMONG ITERATION-PERFORMING BRANCHES

- **First iteration-performing branch, `IPB0`, of second pace-setting program from generation 11 is**

```
(setm3 (+ (* (H?) (E?)) (+ (orn
(setm2 M0) (set2 (W?))) M3)))
```

- **`IPB0`, computes a running sum, `M3`. An increment of +1 is contributed by `W` (tryptophan); +1 is contributed by each residue that is neither `E` nor `H`; and -1 is contributed by either an `E` or a `H` (histidine).**

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

EMERGENCE OF COOPERATIVITY AMONG ITERATION-PERFORMING BRANCHES

• Second iteration-performing branch, `IPB1`, makes an additional contribution to `M3` based on `H`, `E`, and `V` (valine) as follows:

```
   (setm3 (+ (* (H?) (E?)) (+
 (V?) M3)))
```

• Result-producing branch is simply `(IPB1)`. Its value is the running sum to which +1 is contributed by each `V`; +1 is contributed by each `W`; +2 is contributed by each residue that is neither `E` nor `H`; and -2 is contributed by either an `E` or a `H`.

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

EMERGENCE OF HIERARCHY AMONG AUTOMATICALLY DEFINED FUNCTIONS

*** A pace-setting program from generation 24 has two automatically defined functions (a one-argument `ADF1` and a zero-argument `ADF3`) such that `ADF3` refers to `ADF1` (and also to `IPB1`).**

EMERGENCE OF MULTIPLE AUTOMATICALLY DEFINED FUNCTIONS AND MULTIPLE ITERATION-PERFORMING BRANCHES

- **The pace-setting program from generation 26 has three one-argument automatically defined functions as well as two iteration-performing branches.**

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

BEST-OF-RUN PROGRAM FROM GENERATION 42

- Best-of-generation program scores 122 true positives, 122 true negatives, 1 false positive, and 1 false negative and has an in-sample correlation of 0.9938. It has an out-of-sample error rate of 1.6%.
- This program has two one-argument automatically defined functions (`ADF0` and `ADF1`) and two iteration-performing branches (`IPB0` and `IPB1`) that cooperatively integrate global information.
- The result-producing branch is `(IPB1)`

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

BEST-OF-RUN PROGRAM FROM GENERATION 42

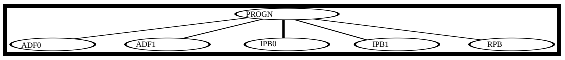
- **`ADF0` is**

```
(adf1 (+ (setm0 (E?))(setm4
(Q?))))
```

Since `ADF1` merely returns its one argument, `ADF0` returns 0 if the current residue is `E` or `Q` (glutamine) and otherwise returns –2 (as well as side-effecting the settable variables `M0` and `M4`).

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

BEST-OF-RUN PROGRAM FROM GENERATION 42 – CONTINUED



- **First iteration-performing branch, `IPB0`:**

```
(setm1 (– (– (setm1 (setm1 (– (setm1 M1)
(setm3 (setm3 (% (– (I?) (R?)) (adf0
(H?)))))))) (setm3 (setm3 (% (– (+ (V?)
M3) (setm2 (+ (– (D?) (+ (V?) (setm3 (+
(orn (Y?) (* (E?) (setm5 (orn (P?)
(D?)))))(+ (setm5 (orn M0 (L?))) M3)))))
(setm3 (R?))))) (adf0 (% (setm1 (– (–
(setm1 (setm1 (– (setm1 M1) (setm3 (setm3
(% (– (I?) (R?)) (adf0 (H?)))))))) (setm3
(setm3 (% (– (+ (V?) M3) (setm2 (+ (– (*
(setm5 (orn (P?) (R?))) (setm5 (orn (P?)
(D?)))) (L?)) (setm3 (orn (Q?) (%  M5
(V?))))))) (setm5 (orn  M0 (L?)))))))
(setm3 (setm3 (% (– (F?) (R?))(adf0
(H?))))))) (E?))))))) (setm3 (setm3 (% (–
(F?) (R?))(adf0 (H?)))))))
```

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

BEST-OF-RUN PROGRAM FROM GENERATION 42 – CONTINUED

- Second iteration-performing branch, **IPB1**:

```
(setm1 (– (setm1 M1) (setm3 (setm3 (% (–
(I?) (adf1 (* (setm0 (setm1 (orn (orn
(P?) (R?)) (– (setm1 M1) (setm3 (setm3
(ifgtz (setm4 (– (Y?) (R?))) (setm1 (Y?))
IPB0)))))))) (setm0 (* (setm0 (orn (K?)
M0)) (setm1 (orn (setm4 (setm1 (setm4
(P?)))) (Q?))))))))) (adf0 (H?)))))))
```

- **Result-producing branch returns the value returned by the second iteration-performing branch, IPB1.**
- **Automatically defined function, ADF0:**

```
(adf1 (+ (setm0 (E?))(setm4
(Q?))))
```

- **ADF1 merely returns its one argument.**

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM WITH ITERATION CREATION

BEST-OF-RUN PROGRAM FROM GENERATION 42 – CONTINUED

- **Both possible avenues of communication and cooperation are employed by this program.**
  - First, two of the six settable variables (`M0` and `M1`) are set in `IPB0` and referenced by `IPB1` (as highlighted by bold-faced type in `IPB1`).

  - Second, `IPB1` contains a reference to the value returned by `IPB0` (also highlighted by bold-faced type in `IPB1`).

COMPARISON OF EIGHT METHODS FOR SOLVING TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

Method	Error
von Heijne 1992	2.8%
Engelman, Steitz, and Goldman 1986	2.7%
Kyte and Doolittle 1982	2.5%
Weiss, Cohen, and Indurkhya 1993	2.5%
GP + Set-creating ADFs	1.6%
GP + Arithmetic-performing ADFs	1.6%
GP + ADFs + six architecture-altering operations	1.6%
GP + ADFs + six architecture-altering operations + restricted iteration creation operation	1.6%

AUTOMATICALLY DEFINED LOOP (ADL)

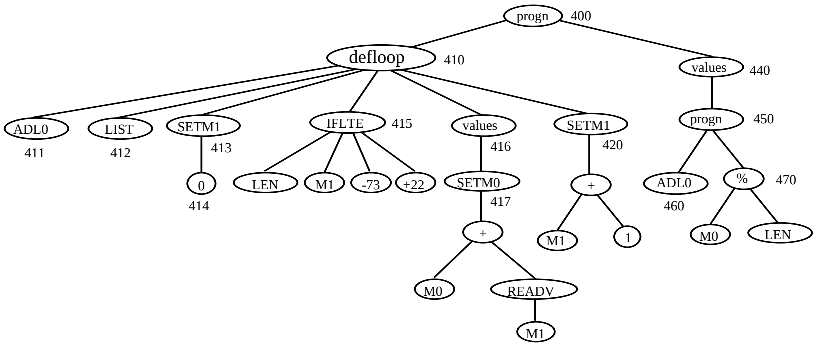
- **four distinct branches, namely**
  - a loop initialization branch, `LIB`,
  - a loop condition branch, `LCB`,
  - a loop body branch, `LBB`, and
  - a loop update branch, `LUB`.
- **Iterative `for` loop in the C programming language:**

```
for (i = 0; i < LEN; i++)
{
  M0 = M0 + V[i];
}
```

- **Using the ADL terminology for `LIB`, `LCB`, `LBB`, and `LUB`, a `for` loop in C would be written as**

```
for (LIB; LCB; LUB)
{
  LBB;
}
```

AUTOMATICALLY DEFINED LOOP (ADL)



- **Initialization and iterative `for` loop in the C programming language:**

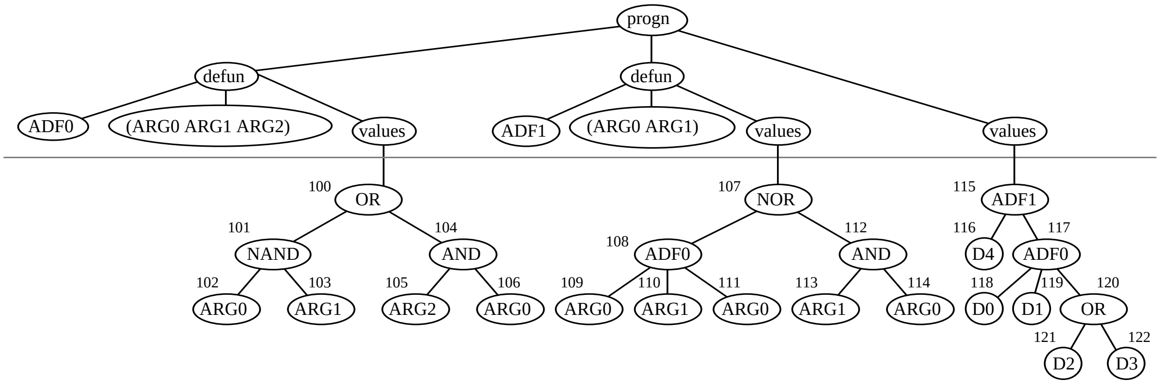
```
M0 = 0;
for (i = 0; i < LEN; i++)
{
  M0 = M0 + V[i];
}
```

AUTOMATICALLY DEFINED FUNCTIONS

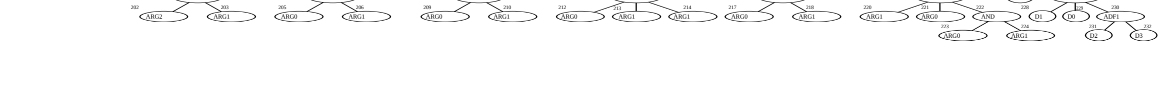
EVOLUTIONARY SELECTION OF THE ARCHITECTURE

POINT TYPING FOR STUCTURE-PRESERVING CROSSOVER

Parent A with an argument map of {3, 2}



Parent B with an argument map of {{3, 2, 2}

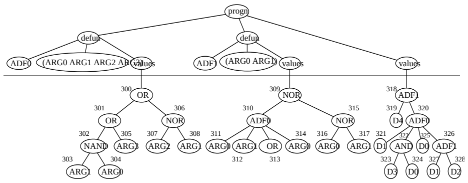


AUTOMATICALLY DEFINED FUNCTIONS

EVOLUTIONARY SELECTION OF THE ARCHITECTURE

POINT TYPING FOR STUCTURE-PRESERVING CROSSOVER

Parent C with an argument map of {4, 2}



GENE DUPLICATION

- **Fly (midge)*Chironomus tentans* (Galli and Wislander 1993)**
- **3,959-bases of DNA with accession number X70063 in GenBank**
- **One subsequence of 732 bases (called "C. tentans Sp38–40.A gene") are in DNA positions positions 918–1,649 and is expressed as protein of length 244**
- **A second subsequence of 759 bases (called "C. tentans Sp38–40.B gene") are in DNA positions 2,513–3,271 and is expressed as protein of length 253.**
- **Both proteins are secreted from the salivary gland of the insect and form water-insoluble fibers which are spun into one of two kinds of tubes – one for larval protection and feeding and one for pupation**

PROTEIN ALIGNMENT OF THE "A" AND "B" PROTEINS

```
First.protein     MRIKFLVVLA VICLFAHYAS ASGMGGDKKP KDAPKPKDAP KPKEVKPVKA     50
Second.protein    MRIKFLVVLA VICLLAHYAS ASGMGGDKKP KDAPKPKDAP KPKEVKPVKA     50

First.protein     ESSEYEIEVI KHQKEKTEKK EKEKKTHVET KKEVKKKEKK QIPCSEKLKD    100
Second.protein    DSSEYEIEVI KHQKEKTEKK EKEKKAHVEI KKKIKNKEKK FVPCSEILKD    100

First.protein     EKLDCETKGV PAGYKAIFKF TENEE-CDWT CDYEALPPPP GAKKDDKKEK    149
Second.protein    EKLECEKNAT P-GYKALFEF KESESFCEWE CDYEAI---P GAKKDEKKEK    146

First.protein     KTVKVVKPPK EKPPKKLRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKNFD    199
Second.protein    KMVKVIKPPK EKPPKKPRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKNFD    196

First.protein     KKFAKLVQGK QKKGAKKAKG GKKAAPKPGP KPGPK----Q ADKP------    239
Second.protein    KKFAKLVQGK QKKGAKKAKG GKKAEPKPGP KPAPKPGPKP APKPVPKPAD    246

First.protein     --KDAKK                                                   244
Second.protein    KPKDAKK                                                   253
```

NEW ARCHITECTURE-ALTERING OPERATORS

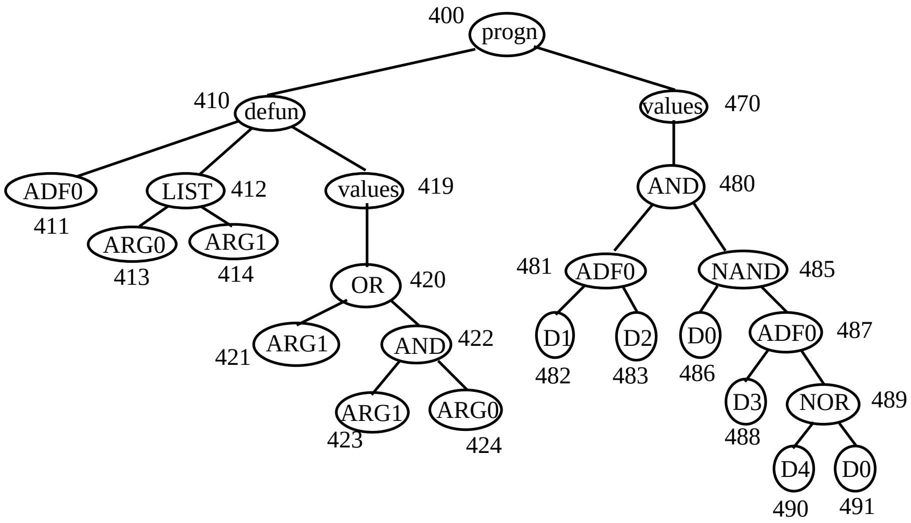
SPECIALIZATION / REFINEMENT / CASE SPLITTING

- Branch duplication
- Argument duplication
- Branch creation
- Argument creation

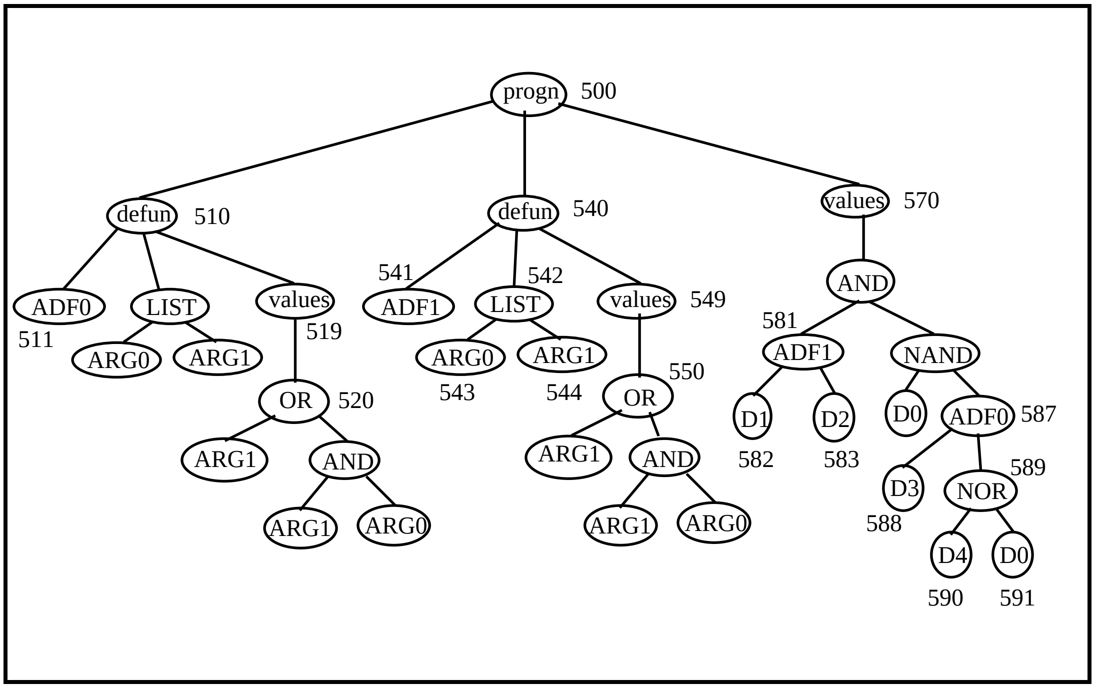
GENERALIZATION

- Branch deletion
- Argument deletion

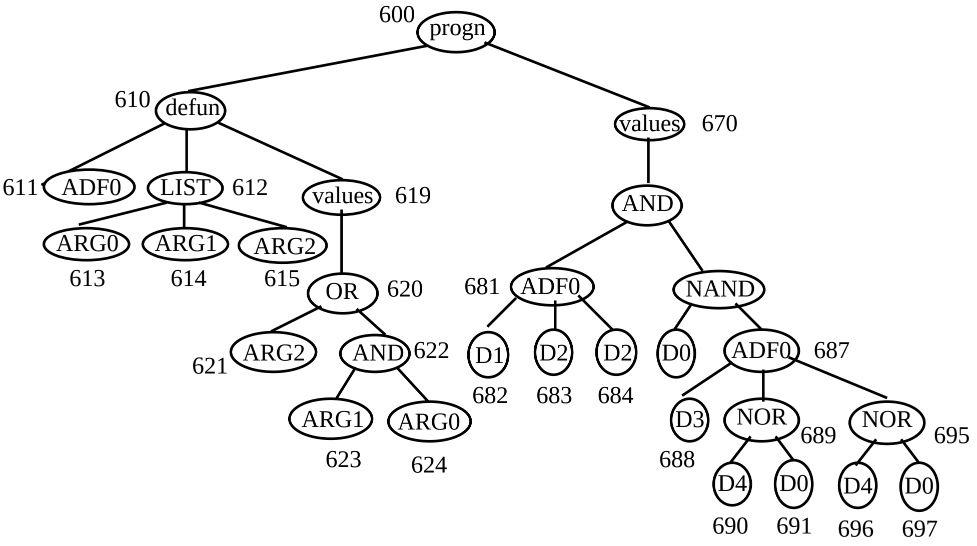
PROGRAM WITH 1 TWO-ARGUMENT AUTOMATICALLY DEFINED FUNCTION (ADF0) AND 1 RESULT-PRODUCING BRANCH – ARGUMENT MAP OF {2}



PROGRAM WITH ARGUMENT MAP OF {2, 2} CREATED USING THE OPERATION OF BRANCH DUPLICATION



PROGRAM WITH ARGUMENT MAP OF {3) CREATED USING THE OPERATION OF ARGUMENT DUPLICATION



PARALLELIZATION OF GA OR GP

- **By fitness cases**
  - Timing (Simulation time, Protein length)
  - Matching between hardware and problem

- **By individuals**
  - Timing (Program size, Simulation time)

- **By runs**
  - Assumes adequacy of population size of a run

- **Demes ("Island" model)**
  - No synchronization of islands
  - Occasional small amounts of migration (low band width requirement for communication)
  - Emigrants go (fitness-based selection)
  - Immigrants arrive and are absorbed (fitness-based making of space)
  - Fault-tolerant

GP APPLIED TO MOLECULAR BIOLOGY

• Handley, Simon. Automated learning of a detector for a-helices in protein sequences via genetic programming. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Ed. Stephanie Forrest. San Mateo, CA: Morgan Kaufmann Publishers, 1993. 271-278.

• Handley, S. 1994. Automated learning of a detector for the cores of a-helices in protein sequences via genetic programming. *Proceedings of the First IEEE Conference on Evolutionary Computation*. IEEE Press, 1994. 1: 474–479.

• Handley, Simon. The prediction of the degree of exposure to solvent of amino acid residues via genetic programming. *Proceedings of the Second International*

Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press, 1994.

GP BOOKS AND VIDEOTAPES

• Blickle, Tobias. 1997. *Theory of Evolutionary Algorithms and Application to System Synthesis*. TIK-Schriftenreihe Nr. 17. Zurich, Switzerland: vdf Hochschul Verlag AG and der ETH Zurich. ISBN 3-7281-2433-8.

• Iba, Hitoshi. 1996. *Genetic Programming*. Tokyo: Tokyo Denki University Press. In Japanese.

• Jacob, Christian. 1997. *Principia Evolvica: Simulierte Evolution mit Mathematica*. Heidelberg, Germany: dpunkt.verlag. In German. English translation forthcoming.

• Koza, John R. *Genetic Programming: On Programming Computers by Means of Natural Selection*. Cambridge, MA: MIT Press 1992.

• Koza, John R. and Rice, James P. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press 1992. (VHS NTSC, PAL, SECAM)

GP BOOKS – CONTINUED

- **Koza, John R.** ***Genetic Programming II: Automatic Discovery of Reusable Programs.*** **Cambridge, MA: MIT Press 1994.**
- **Koza, John R.** ***Genetic Programming II Videotape*****:** ***The Next Generation*****. Cambridge, MA: MIT Press 1994. (VHS in NTSC, PAL, SECAM)**
- **Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin A. 1998.** ***Genetic Programming III*****. San Francisco, CA: Morgan Kaufmann.**
- **Langdon, William B. 1998.** ***Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*** **Amsterdam: Kluwer.**
- **Nordin, Peter. 1997.** ***Evolutionary Program Induction of Binary Machine Code and its Application*****. Munster, Germany: Krehl Verlag.**

GP CONFERENCE AND EURO-GP WORKSHOP PROCEEDINGS

• Banzhaf, Wolfgang, Poli, Riccardo, Schoenauer, Marc, and Fogarty, Terence C. 1998. *Genetic Programming: First European Workshop. EuroGP'98. Paris, France, April 1998 Proceedings. Paris, France. April l998*. Lecture Notes in Computer Science. Volume 1391. Berlin, Germany: Springer-Verlag.

• Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

• Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13–16, 1997,*

Stanford University. San Francisco, CA: Morgan Kaufmann.

- Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoym Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). 1998. *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin*. San Francisco, CA: Morgan Kaufmann.

ADVANCES IN GENETIC PROGRAMMING SERIES (MIT PRESS)

- **Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996.** ***Advances in Genetic Programming 2*****. MIT Press.**
- **Kinnear, Kenneth E. Jr. (editor).** ***Advances in Genetic Programming*****. Cambridge, MA: MIT Press 1994.**
- **Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999.** ***Advances in Genetic Programming 3*****. Cambridge, MA: The MIT Press.**

KLUWER BOOK SERIES ON GENETIC PROGRAMMING

- **Langdon, William B. 1998.** ***Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*** **Amsterdam: Kluwer.**

GENERAL BOOKS ON GENETIC ALGORITHMS

• Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley l989.

• Holland, John H. *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press 1975. Now available as 2nd edition from The MIT Press 1992.

• Davis, Lawrence (editor). *Genetic Algorithms and Simulated Annealing* London: Pittman l987.

• Davis, Lawrence. *Handbook of Genetic Algorithms* Van Nostrand Reinhold.1991.

• Michalewicz, Zbignlew. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag 1992.

• Mitchell, Melanie. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: The MIT Press.

SPECIFIC APPLICATION AREAS OF GA

- **Albrecht, R. F., Reeves, C. R., and Steele, N. C. 1993. *Artificial Neural Nets and Genetic Algorithms*. Springer-Verlag.**
- **Bauer, Richard J., Jr. *Genetic Algorithms and Investment Strategies*. John Wiley. 1994.**
- **Bhanu, Bir and Lee, Sungkee. 1994. *Genetic Learning for Adaptive Image Segmentation*. Boston: Kluwer Academic Publishers.**
- **Buckles Bill P. and Petry, Frederick E. *Genetic Algorithms*. Los Alamitos, CA: The IEEE Computer Society Press. 1992.**
- **Chambers, Lance D. 1995. *Practical Handbook of Genetic Algorithms - Volume 1*. Boca Raton, FL: CRC Press.**
- **Davidor, Yuval. *Genetic Algorithms and Robotics*. Singapore: World Scientific 1990.**

SPECIFIC APPLICATION AREAS OF GA

- **Pal, Sankar K. andWang, Paul GP. Wang. 1996.** ***Genetic Algorithms and Pattern Recognition*****. Boca Raton, FL: CRC Press.**
- **Stender, J. (editor). 1993.** ***Parallel Genetic Algorithms*****. IOS Publishing.**
- **Stender, J., Hillebrand, and Kingdon, J. (editors). 1994.** ***Genetic Algorithms in Optimization, Simulation, and Modeling*****. Amsterdam: IOS Publishing.**

GP E-MAIL LISTS

GENETIC PROGRAMMING (GP) LIST

• To subscribe, send e-mail message to:

```
Genetic-Programming-
Request@CS.Stanford.Edu
```

• Be sure to send to exactly this address, (which includes the word "Request")!

• The BODY of your message must consist of exactly the words:

```
subscribe genetic-programming
```

VARIOUS E-MAIL LISTS

GENETIC ALGORITHM (GA) LIST

GA-List-Request@AIC.NRL.NAVY.MIL

INDUCTIVE LOGIC PROGRAMMING (ILP)

To subscribe, send e-mail message to:
ilpnet@ijs.si
with a SUBJECT heading of:
SUBSCRIBE ILPNEWS

MACHINE LEARNING LIST

ML-Request@ICS.UCI.EDU

GP FTP SITE

An on-line public repository and FTP site and WWW site containing computer code, papers on genetic programming, and frequently asked questions (FAQs) may be accessed by electronic mail by anonymous FTP from the

`pub/genetic-programming`

directory from the site

`ftp.io.com`

The URL of the WWW site is

`ftp://ftp.io.com/pub/genetic-programming`

GA ARCHIVE – FTP AND WWW SITE

- **An FTP and WWW site for the "GA archive" can be accessed through anonymous ftp at `ftp.aic.nrl.navy.mil` [192.26.18.68] in `/pub/galist`. The URL for the WWW pages is `http://www.aic.nrl.navy.mil/galist/`**
Contains genetic algorithm code, conference announcements for the GA field, back issues of the Genetic Algorithms Digest e-mail newsletter

WWW SITE FOR GENETIC PROGRAMMING CONFERENCES INC

www.genetic-programming.org

JOHN KOZA'S HOME PAGE

http://www.smi.stanford.edu/people/koza

Contains

- Information on GP-96, GP-97, GP-98 conferences
- Links to people doing GP research
- List of PhD theses in progress
- Links to many other GP resources
- Abstracts of JK's publications
- Links to other GP WWW pages
- Links to Langdon's complete GP bibliography