INTRODUCTION TO GENETIC PROGRAMMING

TUTORIAL

GECCO-2005—WASHINGTON SATURDAY JUNE 25, 2005

John R. Koza
Consulting Professor (Medical Informatics)
Department of Medicine
School of Medicine

Consulting Professor
Department of Electrical Engineering
School of Engineering

Stanford University
Stanford, California 94305
E-MAIL: koza@stanford.edu
http://www.smi.stanford.edu/people/koza/
http://www.genetic-programming.org
http://www.genetic-programming.com

THE CHALLENGE

"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 (1959)

CRITERION FOR SUCCESS

"The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."

― Arthur Samuel (1983)

MAIN POINTS

• Genetic programming now routinely delivers high-return human-competitive machine intelligence.
• Genetic programming is an automated invention machine.
• Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.

VARIOUS REPRESENTATIONS USED TO TRY TO ACHIEVE ARTIFICIAL INTELLIGENCE (AI) AND MACHINE LEARNING (ML)

• Decision trees
• If-then production rules (e.g., expert systems)
• Horn clauses
• Neural nets (matrices of numerical weights)
• Bayesian networks
• Frames
• Propositional logic
• Binary decision diagrams
• Formal grammars
• Numerical coefficients for polynomials
• Tables of values (reinforcement learning)
• Conceptual clusters
• Concept sets
• Parallel if-then rules (e.g., learning classifier systems)
**A COMPUTER PROGRAM**

![Diagram of a computer program with inputs and outputs and potential subroutines, loops, recursions, and storage]

**REPRESENTATION**

- "Our view is that computer programs are the best representation of computer programs."

**COMPUTER PROGRAM**

=PARSE TREE=PROGRAM TREE

=PROGRAM IN LISP=DATA=LIST

\((+ 1 2 (\text{IF} (> \text{TIME} 10) 3 4))\)

- Terminal set \(T = \{1, 2, 10, 3, 4, \text{TIME}\}\)
- Function set \(F = \{+, \text{IF}, >\}\)

**EXAMPLE OF RANDOM CREATION OF A PROGRAM TREE**

- Terminal set \(T = \{A, B, C\}\)
- Function set \(F = \{+, -, *, \%, \text{IFLTE}\}\)

**BEGIN WITH TWO-ARGUMENT +**

**CONTINUE WITH TWO-ARGUMENT ***

**FINISH WITH TERMINALS A, B, AND C**

- The result is a syntactically valid executable program (provided the set of functions is closed)
**MUTATION OPERATION**

- Select parent probabilistically based on fitness
- Pick point from 1 to \( \text{NUMBER-OF-POINTS} \)
- Delete subtree at the picked point
- Grow new subtree at the mutation point in same way as generated trees for initial random population (generation 0)
- The result is a syntactically valid executable program

**ONE PARENTAL PROGRAM**

```
1  2  3  4
5  6  7  8
```

**OFFSPRING PRODUCED BY MUTATION**

```
1  2  3  4
5  6  7  8
```

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

- Select two parents probabilistically based on fitness
- Randomly pick a number from 1 to \( \text{NUMBER-OF-POINTS} \)
  - independently for each of the two parental programs
- Identify the two subtrees rooted at the two picked points

Parent 1:

\[
(+ (\ast 0.234 Z) (- X 0.789))
\]

Parent 2:

\[
(* (\ast Z Y) (+ Y (* 0.314 Z)))
\]

**THE CROSSOVER OPERATION (TWO OFFSPRING VERSION)**

```
Y + 0.314Z + X - 0.789
0.234Z^Y
```

Offspring 1:

\[
(+ (+ Y (* 0.314 Z)) (- X 0.789))
\]

Offspring 2:

\[
(* (* Z Y) (* 0.234 Z))
\]

- The result is a syntactically valid executable program

**FIVE MAJOR PREPARATORY STEPS FOR GP**

- Determining the set of terminals
- Determining the set of functions
- Determining the fitness measure
- Determining the parameters for the run
  - population size
  - number of generations
  - minor parameters
- Determining the method for designating a result and the criterion for terminating a run
Objective:

Find a computer program with one input (independent variable $x$), whose output equals the value of the quadratic polynomial $x^2 + x + 1$ in range from -1 to +1.

1 Terminal set: $T = \{X, \text{Constants}\}$

2 Function set: $F = \{+, -, *, \%\}$

NOTE: The protected division function $\%$ returns a value of 1 when division by 0 is attempted (including 0 divided by 0).

3 Fitness: The sum of the absolute value of the differences (errors), computed (in some way) over values of the independent variable $x$ from -1.0 to +1.0, between the program’s output and the target quadratic polynomial $x^2 + x + 1$.


5 Termination: An individual emerges whose sum of absolute errors is less than 0.1

---

SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $x^2 + x + 1$

INITIAL POPULATION OF FOUR RANDOMLY CREATED INDIVIDUALS OF GENERATION 0

(x) (y) (z) (w)

$X + 1$ $X^2 + 1$ 2 $X$

FITNESS

0.67 1.00 1.70 2.67

---

SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $x^2 + x + 1$

GENERATION 1

(x) (y) (z) (w)

$X + 1$ 1 $X$ $x^2 + x + 1$

Copy of (a) Mutant of (c) First Second
—picking “2” offspring of offspring of crossover of crossover of
mutation as (a) and (b) (a) and (b)
point

---

SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $x^2 + x^3 + x^2 + x$

(WITH 21 FITNESS CASES)

<table>
<thead>
<tr>
<th>Independent variable (Input)</th>
<th>$X$</th>
<th>Dependent variable (Output)</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td></td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>-0.9</td>
<td></td>
<td>-0.1629</td>
<td></td>
</tr>
<tr>
<td>-0.8</td>
<td></td>
<td>-0.2624</td>
<td></td>
</tr>
<tr>
<td>-0.7</td>
<td></td>
<td>-0.3129</td>
<td></td>
</tr>
<tr>
<td>-0.6</td>
<td></td>
<td>-0.3125</td>
<td></td>
</tr>
<tr>
<td>-0.5</td>
<td></td>
<td>-0.2784</td>
<td></td>
</tr>
<tr>
<td>-0.4</td>
<td></td>
<td>-0.2289</td>
<td></td>
</tr>
<tr>
<td>-0.3</td>
<td></td>
<td>-0.1664</td>
<td></td>
</tr>
<tr>
<td>-0.2</td>
<td></td>
<td>-0.0909</td>
<td></td>
</tr>
<tr>
<td>-0.1</td>
<td></td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0.1111</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td>0.2496</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>0.4251</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td></td>
<td>0.6496</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td>0.9375</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
<td>1.3056</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td>1.7731</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
<td>2.3616</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
<td>3.0951</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
<td>4.0000</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td>4.0000</td>
<td></td>
</tr>
</tbody>
</table>
TABLEAU—SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

| Objective: | Find a function of one independent variable, in symbolic form, that fits a given sample of 21 $(x_i, y_i)$ data points. |
| Terminal set: | $x$ (the independent variable). |
| Function set: | $+, -, *, \div, \sin, \cos, \exp, \text{RLog}$ |
| Fitness cases: | The given sample of 21 data points $(x_i, y_i)$ where the $x_i$ are in 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. |

SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

WORST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 1038

$$(\exp (- ( \% X (- X (\sin X))) (\text{RLog} (\text{RLog} (* X X))))))$$

Equivalent to $e^x/(x\sin x) - \log \log x^x$

MEDIAN INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 23.67 (AVERAGE ERROR OF 1.3)

$$(\cos (\cos (+ (- (* X X) (\% X X)) X)))$$

Equivalent to $\cos [\cos (x^2 + x - 1)]$

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 4.47 (AVERAGE ERROR OF 0.2)

$$(* X (+ (+ (- (\% X X) (\% X X)) (\sin (- X X))) (\text{RLog} (\exp (\exp X))))))$$

Equivalent to $xe^x$
SYMBOLIC REGRESSION
OF QUARTIC POLYNOMIAL \( X^4 + X^3 + X^2 + X \)

CREATION OF GENERATION 1 FROM GENERATION 0

- In the so-called "generational" model for genetic algorithms, a new population is created that is equal in size to the old population
  - 1% mutation (i.e., 5 individuals out of 500)
  - 9% reproduction (i.e., 45 individuals)
  - 90% crossover (i.e., 225 pairs of parents — yielding 450 offspring)

- All participants in mutation, reproduction, and crossover are chosen from the current population PROBABILISTICALLY, BASED ON FITNESS
  - Anything can happen
  - Nothing is guaranteed
  - The search is heavily (but not completely) biased toward high-fitness individuals
  - The best is not guaranteed to be chosen
  - The worst is not necessarily excluded
  - Some (but not much) attention is given even to low-fitness individuals

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 2 WITH RAW FITNESS OF 2.57 (AVERAGE ERROR OF 0.1)

\[
(+ (* (* (+ X (* X (* X (% (% X X) (+ X X))))) (+ X (* X X))))) X) X
\]

Equivalent to...
\[
x^4 + 1.5x^3 + 0.5x^2 + x
\]

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

\[
(+ X (* (+ X (* (* (+ X (- (COS (- X X)) (- X X))) X) X)) X) X)
\]

Equivalent to
\[
x^4 + x^3 + x^2 + x
\]

OBSERVATIONS

- GP works on this problem
- GP determines the size and shape of the solution
  - number of operations needed to solve the problem
  - size and shape of the program tree
  - content of the program tree (i.e., sequence of operations)
- GP operates the same whether the solution is linear, polynomial, a rational fraction of polynomials, exponential, trigonometric, etc.
- It's not how a human programmer would have done it
  - \( \cos (X - X) = 1 \)
  - Not parsimonious

- The extraneous functions – \( \sin, \exp, \log, \) and \( \rcos \) are absent in the best individual of later generations because they are detrimental
  - \( \cos (X - X) = 1 \) is the exception that proves the rule
- The answer is algebraically correct (hence no further cross validation is needed)
CLASSIFICATION PROBLEM
INTER-TWINED 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, ℜ, where ℜ 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.

WALL-FOLLOWING PROBLEM

12 SONAR SENSORS

FITNESS MEASURE
WALL-FOLLOWING PROBLEM
BEST PROGRAM OF GENERATION 57

• Scores 56 hits (out of 56)
• 145 point program tree

AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

• Subroutines provide one way to REUSE code — possibly with different instantiations of the dummy variables (formal parameters)
• Loops (and iterations) provide a 2nd way to REUSE code
• Recursion provide a 3rd way to REUSE code
• Memory provides a 4th way — to REUSE the results of executing code

24 PROBLEMS SHOWN IN 1992 VIDEOTAPE
GENETIC PROGRAMMING: THE MOVIE
(KOZA AND RICE 1992)

• Symbolic Regression
• Intertwined Spirals
• Artificial Ant
• Truck Backer Upper
• Broom Balancing
• Wall Following
• Box Moving
• Discrete Pursuer-Evader Game
• Differential Pursuer-Evader Game
• Co-Evolution of Game-Playing Strategies
• Inverse Kinematics
• Emergent Collecting
• Central Place Foraging
• Block Stacking
• Randomizer
• 1-D Cellular Automata
• 2-D Cellular Automata
• Task Prioritization
• Programmatic Image Compression
• Finding \( \sqrt{2} \)
• Econometric Exchange Equation
• Optimization (Lizard)
• Boolean 11-Multiplexer
• 11-Parity—Automatically Defined Functions

AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

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

<table>
<thead>
<tr>
<th>Fitness case</th>
<th>( L_0 )</th>
<th>( W_0 )</th>
<th>( H_0 )</th>
<th>( L_1 )</th>
<th>( W_1 )</th>
<th>( H_1 )</th>
<th>Dependent variable ( D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>600</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>312</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>–18</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>–171</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>363</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>–36</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>–24</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>45</td>
</tr>
</tbody>
</table>
AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

SOLUTION WITHOUT ADFS

\[- (* (* W0 L0) H0) \]
\[ (* (* W1 L1) H1) \]

\[ D = W0*H0 - W1*L1*H1 \]

AN OVERALL COMPUTER PROGRAM CONSISTING OF ONE FUNCTION-DEFINING BRANCH (ADF, SUBROUTINE) AND ONE RESULT-PRODUCING BRANCH (MAIN PROGRAM)

\[(progn\]
\[(defun volume (arg0 arg1 arg2)\]
\[(values\]
\[(* arg0 (* arg1 arg2)))\]
\[(values \(- (volume L0 W0 H0)\]
\[(volume L1 W1 H1))))\]

AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

IF WE ADD TWO NEW VARIABLES FOR VOLUME \(V_0\) AND \(V_1\), THE 6-DIMENSIONAL NON-LINEAR REGRESSION PROBLEM BECOMES AN 8-DIMENSIONAL PROBLEM

<table>
<thead>
<tr>
<th>Fitness case</th>
<th>(L_0)</th>
<th>(W_0)</th>
<th>(H_0)</th>
<th>(L_1)</th>
<th>(W_1)</th>
<th>(H_1)</th>
<th>(V_0)</th>
<th>(V_1)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>84</td>
<td>30</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>630</td>
<td>30</td>
<td>600</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>360</td>
<td>48</td>
<td>312</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>135</td>
<td>24</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>24</td>
<td>42</td>
<td>-18</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>180</td>
<td>-171</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>405</td>
<td>42</td>
<td>363</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>18</td>
<td>54</td>
<td>-36</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>96</td>
<td>120</td>
<td>-24</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>80</td>
<td>35</td>
<td>45</td>
</tr>
</tbody>
</table>

- However, the problem can now be approached as a 2-dimensional LINEAR regression problem.

AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

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

DIVIDE AND CONQUER

- Decompose a problem into subproblems
- Solve the subproblems
- Assemble the solutions of the subproblems into a solution for the overall problem
AUTOMATICALLY DEFINED FUNCTIONS (ADFs, SUBROUTINES)

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

- Identify regularities
- Change the representation
- Solve the overall problem

In generation 0, we create a population of programs, each consisting of a main result-producing branch (RPB) and one or more function-defining branches (automatically defined functions, ADFs, subroutines)
- Different ingredients for RPB and ADFs
- The terminal set of an ADF typically contains dummy arguments (formal parameters), such as ARG0, ARG1, ...
- The function set of the RPB contains ADF0, ...
- ADFs are private and associated with a particular individual program in the population
- The entire program is executed and evaluated for fitness
- Genetic operation of reproduction is the same as before
- Mutation operation starts (as before) by picking a mutation point from either RPB or an ADF and deleting the subtree rooted at that point. As before, a subtree is then grown at the point. The new subtree is composed of the allowable ingredients for that point — so that the result is a syntactically valid executable program.
- Crossover operation starts (as before) by picking a crossover point from either RPB or an ADF of one parent. The choice of crossover point in the second parent is then restricted (e.g., to the RPB or to the ADF) — so that when the subtrees are swapped, the result is a syntactically valid executable program.

8 MAIN POINTS FROM BOOK
GENETIC PROGRAMMING II:
AUTOMATIC DISCOVERY OF REUSABLE PROGRAMS (KOZA 1994)

- 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.

REUSE
MEMORY AND STORAGE

- (A) Settable (named) variables (Genetic Programming, Koza 1992) using setting (writing) functions (SETM0 X) and (SETM1 Y) and reading by means of terminals M0 and M1.
- (B) Indexed memory similar to linear (vector) computer memory (Teller 1994) using (READ K) and (WRITE X K)
- (C) Matrix memory (Andre 1994)
- (D) Relational memory (Brave 1995, 1996)

LANGDON’S DATA STRUCTURES
- Stacks
- Queues
- Lists
- Rings
REUSE

AUTOMATICALLY DEFINED
ITERATIONS (ADIs)

• Overall program consisting of an automatically defined function ADF0, an iteration-performing branch IPB0, and a result-producing branch RPB0.
• Iteration is over a known, fixed set
  • protein or DNA sequence (of varying length
  • time-series data
  • two-dimensional array of pixels

REUSE

TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

• Goal is to classify a given protein segment as being a transmembrane domain or non-transmembrane area of the protein
• Generation 20 — Run 3 — Subset-creating version
• in-sample correlation of 0.976
• out-of-sample correlation of 0.968
• out-of-sample error rate 1.6%

(progn
  (defun ADF0 ()
    (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 (I?) (H?) (T?) (L?) (W?))) (ORN (ORN (ORN (T?) (L?) (L?)) (ORN (T?) (W?))))
    (ORN (ORN (T?) (L?) (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?) (Q?))) (ORN (ORN (T?) (W?))))
    (ORN (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))))

REUSE

EXAMPLE OF A PROGRAM WITH A FOUR-BRANCH AUTOMATICALLY DEFINED LOOP (ADL0) AND A RESULT-PRODUCING BRANCH

(REUSE

AUTOMATICALLY DEFINED
RECURSION (ADR0) AND A RESULT-PRODUCING BRANCH

• a recursion condition branch, RCB
• a recursion body branch, RBB
• a recursion update branch, RUB
• a recursion ground branch, RGB

(REUSE
GP TECHNIQUES

- control structures involving multiple result-producing branches (Luke and Spector 1996a; Bennett 1996a; Svingen 1997)
- adaptive self-modifying ontogenetic genetic programming (Spector and Stoffel 1996a; 1996b)
- cultural storage and transmission (Spector and Luke 1996a; 1996b)
- hierarchical problem solving (Rosca and Ballard 1994a; 1994b; Rosca 1995; Rosca 1997)
- modules (Angeline and Pollack 1993; 1994; Angeline 1993; 1994; Kinnear 1994b)
- cellular encoding (developmental genetic programming) for evolving neural networks (Gruau 1992a; 1992b; 1993; 1994a; 1994b; Gruau and Whitley 1993; Esparcia-Alcazar and Sharman 1997)
- developmental methods for evolving finite automata using genetic programming (Brave 1996a)
- developmental methods for shape optimization (Kennelly 1997)
- evolving graphs and networks (Luke and Spector 1996b)
- using a grammar to represent bias and background knowledge (Whigham 1995a; 1995b; 1996)
- developmental methods for fuzzy logic systems (Tunstel and Jamshidi 1996)

GP TECHNIQUES—CONTINUED

- diploidy and dominance (Greene 1997a; 1997b)
- Turing completeness of genetic programming (Teller 1994c; Nordin and Banzhaf 1995)
- evolution of chemical topological structures (Nachbar 1997; 1998)
- interactive fitness measures (Poli and Cagnoni 1997) and in particular in graphics and art (Sims 1991a; 1991b; 1992a; 1992b; 1993)
- variations in crossover operations (Poli and Langdon 1997)
- complexity-based fitness measures using minimum description length (Iba, Kurita, de Garis, and Sato 1993; Iba et al. 1994)
- co-evolution (Reynolds 1994c)
- steady state genetic programming (Reynolds 1993; 1994a; 1994b)
- use of noise in fitness cases (Reynolds 1994d)
- balancing parsimony and accuracy (Zhang and Muhlenbein 1993; 1994; 1995; Blickle 1997)
- automatically defined features using genetic algorithms in conjunction with genetic programming (Andre 1994a)
- grammatical evolution (Conor Ryan and Michael O'Neill)

ARCHITECTURE-ALTERING OPERATIONS

PROTEIN ALIGNMENT OF "A" AND "B" PROTEINS

<table>
<thead>
<tr>
<th>First.protein</th>
<th>Second.protein</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRIKFLVVLA V</td>
<td>MIEKFLVLVA V</td>
</tr>
<tr>
<td>VICLFAHYAS A</td>
<td>VICLFAHYAS A</td>
</tr>
<tr>
<td>ASGMGGDKKP K</td>
<td>KDAPKPKDAP K</td>
</tr>
<tr>
<td>KDAPKPKDAP K</td>
<td>KDAPKPKDAP K</td>
</tr>
<tr>
<td>KPKEVKPVK</td>
<td>KPKEVKPVK</td>
</tr>
<tr>
<td>First.protein</td>
<td>Second.protein</td>
</tr>
<tr>
<td>ESSEYEIEVI K</td>
<td>DSSEYEIEVI K</td>
</tr>
<tr>
<td>KHQKEKTEKK E</td>
<td>KHQKEKTEKK E</td>
</tr>
<tr>
<td>EKEKKTHVET K</td>
<td>KKKIKNKEKK F</td>
</tr>
<tr>
<td>KKEVKKKEKK Q</td>
<td>FVPCSEILK</td>
</tr>
<tr>
<td>QIPCSEKLK</td>
<td>FVPCSEILK</td>
</tr>
<tr>
<td>First.protein</td>
<td>Second.protein</td>
</tr>
<tr>
<td>EKLDCETKGV P</td>
<td>EKLECEKNAT P</td>
</tr>
<tr>
<td>PA GYKAIFKF T</td>
<td>KESESFCEWE K</td>
</tr>
<tr>
<td>TENEE-CDWT CD</td>
<td>CDYEALPPPP G</td>
</tr>
<tr>
<td>YEAALPPPP G</td>
<td></td>
</tr>
<tr>
<td>First.protein</td>
<td>Second.protein</td>
</tr>
<tr>
<td>KTVKVVKPPK E</td>
<td>KVVKVIKPPK E</td>
</tr>
<tr>
<td>EKPPKKLRKE C</td>
<td>EKPPKKPRKE C</td>
</tr>
<tr>
<td>CSGEKVIKFQ N</td>
<td>CSGEKVIKFQ N</td>
</tr>
<tr>
<td>NCLVKIRGLI A</td>
<td>NCLVKIRGLI A</td>
</tr>
<tr>
<td>AF DGTKTKNF</td>
<td>AF DGTKTKNF</td>
</tr>
<tr>
<td>First.protein</td>
<td>Second.protein</td>
</tr>
<tr>
<td>--KDAKK 24</td>
<td>KDKDAKK 25</td>
</tr>
<tr>
<td>Second.protein</td>
<td>KPKDAKK 25</td>
</tr>
<tr>
<td>KPKDAKK 25</td>
<td>KPKDAKK 25</td>
</tr>
<tr>
<td>KKFAKLVQGK Q</td>
<td>KKFAKLVQGK Q</td>
</tr>
<tr>
<td>QKKGAKKAKG G</td>
<td>QKKGAKKAKG G</td>
</tr>
<tr>
<td>GKKAAPKPGP K</td>
<td>GKKAEPKPGP K</td>
</tr>
<tr>
<td>KPGPK----Q AD</td>
<td>KPAPKPGPKP APKVPVPK</td>
</tr>
<tr>
<td>ADKP---219</td>
<td>ADKP---219</td>
</tr>
<tr>
<td>First.protein</td>
<td>Second.protein</td>
</tr>
<tr>
<td>2KRP</td>
<td>KEF4</td>
</tr>
<tr>
<td>Second.protein</td>
<td>KEF4</td>
</tr>
<tr>
<td>2KRP</td>
<td>KEF4</td>
</tr>
</tbody>
</table>

46
ARCHITECTURE-ALTERING OPERATIONS

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

ARCHITECTURE-ALTERING OPERATIONS

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

ARCHITECTURE-ALTERING OPERATIONS

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

ARCHITECTURE-ALTERING OPERATIONS

SPECIALIZATION – REFINEMENT – CASE SPLITTING

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

GENERALIZATION

- Branch deletion
- Argument deletion
16 ATTRIBUTES OF A SYSTEM FOR AUTOMATICALLY CREATING COMPUTER PROGRAMS

1 — Starts with "What needs to be done"
2 — Tells us "How to do it"
3 — Produces a computer program
4 — Automatic determination of program size
5 — Code reuse
6 — Parameterized reuse
7 — Internal storage
8 — Iterations, loops, and recursions
9 — Self-organization of hierarchies
10 — Automatic determination of program architecture
11 — Wide range of programming constructs
12 — Well-defined
13 — Problem-independent
14 — Wide applicability
15 — Scalable
16 — Competitive with human-produced results

ARCHITECTURE-ALTERING OPERATIONS

GENETIC PROGRAMMING PROBLEM SOLVER (GPPS) — VERSION 2.0

IMPLEMENTATION OF GP IN ASSEMBLY CODE – COMPILED GENETIC PROGRAMMING SYSTEM (NORDIN 1994)

- Opportunity to speed up GP that is done by slowly INTERPRETING GP program trees. Instead of interpreting the GP program tree, EXECUTE this sequence of assembly code.
  - Can identify small set of primitive functions that is useful for large group of problems, such as +, -, *, % and also use some conditional operations (IFLTE), some logical functions (AND, OR, XOR, XNOR) and perhaps others (e.g., SRL, SLL, SETHI from Sun 4).
  - Then, generate random sequence of assembly code instructions at generation 0 from this small set of machine code instructions (referring to certain registers).
  - If ADFs are involved, generate fixed header and footer of function and appropriate function call.
  - Perform crossover possibly so as to preserve the integrity of subtrees.
  - If ADFs are involved, perform crossover so as to preserve the integrity of the header and footer of function and the function call.

DESIGN OF QUANTUM COMPUTER CIRCUITS USING GP

CELLULAR ENCODING (DEVELOPMENTAL GENETIC PROGRAMMING)

  - Applied by Gruau to six-legged walking creature

AUTOMATIC PARALLELIZATION OF SERIAL PROGRAMS USING GP

  - Start with working serial computer program (embryo)
  - GP program tree contains validity-preserving functions that modify the current program. That is, the functions in the program tree side-effect the current program.
  - Execution of the complete GP program tree progressively modifies the current program
  - Fitness is based on execution time on the parallel computer system

DEVELOPMENTAL GP

THE INITIAL CIRCUIT

- Initial circuit consists of embryo and test fixture
- Embryo has modifiable wires (e.g., Z0 AND Z1)
- Test fixture has input and output ports and usually has source resistor and load resistor. There are no modifiable wires (or modifiable components) in the test fixture.
- Circuit-constructing program trees consist of
  - Component-creating functions
  - Topology-modifying functions
  - Development-controlling functions
- Circuit-constructing program tree has one result-producing branch for each modifiable wire in embryo of the initial circuit

DEVELOPMENTAL GP

DEVELOPMENT OF A CIRCUIT FROM A CIRCUIT-CONSTRUCTING PROGRAM TREE AND THE INITIAL CIRCUIT

(LIST (C (- 0.963 (- (- -0.875 -0.113) 0.880)) (series (flip end) (series (flip end) (L -0.277 end) end) (L (- -0.640 0.749) (L -0.123 end))) (flip (nop (L -0.657 end)))))
DEVELOPMENTAL GP

RESULT OF THE $C(2)$ FUNCTION

\[
\text{LIST} \ (C (-0.963 (-(-0.875 -0.113) 0.880))) \ (\text{series} \ (\text{flip end}) \ (\text{series} \ (\text{flip end}) \ (L -0.277 \text{ end}) \text{ end}) \ (L (- -0.640 0.749) (L -0.123 \text{ end})'))) \ (\text{flip (nop (L -0.657 \text{ end}))))
\]

NOTE: Interpretation of arithmetic value

DEVELOPMENTAL GP

RESULT OF SERIES (5) FUNCTION

\[
\text{LIST} \ (C (-0.963 (-(-0.875 -0.113) 0.880))) \ (\text{series} \ (\text{flip end}) \ (\text{series} \ (\text{flip end}) \ (L -0.277 \text{ end}) \text{ end}) \ (L (- -0.640 0.749) (L -0.123 \text{ end})'))) \ (\text{flip (nop (L -0.657 \text{ end}))))
\]

EVALUATION OF FITNESS OF A CIRCUIT

Program Tree

\[
\begin{array}{c}
\text{IN} \\
\text{Program Tree} \\
\downarrow \\
\text{Fully Designed Circuit (NetGraph)} \\
\downarrow \\
\text{Circuit Netlist (ascii)} \\
\downarrow \\
\text{Circuit Simulator (SPICE)} \\
\downarrow \\
\text{Circuit Behavior (Output)} \\
\downarrow \\
\text{Fitness}
\end{array}
\]

Fully Designed Circuit (NetGraph)

Circuit Simulator (SPICE)

Circuit Behavior (Output)

Fitness

BEHAVIOR OF A LOWPASS FILTER VIEWED IN THE FREQUENCY DOMAIN

- Examine circuit’s behavior for each of 101 frequency values chosen over five decades of frequency (from 1 Hz to 100,000 Hz) with each decade divided into 20 parts (using a logarithmic scale). The fitness measure
  - does not penalize ideal values
  - slightly penalizes acceptable deviations
  - heavily penalizes unacceptable deviations
- Fitness is sum $F(t) = \sum_{i=0}^{100} [W(f_i) d(f_i)]$
  - $f(i)$ is the frequency of fitness case $i$
  - $d(x)$ is the difference between the target and observed values at frequency of fitness case $i$
  - $W(y,x)$ is the weighting at frequency $x$
**TABLEAU — LOWPASS FILTER (WITHOUT ADFs OR ARCHITECTURE-ALTERING OPERATIONS)**

| Objective: | Design a lowpass filter composed of inductors and capacitors with a passband below 1,000 Hz, a stopband above 2,000 Hz, a maximum allowable passband deviation of 30 millivolts, and a maximum allowable stopband deviation of 1 millivolt. |
| Test fixture and embryo: | One-input, one-output initial circuit with a source resistor, load resistor, and two modifiable wires. |
| Program architecture: | Two result-producing branches, RPB0 and RPB1 (i.e., one RPB per modifiable wire in the embryo). |
| Initial function set for the result-producing branches: | For construction-continuing subtrees: \( F_{csc-rpb-initial} = \{ C, L, SERIES, PARALLEL0, FLIP, NOP, TWO\_GROUND, TWO\_VIA0, TWO\_VIA1, TWO\_VIA2, TWO\_VIA3, TWO\_VIA4, TWO\_VIA5, TWO\_VIA6, TWO\_VIA7 \} \). For arithmetic-performing subtrees: \( F_{aps} = \{ +, - \} \). |
| Initial terminal set for the result-producing branches: | For construction-continuing subtrees: \( T_{csc-rpb-initial} = \{ END \} \). For arithmetic-performing subtrees: \( T_{aps} = \{ e\text{-smaller-reals} \} \). |

**Fitness cases:** 101 frequency values in an interval of five decades of frequency values between 1 Hz and 100,000 Hz.

**Raw fitness:** Fitness is the sum, over the 101 sampled frequencies (fitness cases), of the absolute weighted deviation between the actual value of the output voltage that is produced by the circuit at the probe point and the target value for voltage. The weighting penalizes unacceptable output voltages much more heavily than deviating, but acceptable, voltages.

**Standardized fitness:** Same as raw fitness.

**Hits:** The number of hits is defined as the number of fitness cases (out of 101) for which the voltage is acceptable or ideal or that lie in the "don't care" band.

**Wrapper:** None.

**Parameters:** \( M = 1,000 \) to 320,000. \( G = 1,001 \). \( Q = 1,000 \). \( D = 64 \). \( B = 2\% \). \( N_{rpb} = 2 \). \( S_{rpb} = 200 \).

**Result designation:** Best-so-far pace-setting individual.

**Success predicate:** A program scores the maximum number (101) of hits.

---

**EVOLVED CAMPBELL FILTER (7-RUNG LADDER)**

- This genetically evolved circuit infringes on U. S. patent 1,227,113 issued to George Campbell of American Telephone and Telegraph in 1917 (claim 2):

  An electric wave filter consisting of a connecting line of negligible attenuation composed of a plurality of sections, each section including a capacity element and an inductance element, one of said elements of each section being in series with the line and the other in shunt across the line, said capacity and inductance elements having precomputed values dependent upon the upper limiting frequency and the lower limiting frequency of a range of frequencies it is desired to transmit without attenuation, the values of said capacity and inductance elements being so proportioned that the structure transmits with practically negligible attenuation sinusoidal currents of all frequencies lying between said two limiting frequencies, while attenuating and approximately extinguishing currents of neighboring frequencies lying outside of said limiting frequencies."

---

**EVOLVED ZOBEL FILTER**

- Infringes on U. S. patent 1,538,964 issued in 1925 to Otto Zobel of American Telephone and Telegraph Company for an “M-derived half section” used in conjunction with one or more “constant K” sections.
- One M-derived half section \( (C2 \text{ and } L11) \)
- Cascade of three symmetric T-sections
GENETICALLY EVOLVED 10 DB AMPLIFIER FROM GENERATION 45

SHOWING THE VOLTAGE GAIN STAGE AND DARLINGTON Emitter FOLLOWER SECTION

POST-2000 PATENTED INVENTIONS

HIGH CURRENT LOAD CIRCUIT
BEST-OF-RUN FROM GENERATION 114

POST-2000 PATENTED INVENTIONS

REGISTER-CONTROLLED CAPACITOR CIRCUIT

SMALLEST COMPLIANT FROM GENERATION 98

POST-2000 PATENTED INVENTIONS

LOW-VOLTAGE CUBIC SIGNAL GENERATION CIRCUIT
BEST-OF-RUN FROM GENERATION 182
POST-2000 PATENTED INVENTIONS

LOW-VOLTAGE BALUN CIRCUIT
BEST EVOLVED FROM GENERATION 84

VOLTAGE-CURRENT-CONVERSION CIRCUIT
BEST-OF-RUN FROM GENERATION 109

TUNABLE INTEGRATED ACTIVE FILTER — GENERATION 50

21 PREVIOUSLY PATENTED INVENTIONS REINVENTED BY GP

<table>
<thead>
<tr>
<th>Invention</th>
<th>Date</th>
<th>Inventor</th>
<th>Place</th>
<th>Patent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Darlington emitter-follower section</td>
<td>1953</td>
<td>Sidney Darlington</td>
<td>Bell Telephone Laboratories</td>
<td>2,663,806</td>
</tr>
<tr>
<td>2  Ladder filter</td>
<td>1917</td>
<td>George Campbell</td>
<td>American Telephone and Telegraph</td>
<td>1,227,113</td>
</tr>
<tr>
<td>3  Crossover filter</td>
<td>1925</td>
<td>Otto Julius Zobel</td>
<td>American Telephone and Telegraph</td>
<td>1,538,964</td>
</tr>
<tr>
<td>4  &quot;M-derived half section&quot; filter</td>
<td>1925</td>
<td>Otto Julius Zobel</td>
<td>American Telephone and Telegraph</td>
<td>1,538,964</td>
</tr>
<tr>
<td>5  Cauer (elliptic) topology for filters</td>
<td>1934–1936</td>
<td>Wilhelm Cauer</td>
<td>University of Gottingen</td>
<td>1,958,742, 1,989,545</td>
</tr>
<tr>
<td>6  Sorting network</td>
<td>1962</td>
<td>Daniel G. O'Connor and Raymond J. Nelson</td>
<td>General Precision, Inc.</td>
<td>3,029,413</td>
</tr>
<tr>
<td>7  Computation circuits</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
</tr>
<tr>
<td>8  Electronic thermometer</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
</tr>
<tr>
<td>9  Voltage reference circuit</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
</tr>
<tr>
<td>10 60 dB and 96 dB amplifiers</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
<td>See text</td>
</tr>
<tr>
<td>11 Second-derivative controller</td>
<td>1942</td>
<td>Harry Jones</td>
<td>Brown Instrument Company</td>
<td>2,282,726</td>
</tr>
<tr>
<td>12 Philbrick circuit</td>
<td>1956</td>
<td>George Philbrick</td>
<td>George A. Philbrick Researches</td>
<td>2,730,679</td>
</tr>
<tr>
<td>13 SASSD circuit</td>
<td>1971</td>
<td>David H. Chang and Bill H.</td>
<td>Texas Instruments Incorporated</td>
<td>3,568,760</td>
</tr>
</tbody>
</table>
2 PATENTED INVENTIONS CREATED BY GENETIC PROGRAMMING

SOLUTION NO. 1

SOLUTION NO. 5

LAYOUT — LOWPASS FILTER 100%-COMPLIANT CIRCUITS
GENERATION 25 WITH 5 CAPACITORS AND 11 INDUCTORS — AREA OF 1775.2

GENERATION 30 WITH 10 INDUCTORS AND 5 CAPACITORS — AREA OF 950.3

BEST-OF-RUN CIRCUIT OF GENERATION 138 WITH 4 INDUCTORS AND 4 CAPACITORS — AREA OF 359.4

LAYOUT — 60 DB AMPLIFIER (USING TRANSISTORS)

COMPARISON

<table>
<thead>
<tr>
<th>Gen</th>
<th>Component</th>
<th>Area</th>
<th>Four penalties</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>27</td>
<td>8,234</td>
<td>33.034348</td>
<td>33.042583</td>
</tr>
<tr>
<td>101</td>
<td>19</td>
<td>4,751</td>
<td>0.061965</td>
<td>0.004751</td>
</tr>
</tbody>
</table>

BEST-OF-RUN CIRCUIT FROM GENERATION 101

DESIGN OF OPTICAL LENS SYSTEMS (AL-SAKRAN, KOZA, AND JONES, 2005; KOZA, AL-SAKRAN, AND JONES, 2005)

Tackaberry-Muller lens system

Lens file for Tackaberry-Muller system

<table>
<thead>
<tr>
<th>Surface</th>
<th>Distance</th>
<th>Diameter</th>
<th>Material</th>
<th>Aperture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>1P</td>
<td>0.0000</td>
<td>air</td>
<td>0.00</td>
</tr>
<tr>
<td>Entry pupil</td>
<td>0.01000</td>
<td>0.3056</td>
<td>BK7</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0.07200</td>
<td>1.6027</td>
<td>air</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>0.08500</td>
<td>1.8034</td>
<td>air</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.08400</td>
<td>1.8034</td>
<td>BK7</td>
<td>0.02</td>
</tr>
<tr>
<td>4</td>
<td>0.10000</td>
<td>1.8949</td>
<td>SF61</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>0.11800</td>
<td>1.9120</td>
<td>SF61</td>
<td>0.02</td>
</tr>
<tr>
<td>6</td>
<td>0.13400</td>
<td>1.9120</td>
<td>air</td>
<td>0.02</td>
</tr>
<tr>
<td>Image</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DEVELOPMENTAL PROCESS

TURTLE STARTS AT POINT G ALONG
MAIN AXIS B

TURTLE INSERTS SURFACE 1

TURTLE INSERTS SURFACE 2

DEVELOPMENTAL PROCESS—
CONTINUED

LENS SPLITTING OPERATION

LENS SYSTEM BEFORE LENS-
SPLITTING OPERATION

LENS SYSTEM AFTER LENS-SPLITTING
OPERATION
GLASS MUTATION

GLASS MAP FOR THE 199 TYPES OF GLASS IN THE SCHOTT CATALOG

PID CONTROLLER

Block diagram of a plant and a PID controller composed of proportional (P), integrative (I), and derivative (D) blocks

PROGRAM TREE REPRESENTATION FOR PID CONTROLLER

• ADF can be used for reuse.
• Automatically defined function ADF0 takes the difference between the reference signal and the plant output and makes this difference available to three points in the result-producing branch

FUNCTION SET AND TERMINAL SET FOR TWO-LAG PLANT PROBLEM

• The function set, F (for every part of the result-producing branch and any automatically defined functions except the arithmetic-performing subtrees) is

\[ F = \{ \text{GAIN, INVERTER, LEAD, LAG, LAG2, DIFFERENTIAL\_INPUT\_INTEGRATOR, DIFFERENTIATOR, ADD\_SIGNAL, SUB\_SIGNAL, ADD\_3\_SIGNAL, ADF0, ADF1, ADF2, ADF3, ADF4}\} \]

• The terminal set, T, (for every part of the result-producing branch and any automatically defined functions except the arithmetic-performing subtrees) is

\[ T = \{ \text{REFERENCE\_SIGNAL, CONTROLLER\_OUTPUT, PLANT\_OUTPUT, CONSTANT\_0}\} \]
ARITHMETIC-PERFORMING SUBTREES FOR THE TWO-LAG PLANT PROBLEM

- Signal processing blocks such as GAIN, LEAD, LAG, and LAG2 possess numerical parameter(s)
- Parameter values can be established by an arithmetic-performing subtree
- A constrained syntactic structure enforces a different function and terminal set for the arithmetic-performing subtrees (as opposed to all other parts of the program tree).
- Terminal set, $T_{\text{aps}}$, for the arithmetic-performing subtrees
  
  \[
  T_{\text{aps}} = \{ \mathbb{R} \}
  \]

  where $\mathbb{R}$ denotes constant numerical terminals in the range from -1.0 to +1.0

- Function set, $F_{\text{aps}}$, for the arithmetic-performing subtrees
  
  \[
  F_{\text{aps}} = \{ \text{ADD\_NUMERIC, SUB\_NUMERIC} \}
  \]

FITNESS MEASURE FOR TWO-LAG PLANT

- 10-element fitness measure
- The first eight elements of the fitness measure represent the eight choices of a particular one of two different values of the plant's internal gain, $K$ (1.0 and 2.0), in conjunction with a particular one of two different values of the plant's time constant $\tau$ (0.5 and 1.0), in conjunction with a particular one of two different values for the height of the reference signal. The two reference signals are step functions that rise from 0 to 1 volts (or 1 microvolts) at $t = 100$ milliseconds.
- For each of these eight fitness cases, a transient analysis is performed in the time domain using the SPICE simulator. The contribution to fitness for each of these eight elements is

  \[
  \int_0^T e(t) dt
  \]

  where $e(t)$ is difference between plant output and reference signal.
- Multiplication by $B$ ($10^6$ or 1) makes both reference signals equally influential.
- Additional weighting function, $A$, heavily penalizes non-compliant amounts of overshoot. $A$ weights all variations up to 2% above the reference signal by 1.0, but others by 10.0.
- The 9th element of the fitness measure exposes the controller to an extreme spiked reference signal.
- The 10th element constrains the frequency of the control variable so as to avoid extreme high frequencies.

BEST-OF-RUN GENETICALLY EVOLVED CONTROLLER FROM GENERATION 32 FOR THE TWO-LAG PLANT

![Best-of-run controller diagram](attachment:best_of_run_controller.png)

COMPARISON OF THE TIME-DOMAIN RESPONSE TO 1-VOLT STEP INPUT FOR THE EVOLVED CONTROLLER (TRIANGLES) AND THE BISHOP AND DORF CONTROLLER (SQUARES) FOR THE TWO-LAG PLANT WITH $K=1$ AND $\tau=1$

![Time-domain response comparison](attachment:time_domain_response.png)

OVERALL MODEL

![Overall model](attachment:overall_model.png)
COMPARISON OF THE TIME-DOMAIN RESPONSE TO A 1-VOLT DISTURBANCE SIGNAL OF THE EVOLVED CONTROLLER (TRIANGLES) AND THE BISHOP AND DORF CONTROLLER (CIRCLES) FOR THE TWO-LAG PLANT WITH $K=1$ AND $\tau=1$

CROSS-DOMAIN FEATURES OF RUNS OF GENETIC PROGRAMMING USED TO EVOLVE DESIGNS FOR ANALOG CIRCUITS, OPTICAL LENS SYSTEMS, CONTROLLERS, ANTENNAS, MECHANICAL SYSTEMS, AND QUANTUM COMPUTING CIRCUITS

- optical lens systems (Al-Sakran, Koza, and Jones, 2005; Koza, Al-Sakran, and Jones, 2005),
- analog electrical circuits (Koza, Bennett, Andre, and Keane 1996; Koza, Bennett, Andre, and Keane 1999),
- antennas (Lohn, Hornby, and Linden 2004; Comisky, Yu, and Koza 2000),
- controllers (Koza, Keane, Streeter, Mydlowec, Yu, and Lanza 2003; Keane, Koza, Streeter 2005),
- mechanical systems (Lipson 2004), and
- quantum computing circuits (Spector 2004)

REVERSE ENGINEERING OF METABOLIC PATHWAYS (4-REACTION NETWORK IN PHOSPHOLIPID CYCLE)

BEST-OF-GENERATION 66

DESIRED

CROSS-DOMAIN FEATURES

- Native representations are sufficient when working with genetic programming
- Genetic programming breeds simulatability
- Genetic programming starts small
- Genetic programming frequently exploits a simulator’s built-in assumption of reasonableness
- Genetic programming engineers around existing patents and creates novel designs more frequently than it creates infringing solutions
NATIVE REPRESENTATIONS ARE SUFFICIENT WHEN WORKING WITH GENETIC PROGRAMMING

Tackaberry-Muller lens system

Lens file for Tackaberry-Muller system

<table>
<thead>
<tr>
<th>Surface</th>
<th>Distance</th>
<th>Material</th>
<th>Aperture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>10</td>
<td>flat</td>
<td>0.1</td>
</tr>
<tr>
<td>Entry pupil</td>
<td>0.18</td>
<td>air</td>
<td>0.18</td>
</tr>
<tr>
<td>1</td>
<td>0.2100</td>
<td>BK7</td>
<td>0.42</td>
</tr>
<tr>
<td>2</td>
<td>0.0720</td>
<td>air</td>
<td>0.42</td>
</tr>
<tr>
<td>3</td>
<td>0.2500</td>
<td>BK7</td>
<td>0.42</td>
</tr>
<tr>
<td>4</td>
<td>0.2130</td>
<td>air</td>
<td>0.42</td>
</tr>
<tr>
<td>5</td>
<td>0.5710</td>
<td>BK7</td>
<td>0.42</td>
</tr>
<tr>
<td>6</td>
<td>0.1300</td>
<td>SF61</td>
<td>0.62</td>
</tr>
<tr>
<td>7</td>
<td>0.4748</td>
<td>air</td>
<td>0.62</td>
</tr>
<tr>
<td>Image</td>
<td>flat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GENETIC PROGRAMMING BREEDS SIMULATABILITY

Unsimulatable individuals

GENETIC PROGRAMMING ENGINEERS AROUND EXISTING PATENTS AND CREATES NOVEL DESIGNS MORE FREQUENTLY THAN IT CREATES INFRINGING SOLUTIONS
GENETIC PROGRAMMING FREQUENTLY EXPLOITS A SIMULATOR’S BUILT-IN ASSUMPTION OF REASONABLENESS

CHARACTERISTICS SUGGESTING THE USE OF GENETIC PROGRAMMING

(1) discovering the size and shape of the solution,
(2) reusing substructures,
(3) discovering the number of substructures,
(4) discovering the nature of the hierarchical references among substructures,
(5) passing parameters to a substructure,
(6) discovering the type of substructures (e.g., subroutines, iterations, loops, recursions, or storage),
(7) discovering the number of arguments possessed by a substructure,
(8) maintaining syntactic validity and locality by means of a developmental process, or
(9) discovering a general solution in the form of a parameterized topology containing free variables.

MANY DIFFERENT GA/ES ENCODINGS HAVE BEEN SUCCESSFULLY USED

A mixture of real-valued variables, integer-valued variables, and categorical variables are encoded in the chromosome

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Component Value</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistor</td>
<td>2.5 Ω</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Bit-string chromosome
- The component type (a categorical variable) is encoded as 2 bits (01 = resistor, etc.)
- The component value (real-valued number) is encoded as 8 bits
- The node (integer-valued variable) to which the component's 1st lead is connected is encoded by 3 bits
- The node (integer-valued variable) to which the component's 2nd lead is connected is encoded by 3 bits
- Note that the number of nodes is capped at 8 (or assumed to be 8)

IT IS OFTEN POSSIBLE TO USE THE GENETIC ALGORITHM (GA) OR EVOLUTION STRATEGIES EVEN WHEN THE SIZE AND SHAPE OF THE SOLUTION IS A MAJOR ISSUE

- Variable-length genetic algorithm (VGA)
- Maintain constraints

Chromosome #1

<table>
<thead>
<tr>
<th>1st Component</th>
<th>2nd Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>0.220</td>
</tr>
<tr>
<td>C</td>
<td>403.0</td>
</tr>
<tr>
<td>L</td>
<td>0.120</td>
</tr>
<tr>
<td>L</td>
<td>0.160</td>
</tr>
<tr>
<td>L</td>
<td>0.221</td>
</tr>
<tr>
<td>L</td>
<td>0.261</td>
</tr>
<tr>
<td>L</td>
<td>0.421</td>
</tr>
<tr>
<td>L</td>
<td>0.630</td>
</tr>
</tbody>
</table>

Chromosome #2

<table>
<thead>
<tr>
<th>1st Component</th>
<th>2nd Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0.250</td>
</tr>
<tr>
<td>C</td>
<td>100.0</td>
</tr>
<tr>
<td>L</td>
<td>0.120</td>
</tr>
<tr>
<td>L</td>
<td>0.160</td>
</tr>
<tr>
<td>L</td>
<td>0.221</td>
</tr>
<tr>
<td>L</td>
<td>0.261</td>
</tr>
<tr>
<td>L</td>
<td>0.421</td>
</tr>
<tr>
<td>L</td>
<td>0.630</td>
</tr>
</tbody>
</table>

Nominal Offspring #1 is invalid

Penalize (in fitness measure)
- Delete
- Repair (most common method)
- Inundate
STRONG INDICATIONS FOR USING GENETIC ALGORITHM (GA) OR EVOLUTION STRATEGIES (ES)

- The size and shape of the solution is known or fixed
- Ascertaining numerical parameters is the major issue
- Simplicity is a major consideration
  - On-chip evolution the algorithm’s logic is implemented on the chip in hardware

AUTOMATIC SYNTHESIS OF A YAGI-UDA WIRE ANTENNA USING GENETIC ALGORITHM (LINDEN 1997)

- When the genetic algorithm (GA) operating on fixed-length character strings was used to synthesize a particular Yagi-Uda wire antenna by Linden (1997), the chromosome was based on
  - a particular number of reflectors (one) and
  - a particular number of directors.

The chromosome encoded
- the spacing between the parallel wires
- the length of each of the parallel wires

AUTOMATIC SYNTHESIS OF A YAGI-UDA WIRE ANTENNA USING GENETIC ALGORITHM (LINDEN 1997) — CONTINUED

- When the genetic algorithm (GA) operating on fixed-length character strings was used to synthesize a Yagi-Uda wire antenna (Linden 1997), the following decisions were made by the human user prior to the start of the run:
  1. the number of reflectors (one),
  2. the number of directors,
  3. the fact that the driven element, the directors, and the reflector are all single straight wires,
  4. the fact that the driven element, the directors, and the reflector are all arranged in parallel,
  5. the fact that the energy source (via the transmission line) is connected only to single straight wire (the driven element) — that is, all the directors and reflectors are parasitically coupled

- Characteristics (3), (4), and (5) are essential characteristics of the Yagi-Uda antenna, namely an antenna with multiple parallel parasitically coupled straight-line directors, a single parallel parasitically coupled straight-line reflector, and a straight-line driven element. That is, the GA run assumed that the answer would be a Yagi-Uda antenna.

EXAMPLE OF TURTLE FUNCTIONS USED TO CREATE WIRE ANTENNA

1 (PROGN3
2   (TURN-RIGHT 0.125)
3   (LANDMARK
4       (REPEAT 2
5         (PROGN2
6           (DRAW 1.0 HALF-MM-WIRE)
7             (DRAW 0.5 NO-WIRE)))
8   (TRANSLATE-RIGHT 0.125 0.75))

 characteristics (a) through (g)
The GP run discovered
(1) the number of reflectors (one),
(2) the number of directors,
(3) the fact that the driven element, the directors, and the reflector are all single straight wires,
(4) the fact that the driven element, the directors, and the reflector are all arranged in parallel,
(5) the fact that the energy source (via the transmission line) is connected only to single straight wire (the driven element) — that is, all the directors and reflectors are parasitically coupled

Characteristics (3), (4), and (5) are essential characteristics of the Yagi-Uda antenna, namely an antenna with multiple parallel parasitically coupled straight-line directors, a single parallel parasitically coupled straight-line reflector, and a straight-line driven element.
REUSE
LOWPASS FILTER USING ADFs
GEN 20 – FOUR-RUNG LADDER

QUADRUPLY-CALLED TWO-PORTED
ADF0

BEHAVIOR IN FREQUENCY DOMAIN

REUSE
LOWPASS FILTER USING ADFs
GENERATION 31 — TOPOLOGY OF
CAUER (ELLIPTIC) FILTER

QUINTUPLY-CALLED THREE-PORTED
ADF0

BEHAVIOR IN FREQUENCY DOMAIN

PASSING A PARAMETER TO A
SUBSTRUCTURE

- The set of potential terminals for each construction-
  continuing subtree of an automatically defined function,
  $T_{ccs-ADF-potential}$, is

$$T_{ccs-ADF-potential} = \{ ARG0 \}$$

EMERGENCE OF A PARAMETERIZED
ARGUMENT IN A CIRCUIT
SUBSTRUCTURE

HIERARCHY OF BRANCHES FOR THE
BEST-OF-RUN CIRCUIT FROM
GENERATION 158

PASSING A PARAMETER TO A
SUBSTRUCTURE

BEST-OF-RUN CIRCUIT FROM
GENERATION 158
THREE-PORTED AUTOMATICALLY DEFINED FUNCTION ADF3 OF THE BEST-OF-RUN CIRCUIT FROM GENERATION 158

ADF3 CONTAINS CAPACITOR C39 PARAMETERIZED BY DUMMY VARIABLE ARG0

ADF3 DOES THREE THINGS

• The structure that develops out of ADF3 includes a capacitor C112 whose value (5,130 uF) is not a function of its dummy variable, ARG0.

• The structure that develops out of ADF3 has one hierarchical reference to ADF2. As previously mentioned, the invocation of ADF2 is done with a constant (9.737455E-01) so this invocation of ADF2 produces a 259 µH inductor.

• Most importantly, the structure that develops out of ADF3 creates a capacitor (C39) whose sizing, $F(ARG0)$, is a function of the dummy variable, ARG0, of automatically defined function ADF3. Capacitor C39 has different sizing on different invocations of automatically defined function ADF3.

• The combined effect of ADF3 is to insert the following three components:
  • an unparameterized 5,130 uF capacitor,
  • a parameterized capacitor C39 whose component value is dependent on ARG0 of ADF3, and
  • a parameterized inductor (created by ADF2) whose sizing is parameterized, but which, in practice, is called with a constant value.

THE FIRST RESULT-PRODUCING BRANCH, RPB0, CALLING ADF3

AUTOMATICALLY DEFINED FUNCTION ADF3

EMERGENCE OF A PARAMETERIZED ARGUMENT IN A CIRCUIT SUBSTRUCTURE

HIERARCHY OF BRANCHES FOR THE BEST-OF-RUN CIRCUIT- FROM GENERATION 158
FREE VARIABLE (INPUT) AND CONDITIONALS

SOLVING A QUADRATIC EQUATION USING THE GENETIC ALGORITHM

• Suppose we want the 2 roots of the quadratic equation

\[ 1x^2 - 3x + 2 = 0 \]

• Using the genetic algorithm (GA) operating on a fixed-length character string, we can search a space of encodings using an alphabet size of 2 (i.e., binary) of length, say, 16 representing two real numbers (each with, say, 4 bits to left of the "decimal" point). After running the GA, a solution is

```
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
```

1.0 2.0

• Alternatively, we could use a "floating point" genetic algorithm (GA) to search a space of 2-part encodings. A solution is

```
1.0 2.0
```

• In either case, the result is a solution to ONE INSTANCE of the quadratic equation problem.

GENERAL APPEARANCE OF ONE POSSIBLE CHROMOSOME ENCODING USED TO SOLVE ONE INSTANCE OF A CIRCUIT PROBLEM USING THE GENETIC ALGORITHM (GA) OPERATING ON FIXED-LENGTH CHARACTER STRINGS

EXAMPLE CIRCUIT

SOLVING A QUADRATIC EQUATION USING GENETIC PROGRAMMING (GP)

• Using genetic programming (GP), we can solve the general, parameterized quadratic equation

\[ ax^2 + bx + c = 0 \]

by searching the space of computer programs for a program that takes \( a \), \( b \), and \( c \) as inputs

THE GENERAL APPEARANCE OF EXPRESSIONS USED TO SOLVE ONE INSTANCE OF A CIRCUIT PROBLEM USING GENETIC PROGRAMMING (GP) IN GENETIC PROGRAMMING III (1999)

EXAMPLE CIRCUIT (GEN 0)
PARAMETERIZED TOPOLOGY USING CONDITIONAL DEVELOPMENTAL OPERATORS (GENETIC SWITCH)

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT

- Best-of-run circuit from generation 93 when inputs call for a highpass filter (i.e., $F_1 > F_2$).

- Best-of-run circuit from generation 93 when inputs call for a lowpass filter.

PARAMETERIZED TOPOLOGY FOR "GENERALIZED" LOWPASS FILTER

VARIABLE CUTOFF LOWPASS FILTER

- Want lowpass filter whose passband ends at frequencies $f = 1,000, 1,780, 3,160, 5,620, 10,000, 17,800, 31,600, 56,200, 100,000$ Hz

**PETA-OPS**

- Human brain operates at $10^{12}$ neurons operating at $10^3$ per second = $10^{15}$ ops per second
- $10^{15}$ ops = 1 peta-op = 1 bs (brain second)
GENETIC PROGRAMMING OVER 15-YEAR PERIOD 1987–2002

<table>
<thead>
<tr>
<th>System</th>
<th>Period of usage</th>
<th>Petacycles (10^12 cycles) per day for entire system</th>
<th>Speed-up over previous system</th>
<th>Speed-up over first system in this table</th>
<th>Human-competitive results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Texas Instruments LISP machine</td>
<td>1987–1994</td>
<td>0.00216</td>
<td>1 (base)</td>
<td>1 (base)</td>
<td>0</td>
</tr>
<tr>
<td>64-node Transtech transputer parallel machine</td>
<td>1994–1997</td>
<td>0.02</td>
<td>9</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>64-node Parsytec parallel machine</td>
<td>1995–2000</td>
<td>0.44</td>
<td>22</td>
<td>204</td>
<td>12</td>
</tr>
<tr>
<td>70-node Alpha parallel machine</td>
<td>1999–2001</td>
<td>3.2</td>
<td>7.3</td>
<td>1,481</td>
<td>2</td>
</tr>
<tr>
<td>1,000-node Pentium II parallel machine</td>
<td>2000–2002</td>
<td>30.0</td>
<td>9.4</td>
<td>13,900</td>
<td>12</td>
</tr>
</tbody>
</table>

PROGRESSION OF RESULTS

<table>
<thead>
<tr>
<th>System</th>
<th>Period</th>
<th>Speed-up Qualitative nature of the results produced by genetic programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial LISP machine</td>
<td>1987–1994</td>
<td>1 (base) • Toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning</td>
</tr>
<tr>
<td>64-node Transtech 8-bit transputer</td>
<td>1994–1997</td>
<td>9 • Two human-competitive results involving one-dimensional discrete data (not patent-related)</td>
</tr>
<tr>
<td>64-node Parsytec parallel machine</td>
<td>1995–2000</td>
<td>22 • One human-competitive result involving two-dimensional discrete data • Numerous human-competitive results involving continuous signals analyzed in the frequency domain • Numerous human-competitive results involving 20th-century patented inventions</td>
</tr>
<tr>
<td>70-node Alpha parallel machine</td>
<td>1999–2001</td>
<td>7.3 • One human-competitive result involving continuous signals analyzed in the time domain • Circuit synthesis extended from topology and sizing to include routing and placement (layout)</td>
</tr>
<tr>
<td>1,000-node Pentium II parallel machine</td>
<td>2000–2002</td>
<td>9.4 • Numerous human-competitive results involving continuous signals analyzed in the time domain • Numerous general solutions to problems in the form of parameterized topologies • Six human-competitive results duplicating the functionality of 21st-century patented inventions</td>
</tr>
<tr>
<td>Long (4-week) runs of 1,000-node Pentium II parallel machine</td>
<td>2002</td>
<td>9.3 • Generation of two patentable new inventions</td>
</tr>
</tbody>
</table>

PROGRESSION OF QUALITATIVELY MORE SUBSTANTIAL RESULTS PRODUCED BY GENETIC PROGRAMMING IN RELATION TO FIVE ORDER-OF-MAGNITUDE INCREASES IN COMPUTATIONAL POWER

• toy problems
• human-competitive results not related to patented inventions
• 20th-century patented inventions
• 21st-century patented inventions
• patentable new inventions

EVOLVABLE HARDWARE

RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

SMALL 5 BY 5 CORNER OF XILINX XC6216 FPGA
EVOLVABLE HARDWARE
RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

SORTING NETWORKS

• A 16-step 7-sorter was evolved that has two fewer steps than the sorting network described in O'Connor and Nelsons' patent (1962) and that has the same number of steps as the 7-sorter that was devised by Floyd and Knuth subsequent to the patent and described in Knuth 1973.

GENETICALLY EVOLVED 7-SORTER

FUNDAMENTAL DIFFERENCES BETWEEN GP AND OTHER APPROACHES TO AI AND ML

(1) Representation: Genetic programming overtly conducts it search for a solution to the given problem in program space.
(2) Role of point-to-point transformations in the search: Genetic programming does not conduct its search by transforming a single point in the search space into another single point, but instead transforms a set of points into another set of points.
(3) Role of hill climbing in the search: Genetic programming does not rely exclusively on greedy hill climbing to conduct its search, but instead allocates a certain number of trials, in a principled way, to choices that are known to be inferior.
(4) Role of determinism in the search: Genetic programming conducts its search probabilistically.
(5) Role of an explicit knowledge base: None.
(6) Role of formal logic in the search: None.
(7) Underpinnings of the technique: Biologically inspired.

EIGHT CRITERIA FOR HUMAN-COMPETITIVENESS

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.</td>
</tr>
<tr>
<td>B</td>
<td>The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.</td>
</tr>
<tr>
<td>C</td>
<td>The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.</td>
</tr>
<tr>
<td>D</td>
<td>The result is equal to or better than a result that was published in its own right as a new scientific result, independent of the fact that the result was mechanically created.</td>
</tr>
<tr>
<td>E</td>
<td>The result is equal to or better than the most recent human-created solution to a problem for which there has been a succession of increasingly better human-created solutions.</td>
</tr>
<tr>
<td>F</td>
<td>The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.</td>
</tr>
<tr>
<td>G</td>
<td>The result solves a problem of undisputable difficulty in its field.</td>
</tr>
<tr>
<td>H</td>
<td>The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).</td>
</tr>
</tbody>
</table>

37 HUMAN-COMPETITIVE RESULTS (LIST AS OF APRIL 2004)

<table>
<thead>
<tr>
<th>Claimed instance</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa &quot;early promise&quot; problem</td>
<td><img src="image1.png" alt="Picture" /></td>
</tr>
<tr>
<td>Specter, Barnum, and Bernstein 1998</td>
<td></td>
</tr>
<tr>
<td>Creation of a better-than-classical quantum algorithm for Grover's database search problem</td>
<td><img src="image2.png" alt="Picture" /></td>
</tr>
<tr>
<td>Specter, Barnum, and Bernstein 1999</td>
<td></td>
</tr>
</tbody>
</table>
Creation of a quantum algorithm for the depth-two AND/OR query problem that is better than any previously published result
Spector, Barnum, Bernstein, and Swamy 1999; Barnum, Bernstein, and Spector 2000

Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously published result
Barnum, Bernstein, and Spector 2000

Creation of a protocol for communicating information through a quantum gate that was previously thought not to permit such communication
Spector and Bernstein 2003

To understand one needs to know what the Smolin gate is and this is given in smolin-gate.jpg

\[
\text{Smolin} = \begin{bmatrix}
\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}}
\end{bmatrix}
\]

Creation of a novel variant of quantum dense coding
Spector and Bernstein 2003

To understand one needs to know what the BS gate is and this is given in bs-gate.jpg

\[
\text{BS}(\theta) = \begin{bmatrix}
\cos(\theta) & 0 & 0 & \sin(\theta) \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
\sin(\theta) & 0 & 0 & -\cos(\theta)
\end{bmatrix}
\]

Creation of a soccer-playing program that won its first two games in the Robo Cup 1997 competition

Creation of a soccer-playing program that ranked in the middle of the field of 34 human-written programs in the Robo Cup 1998 competition
Andre and Teller 1999
<table>
<thead>
<tr>
<th>Residue</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, F, I, L, M, or V</td>
<td>0</td>
</tr>
</tbody>
</table>

Creation of four different algorithms for the transmembrane segment identification problem for proteins

Sections 18.8 and 18.10 of Genetic Programming II and sections 16.5 and 17.2 of Genetic Programming III

Rediscovery of the Zobel "M-derived half section" and "constant K" filter sections

Section 25.15.2 of Genetic Programming III

Automatic decomposition of the problem of synthesizing a crossover (woofer-tweeter) filter

Section 32.3 of Genetic Programming III

Rediscovery of the Zobel "M-derived half section" and "constant K" filter sections

Section 25.15.2 of Genetic Programming III

Automatic decomposition of the problem of synthesizing a crossover (woofer-tweeter) filter

Section 32.3 of Genetic Programming III

Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits

Section 42.3 of Genetic Programming III

Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits

Section 42.3 of Genetic Programming III

Synthesis of analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions

Section 47.5.3 of Genetic Programming III

Gaussian computational circuit using MOSFET transistors

Synthesis of 60 and 96 decibel amplifiers

Section 45.3 of Genetic Programming III

Synthesis of 60 and 96 decibel amplifiers

Section 45.3 of Genetic Programming III

Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits

Section 42.3 of Genetic Programming III

Gaussian computational circuit using MOSFET transistors

Synthesis of analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions

Section 47.5.3 of Genetic Programming III

Gaussian computational circuit using MOSFET transistors

Synthesis of 60 and 96 decibel amplifiers

Section 45.3 of Genetic Programming III

Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits

Section 42.3 of Genetic Programming III

Gaussian computational circuit using MOSFET transistors

Synthesis of analog computational circuits for squaring, cubing, square root, cube root, logarithm, and Gaussian functions

Section 47.5.3 of Genetic Programming III

Gaussian computational circuit using MOSFET transistors

Synthesis of 60 and 96 decibel amplifiers

Section 45.3 of Genetic Programming III
Synthesis of a real-time analog circuit for time-optimal control of a robot
Section 48.3 of Genetic Programming III

Synthesis of an electronic thermometer
Section 49.3 of Genetic Programming III

Synthesis of a voltage reference circuit
Section 50.3 of Genetic Programming III

Creation of a cellular automata rule for the majority classification problem that is better than the Gacs-Kurdyumov-Levin (GKL) rule and all other known rules written by humans

Andre, Bennett, and Koza 1996 and section 58.4 of Genetic Programming III

Rule | State Transition Rule | Accuracy |
--- | --- | --- |
Gacs-Kurdyumov-Levin (GKL) 1978 human-written | 00000000 01011111 00000000 01011111 00000000 01011111 00000000 01011111 11111111 01011111 | 81.6% |
Davis 1995 human-written | 00000000 00101111 00000011 01011111 00000000 00011111 11001111 00011111 00000000 00101111 11111100 01011111 00000000 00011111 | 81.800% |
Das (1995) human-written | 00000111 00000000 00000111 11111111 00001111 00000000 00001111 11111111 00001111 00000000 00000111 11111111 00001111 00110001 00001111 11111111 | 82.178% |
Best rule evolved by genetic programming (1999) | 00000101 00000000 01010101 00000101 00000101 00000000 01010101 00000101 01010101 11111111 01010101 11111111 01010101 11111111 01010101 11111111 | 82.326% |

Creation of motifs that detect the D- E- A- D box family of proteins and the manganese superoxide dismutase family
Section 59.8 of Genetic Programming III

[IV]-[lim]-D-E-(AI)6-[lim]-[lim]-[-lim]-[lim]-[lim]-[lim]-[lim]
Synthesis of topology for a PID-2 (proportional, integrative, derivative, and second derivative) controller
Section 3.7 of Genetic Programming IV

Synthesis of an analog circuit equivalent to Philbrick circuit
Section 4.3 of Genetic Programming IV

Synthesis of a NAND circuit
Section 4.4 of Genetic Programming IV

Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits

Rediscovery of negative feedback
Chapter 14 of Genetic Programming IV

Synthesis of a low-voltage balun circuit
Section 15.4.1 of Genetic Programming IV

Synthesis of a mixed analog-digital variable capacitor circuit
Section 15.4.2 of Genetic Programming IV

Synthesis of a high-current load circuit
Section 15.4.3 of Genetic Programming IV
Synthesis of a voltage-current conversion circuit
Section 15.4.4 of Genetic Programming IV

Synthesis of a cubic function generator
Section 15.4.5 of Genetic Programming IV

Synthesis of a tunable integrated active filter
Section 15.4.6 of Genetic Programming IV

Creation of PID tuning rules that outperform the Ziegler-Nichols and Åström-Hägglund tuning rules
Chapter 12 of Genetic Programming IV

The topology (above) was not evolved, but was the standard PID topology. Evolved equations for $K_p$-final, $K_i$-final, $K_d$-final, and $b$-final:

$$K_p\text{-final} = 0.32 \times K_p \times \frac{-1.72 \times 0.002540 \times \frac{\tau_p}{\tau}} - 0.170 \times 10^5$$

$$K_i\text{-final} = 0.22 \times K_i \times \frac{-1.2 \times 0.003525 \times \frac{\tau_p}{\tau}}$$

$$K_d\text{-final} = 0.106 \times K_d \times \frac{-1.2 \times 0.003525 \times \frac{\tau_p}{\tau}}$$

$$b\text{-final} = 0.23 \times \frac{0.1}{\tau_p}$$

The above topology and equations 31, 32, 33, and 34 were evolved:

$$\frac{|e(\tau) - e(\tau + 1)|}{|e(\tau) + |e(\tau)|^2|}$$ [31]

$$|e(\tau)|$$ [34]

$$NLM(\log|e(\tau)|) \times e(\tau)$$ [32]

$$NLM(\log|e(\tau)|) \times e(\tau)$$ [33]
PROMISING GP APPLICATION AREAS

- Problem areas involving many variables that are interrelated in highly non-linear ways
- Inter-relationship of variables is not well understood
- A good approximate solution is satisfactory
  - design
  - control
  - classification and pattern recognition
  - data mining
  - system identification and forecasting
- Discovery of the size and shape of the solution is a major part of the problem
- Areas where humans find it difficult to write programs
  - parallel computers
  - cellular automata
  - multi-agent strategies / distributed AI
  - FPGAs
- "black art" problems
  - synthesis of topology and sizing of analog circuits
  - synthesis of topology and tuning of controllers
  - quantum computing circuits
  - synthesis of designs for antennas
- Areas where you simply have no idea how to program a solution, but where the objective (fitness measure) is clear
- Problem areas where large computerized databases are accumulating and computerized techniques are needed to analyze the data

TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE

- Turing made the connection between searches and the challenge of getting a computer to solve a problem without explicitly programming it in his 1948 essay "Intelligent Machines" (in Mechanical Intelligence: Collected Works of A. M. Turing, 1992, edited by D. C. Ince).

"Further research into intelligence of machinery will probably be very greatly concerned with 'searches' ...

TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE — CONTINUED

1. LOGIC-BASED SEARCH
   One approach that Turing identified is a search through the space of integers representing candidate computer programs.

2. CULTURAL SEARCH
   Another approach is the "cultural search" which relies on knowledge and expertise acquired over a period of years from others (akin to present-day knowledge-based systems).
3. GENETICAL OR EVOLUTIONARY SEARCH

"There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value."

- from Turing's 1950 paper "Computing Machinery and Intelligence" ...

"We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications"

"Structure of the child machine = Hereditary material"

"Changes of the child machine = Mutations"

"Natural selection = Judgment of the experimenter"

### MAIN POINTS OF JAWS-1,2,3,4 BOOKS

<table>
<thead>
<tr>
<th>Book</th>
<th>Main Points</th>
</tr>
</thead>
</table>
| 1992       | • Virtually all problems in artificial intelligence, machine learning, adaptive systems, and automated learning can be recast as a search for a computer program.  
• Genetic programming provides a way to successfully conduct the search for a computer program in the space of computer programs. |
| 1994       | • Scalability is essential for solving non-trivial problems in artificial intelligence, machine learning, adaptive systems, and automated learning.  
• Scalability can be achieved by reuse.  
• Genetic programming provides a way to automatically discover and reuse subprograms in the course of automatically creating computer programs to solve problems. |
| 1999       | • Genetic programming possesses the attributes that can reasonably be expected of a system for automatically creating computer programs. |
| 2003       | • Genetic programming now routinely delivers high-return human-competitive machine intelligence.  
• Genetic programming is an automated invention machine.  
• Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.  
• Genetic programming has delivered a progression of qualitatively more substantial results in synchrony with five approximately order-of-magnitude increases in the expenditure of computer time. |

### SOME RECENT CONFERENCE PROCEEDINGS

**ASGP**


**GECCO**


**EURO-GP**


**GP Conference (Now part of GECCO)**


**GPTP**

3 EDITED ADVANCES IN GENETIC PROGRAMMING BOOKS


4 VIDEOTAPES ON GP


WILLIAM LANGDON’S BIBLIOGRAPHY ON GENETIC PROGRAMMING

This bibliography is the most extensive in the field and contains over 3,034 papers (as of January 2003) by over 880 authors.

Visit http://www.cs.bham.ac.uk/~wbl/biblio/ or http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html

GENETIC PROGRAMMING AND EVOLVABLE MACHINES JOURNAL FROM KLUWER ACADEMIC PUBLISHERS (NOW SPRINGER)

Editor: Wolfgang Banzhaf

GENETIC PROGRAMMING BOOK SERIES FROM KLUWER ACADEMIC PUBLISHERS (NOW SPRINGER)

Editor: John Koza
koza@stanford.edu

GP MAILING LIST

To subscribe to the Genetic Programming e-mail list,
• send e-mail message to:
  genetic_programming-subscribe@yahoogroups.com
• visit the web page
  http://groups.yahoo.com/group/genetic_programming/

INTERNATIONAL SOCIETY FOR GENETIC AND EVOLUTIONARY COMPUTATION (ISGEC)

For information on ISGEC, the annual GECCO conference, or the bi-annual FOGA workshop, visit www.isgec.org

FOR ADDITIONAL INFORMATION ON THE GP FIELD

Visit http://www.genetic-programming.org for
• links computer code in various programming languages (including C, C++, Java, Mathematica, LISP)
• partial list of people active in genetic programming
• list of known completed PhD theses on GP
• list of students known to be working on PhD theses on GP
• information for instructors of university courses on genetic algorithms and genetic programming