INTRODUCTION TO GENETIC PROGRAMMING

TUTORIAL

GECCO-2008—ATLANTA
JULY 12–16, 2008

John R. Koza
Stanford University
E-MAIL: koza@stanford.edu
http://www.johkoza.com
http://www.genetic-programming.org

OUTLINE

- **Reason for genetic programming (GP)**
- **The GP algorithm (flowchart, ops, examples)**
- **Developmental GP**
  - **Analog electrical circuits**
  - **Optical lens systems**
  - **Antenna**
  - **Automatic parallelization of programs**
- **Cross-domain features**
- **Reuse**
- **Parameterized topologies**
- **Parallel computing**
- **Qualitative progression of results (Moore's law)**
- **Evolvable hardware**
- **Differences between GP and AI and ML**
- **Human-competitive results**
- **Promising GP application areas**
- **Sources of additional information**

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.**

REASON FOR GENETIC PROGRAMMING

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)

GECCO'08, July 12-16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-131-6/08/07.

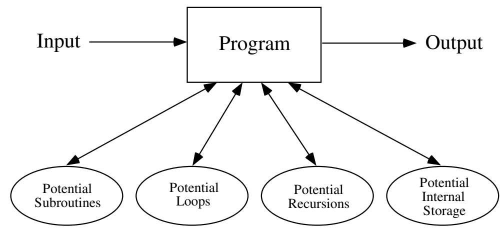
5

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)**

6

A COMPUTER PROGRAM



REPRESENTATION

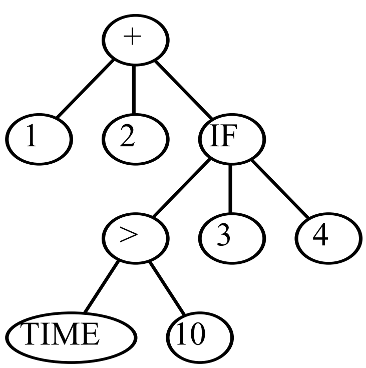
- **"Our view is that computer programs are the best representation of computer programs."**

7

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

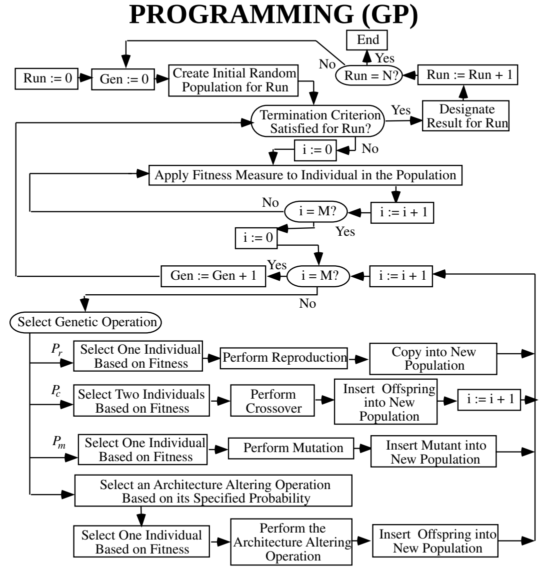
`(+ 1 2 (IF (> TIME 10) 3 4))`

- **Terminal set T = {`1, 2, 10, 3, 4, TIME`}**
- **Function set F = {`+, IF, >`}**



8

FLOWCHART FOR GENETIC PROGRAMMING (GP)

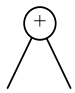


9

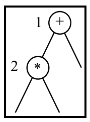
EXAMPLE OF RANDOM CREATION OF A PROGRAM TREE

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

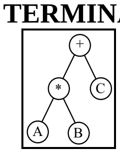
BEGIN WITH TWO-ARGUMENT `+`



CONTINUE WITH TWO-ARGUMENT `*`



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



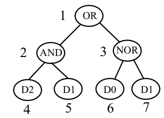
- **The result is a syntactically valid executable program (provided the set of functions is closed)**

10

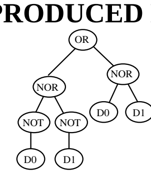
MUTATION OPERATION

- **Select parent probabilistically based on fitness**
- **Pick point from 1 to `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



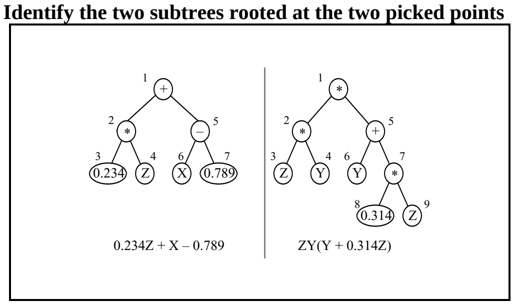
OFFSPRING PRODUCED BY MUTATION



11

CROSSOVER (SEXUAL RECOMBINATION) OPERATION FOR COMPUTER PROGRAMS

- **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**



Parent 1:

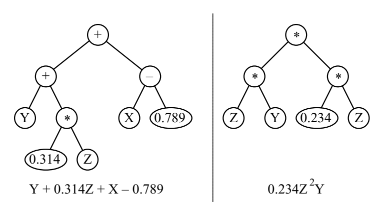
```
(+ (* 0.234 Z) (- X 0.789))
```

Parent 2:

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

12

THE CROSSOVER OPERATION (TWO OFFSPRING VERSION)



Offspring 1:

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

Offspring 2:

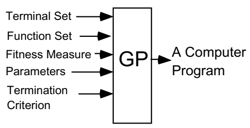
```
(* (* Z Y) (* 0.234 Z))
```

- **The result is a syntactically valid executable program**

13

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**



14

SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #1 (WITH 21 FITNESS CASES)

Independent variable *X* (Input)	Dependent Variable *Y* (Output)
-1.0	1.00
-0.9	0.91
-0.8	0.84
-0.7	0.79
-0.6	0.76
-0.5	0.75
-0.4	0.76
-0.3	0.79
-0.2	0.84
-0.1	0.91
0	1.00
0.1	1.11
0.2	1.24
0.3	1.39
0.4	1.56
0.5	1.75
0.6	1.96
0.7	2.19
0.8	2.44
0.9	2.71
1.0	3.00

15

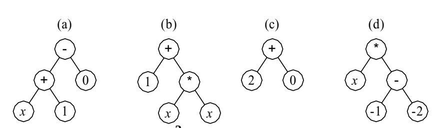
TABLEAU FOR SYMBOLIC OF UNKNOWN FUNCTION #1

	Objective:	Find a computer program with one input (independent variable *x*), whose output equals the values in the table in range from -1 to +1.
1	Terminal set:	`T = {X, 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 $x^2 + x + 1$.
4	Parameters:	Population size M = 4.
5	Termination:	An individual emerges whose sum of absolute errors is less than 0.1

16

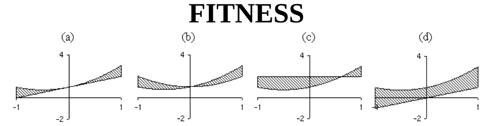
SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #1

INITIAL POPULATION OF FOUR RANDOMLY CREATED INDIVIDUALS OF GENERATION 0



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

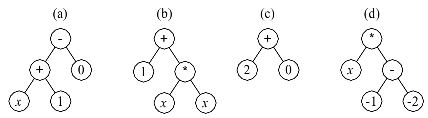
FITNESS



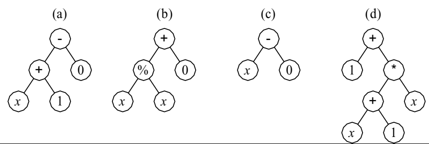
0.67 1.00 1.70 2.67

17

SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #1



GENERATION 1



$x + 1$	**1**	**x**	**$x^2 + x + 1$**
Copy of (a)	**Mutant of (c)**	**1st offspring of crossover of (a) and (b)**	**2nd offspring of crossover of (a) and (b)**
	Picking "2" as mutation point	**Picking "+" of parent (a) and left-most "x" of parent (b) as crossover points**	**Picking "+" of parent (a) and left-most "x" of parent (b) as crossover points**

18

SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #1 —QUADRATIC POLYNOMIAL $X^2 + X + 1$

OBSERVATIONS

- **GP works on this very simple illustrative 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)**
- **Most importantly, the solution resulted from a recombination (crossover) of two "pretty good" elements, namely**
  - **the linear term x**
  - **the quadratic term $x^2 + 1$**
- **The answer is algebraically correct (hence no further cross validation is needed)**

19

SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2 (WITH 21 FITNESS CASES)

Independent variable X (Input)	Dependent Variable Y (Output)
-1.0	0.0000
-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

20

TABLEAU—SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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:	`+, -, *, %, SIN, COS, EXP, 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 UNKNOWN FUNCTION #2

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

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

Equivalent to

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

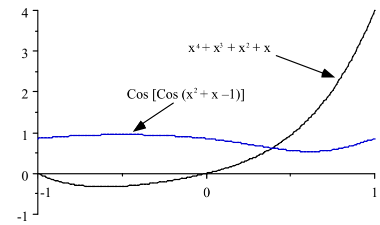
SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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

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

Equivalent to

$$\mathrm{Cos}\,[\mathrm{Cos}\,(x^2 + x - 1)]$$



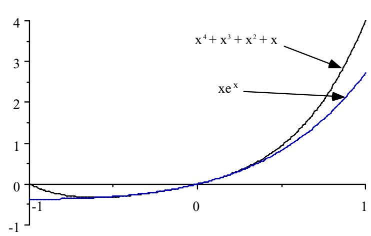
SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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

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

Equivalent to

$$xe^x$$



SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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**
- **Parameters**
  - **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 <u>PROBABILISTICALLY, BASED ON FITNESS</u>**
  - **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**

25

SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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

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

Equivalent to...

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

26

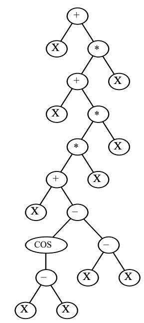
SYMBOLIC REGRESSION OF UNKNOWN FUNCTION #2

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

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



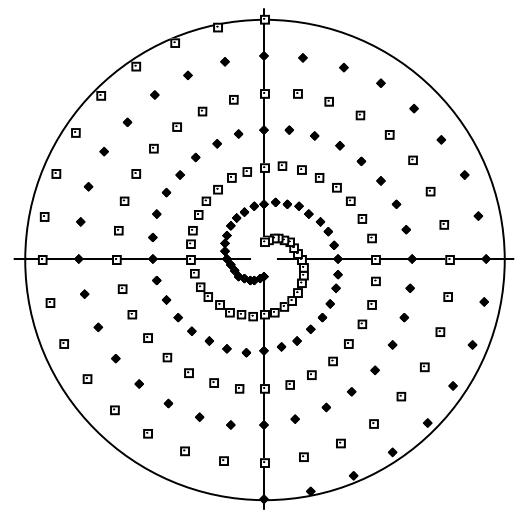
27

SYMBOLIC REGRESSION OF FUNCTION #2—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`, `RLOG`, 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)**

28

CLASSIFICATION PROBLEM INTER-TWINED SPIRALS



29

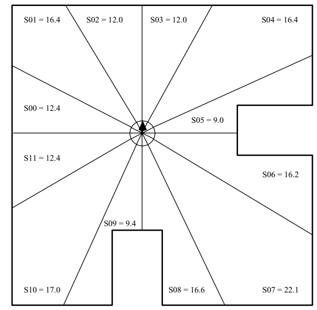
GP TABLEAU – INTERTWINED SPIRALS

Objective:	**Find a program to classify a given point in the *x*-*y* plane to the red or blue spiral.**
Terminal set:	**`X`, `Y`, 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.**

30

WALL-FOLLOWING PROBLEM

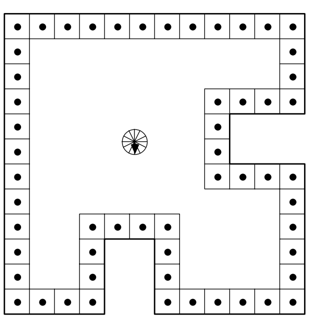
12 SONAR SENSORS



31

WALL-FOLLOWING PROBLEM

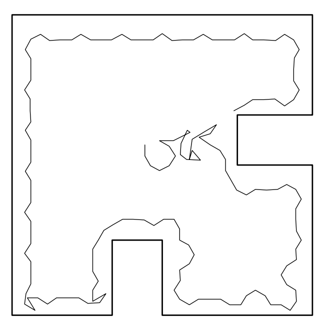
FITNESS MEASURE



32

WALL-FOLLOWING PROBLEM
BEST PROGRAM OF GENERATION 57

- **Scores 56 hits (out of 56)**
- **145point program tree**



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 3√2**
- **Econometric Exchange Equation**
- **Optimization (Lizard)**
- **Boolean 11-Multiplexer**
- **11-Parity–Automatically Defined Functions**

DEVELOPMENTAL GA

- **Wilson 1987**
- **Kitano 1990**

DEVELOPMENTAL GP

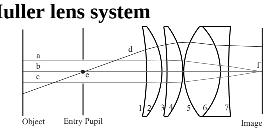
CELLULAR ENCODING (DEVELOPMENTAL GENETIC PROGRAMMING)

- **Gruau, Frederic. 1992b. *Cellular Encoding of Genetic Neural Networks*. Technical report 92-21. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieure de Lyon. May 1992.**
- **Also: Gruau 1992a 1992b 1993 1994a 1994b; Gruau and Whitley 1993; Esparcia-Alcazar and Sharman 1997)**
- **Applied by Gruau and Whitley (1995) to 2-pole-balancing problem**
- **Applied by Gruau to six-legged walking creature**
- **Applied by Brave (1995, 1996) to finite automata**
- **Analog electrical circuits (Koza, Bennett, Andre, Keane 1995)**
- **Ontogenetic genetic programming (Spector and Stoffel 1996)**

DEVELOPMENTAL GP

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

Tackaberry-Muller lens system

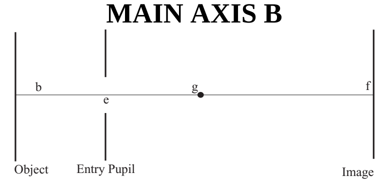


Lens file for Tackaberry-Muller system

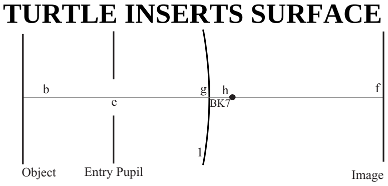
Surface	Distance	Radius	Material	Aperture
Object	10^{10}	flat	air	
Entry pupil	0.88	flat	air	0.18
1	0.21900	–3.5236	BK7	0.62
2	0.07280	–1.0527	air	0.62
3	0.22500	–4.4072	BK7	0.62
4	0.01360	–1.0704	air	0.62
5	0.52100	1.02491	BK7	0.62
6	0.11800	–0.9349	SF61	0.62
7	0.47485	7.94281	air	0.62
Image		flat		

DEVELOPMENTAL PROCESS

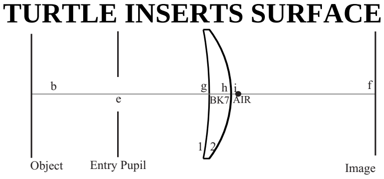
TURTLE STARTS AT POINT G ALONG MAIN AXIS B



TURTLE INSERTS SURFACE 1

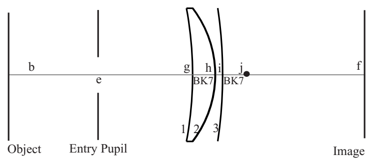


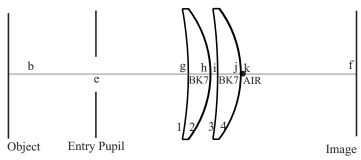
TURTLE INSERTS SURFACE 2



37

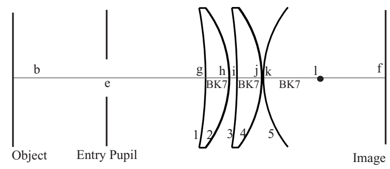
DEVELOPMENTAL PROCESS—CONTINUED

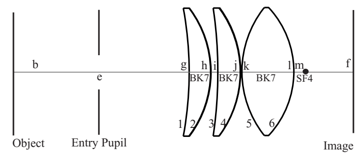


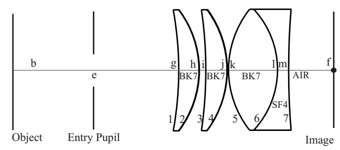


38

DEVELOPMENTAL PROCESS—CONTINUED



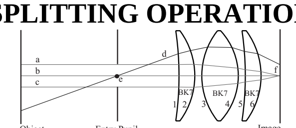




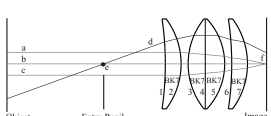
39

LENS SPLITTING OPERATION

LENS SYSTEM BEFORE LENS-SPLITTING OPERATION



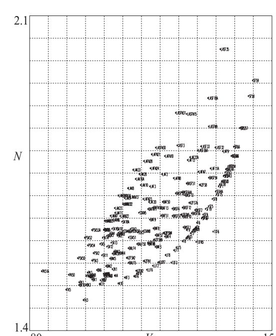
LENS SYSTEM AFTER LENS-SPLITTING OPERATION



40

GLASS MUTATION

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



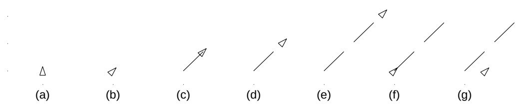
41

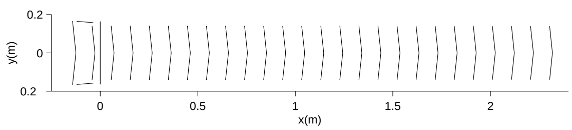
DEVELOPMENTAL GP

AUTOMATIC SYNTHESIS OF ANTENNA

EXAMPLE OF TURTLE FUNCTIONS

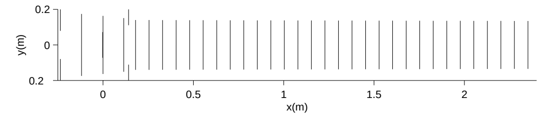
```
(PROGN3
  (TURN-RIGHT 0.125)
  (LANDMARK
    (REPEAT 2
      (PROGN2
        (DRAW 1.0 HALF-MM-WIRE)
        (DRAW 0.5 NO-WIRE)))
  (TRANSLATE-RIGHT 0.125 0.75))
```





42

BEST-OF-RUN ANTENNA FROM GENERATION 90 — FITNESS OF-16.04

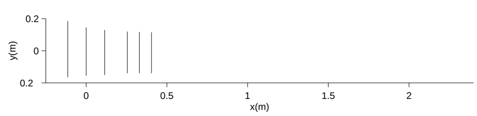


• The GP run discovered
- **(1) the number of reflectors (one),**
- **(2) the number of directors,**
- **(3) the fact that the driven element, the directors, and the reflector are all single straight wires,**
- **(4) the fact that the driven element, the directors, and the reflector are all arranged 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.

43

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.**

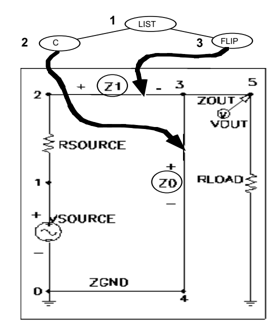
The chromosome encoded
- **the spacing between the parallel wires**
- **the length of each of the parallel wires**

44

DEVELOPMENTAL GP

ANALOG ELECTRICAL CIRCUITS

THE INITIAL CIRCUIT



DEVELOPMENTAL GP

ANALOG ELECTRICAL CIRCUITS

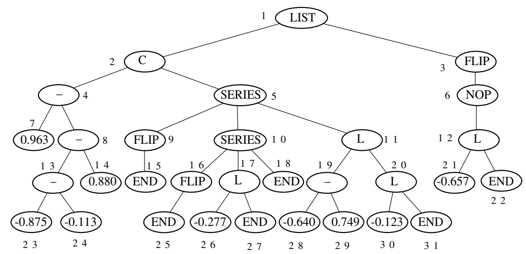
THE INITIAL CIRCUIT

- **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

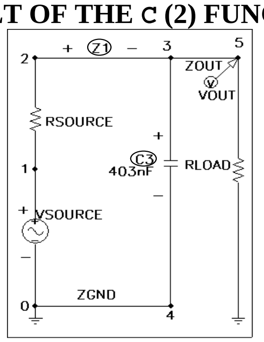
DEVELOPMENT OF A CIRCUIT FROM A CIRCUIT-CONSTRUCTING PROGRAM TREE AND THE INITIAL CIRCUIT

(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

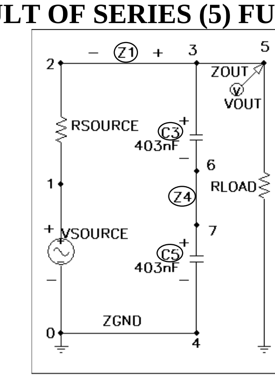


(LIST (C (– 0.963 (– (– -0.875 -0.113) 0.880)) (series (flip end) (series (flip end) (L -0.277 end) end) (L (– -0.640 0.749) (L -0.123 end)))) (flip (nop (L -0.657 end)))))

NOTE: Interpretation of arithmetic value

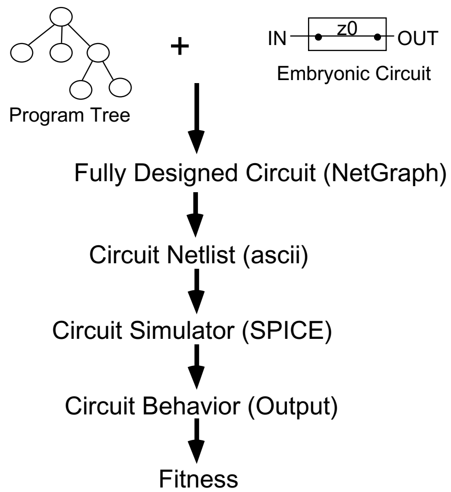
DEVELOPMENTAL GP

RESULT OF SERIES (5) FUNCTION

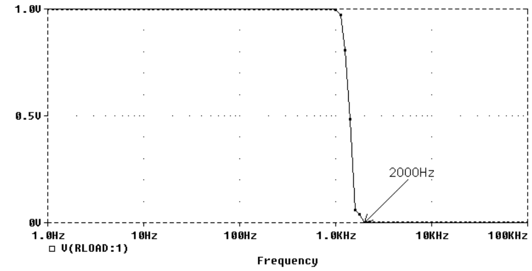


(LIST (C (– 0.963 (– (– -0.875 -0.113) 0.880)) (series (flip end) (series (flip end) (L -0.277 end) end) (L (– -0.640 0.749) (L -0.123 end)))) (flip (nop (L -0.657 end)))))

EVALUATION OF FITNESS OF A CIRCUIT



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 $F(t) = \sum_{i=0}^{100} [W(f_i)d(f_i)]$

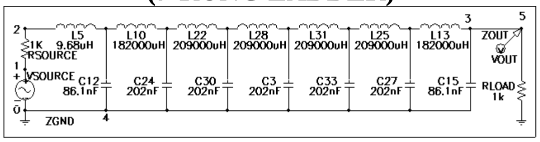
- $f(i)$ **is the frequency of fitness case** i
- $d(x)$ **is the difference between the target and observed values at frequency of fitness case** i
- $W(y,x)$ **is the weighting at frequency** x

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:** $F_{ccs\text{-}rpb\text{-}initial}$ **= {`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:** $T_{ccs\text{-}rpb\text{-}initial}$ **= {`END`}.** **For arithmetic-performing subtrees:** $T_{aps} = \{\leftarrow_{smaller\text{-}reals}\}$**.**

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)

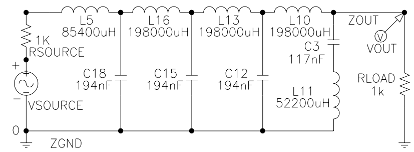


- **This genetically evolved circuit infringes on U. S. patent 1,227,113 issued to George Campbell of American Telephone and Telegraph in 1917 (claim 2):**

> **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."**

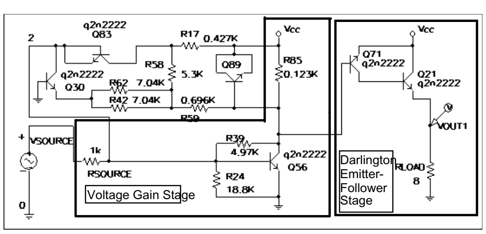
EVOLVED ZOBEL FILTER

- **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` and `L11`)**
- **Cascade of three symmetric T-sections**



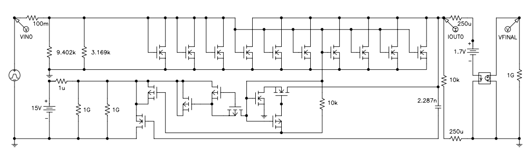
GENETICALLY EVOLVED 10 DB AMPLIFIER FROM GENERATION 45

SHOWING THE VOLTAGE GAIN STAGE AND DARLINGTON EMITTER FOLLOWER SECTION



POST-2000 PATENTED INVENTIONS

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

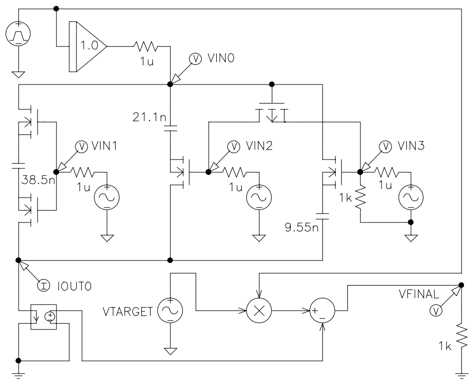


57

POST-2000 PATENTED INVENTIONS

REGISTER-CONTROLLED CAPACITOR CIRCUIT

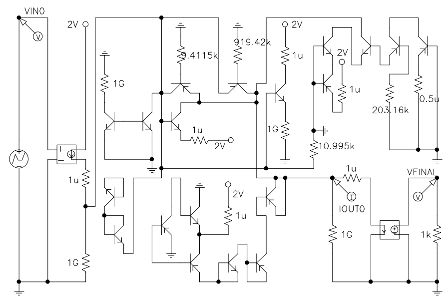
SMALLEST COMPLIANT FROM GENERATION 98



58

POST-2000 PATENTED INVENTIONS

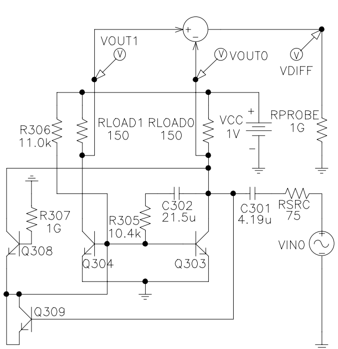
LOW-VOLTAGE CUBIC SIGNAL GENERATION CIRCUIT BEST-OF-RUN FROM GENERATION 182



59

POST-2000 PATENTED INVENTIONS

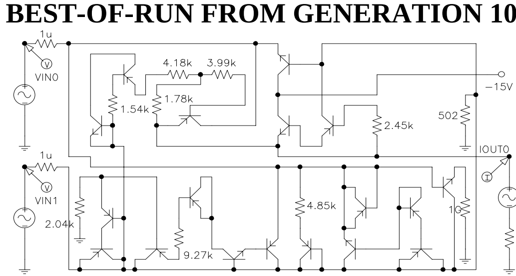
LOW-VOLTAGE BALUN CIRCUIT BEST EVOLVED FROM GENERATION 84



60

POST-2000 PATENTED INVENTIONS

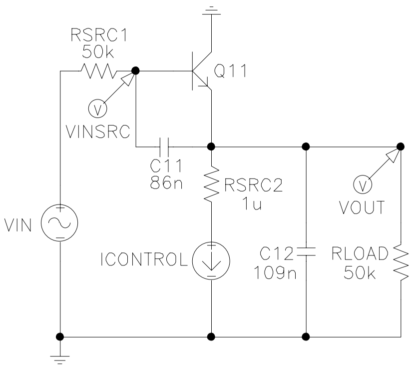
VOLTAGE-CURRENT-CONVERSION CIRCUIT BEST-OF-RUN FROM GENERATION 109



61

POST-2000 PATENTED INVENTIONS

TUNABLE INTEGRATED ACTIVE FILTER — GENERATION 50



62

21 PREVIOUSLY PATENTED INVENTIONS REINVENTED BY GP

	Invention	Date	Inventor	Place	Patent
1	Darlington emitter-follower section	1953	Sidney Darlington	Bell Telephone Laboratories	2,663,806
2	Ladder filter	1917	George Campbell	American Telephone and Telegraph	1,227,113
3	Crossover filter	1925	Otto Julius Zobel	American Telephone and Telegraph	1,538,964
4	"*M*-derived half section" filter	1925	Otto Julius Zobel	American Telephone and Telegraph	1,538,964
5	Cauer (elliptic) topology for filters	1934–1936	Wilhelm Cauer	University of Gottingen	1,958,742, 1,989,545
6	Sorting network	1962	Daniel G. O'Connor and Raymond J. Nelson	General Precision, Inc.	3,029,413
7	Computation al circuits	See text	See text	See text	See text
8	Electronic thermometer	See text	See text	See text	See text
9	Voltage reference circuit	See text	See text	See text	See text
10	60 dB and 96 dB amplifiers	See text	See text	See text	See text
11	Second-derivative controller	1942	Harry Jones	Brown Instrument Company	2,282,726
12	Philbrick circuit	1956	George Philbrick	George A. Philbrick Researches	2,730,679
13	NAND circuit	1971	David H. Chung and Bill H.	Texas Instruments Incorporated	3,560,760

63

			Terrell		
14	PID (proportional, integrative, and derivative) controller	1939	Albert Callender and Allan Stevenson	Imperial Chemical Limited	2,175,985
15	Negative feedback	1937	Harold S. Black	American Telephone and Telegraph	2,102,670, 2,102,671
16	Low-voltage balun circuit	2001	Sang Gug Lee	Information and Communications University	6,265,908
17	Mixed analog-digital variable capacitor circuit	2000	Turgut Sefket Aytur	Lucent Technologies Inc.	6,013,958
18	High-current load circuit	2001	Timothy Daun-Lindberg and Michael Miller	International Business Machines Corporation	6,211,726
19	Voltage-current conversion circuit	2000	Akira Ikeuchi and Naoshi Tokuda	Mitsumi Electric Co., Ltd.	6,166,529
20	Cubic function generator	2000	Stefano Cipriani and Anthony A. Takeshian	Conexant Systems, Inc.	6,160,427
21	Tunable integrated active filter	2001	Robert Irvine and Bernd Kolb	Infineon Technologies AG	6,225,859

64

2 PATENTED INVENTIONS CREATED BY GENETIC PROGRAMMING

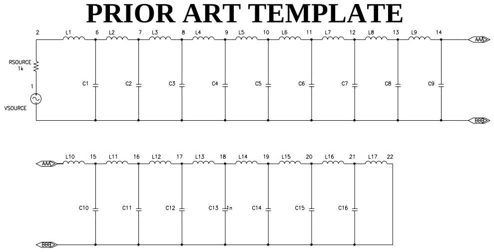
Keane, Martin A., Koza, John R., and Streeter, Matthew J. 2005. *Apparatus for Improved General-Purpose PID and Non-PID Controllers*. U. S. Patent 6,847,851. Filed July 12, 2002. Issued January 25, 2005.

NOVELTY-DRIVEN EVOLUTION

EXAMPLE OF LOWPASS FILTER

- **Two factors in fitness measure**
  - **Circuit's behavior in the frequency domain**
  - **Largest number of nodes and edges (circuit components) of a subgraph of the given circuit that is isomorphic to a subgraph of a template representing the prior art. Graph isomorphism algorithm with the cost function being based on the number of shared nodes and edges (instead of just the number of nodes).**

PRIOR ART TEMPLATE

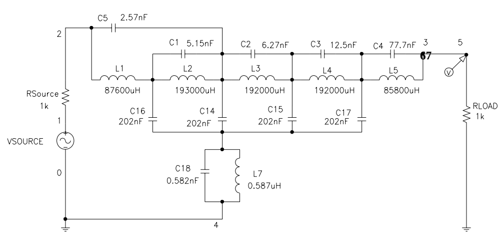


NOVELTY-DRIVEN EVOLUTION — CONTINUED

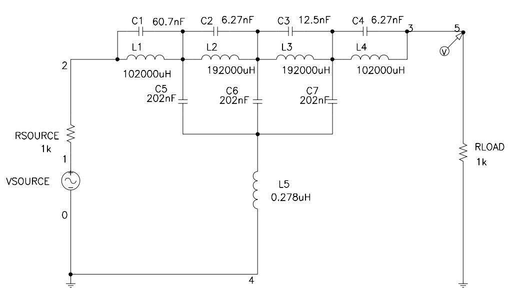
- **For circuits not scoring the maximum number (101) of hits, the fitness of a circuit is the product of the two factors.**
- **For circuits scoring 101 hits (100%-compliant individuals), fitness is the number of shared nodes and edges divided by 10,000.**

FITNESS OF EIGHT 100%-COMPLIANT CIRCUITS

Solution	Frequency factor	Isomorphism factor	Fitness
1	0.051039	7	0.357273
2	0.117093	7	0.819651
3	0.103064	7	0.721448
4	0.161101	7	1.127707
5	0.044382	13	0.044382
6	0.133877	7	0.937139
7	0.059993	5	0.299965
8	0.062345	11	0.685795



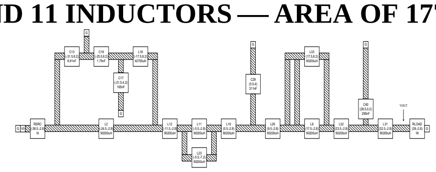
SOLUTION NO. 1



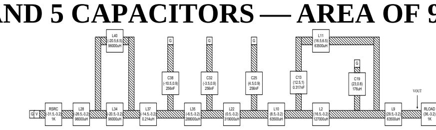
SOLUTION NO. 5

LAYOUT — LOWPASS FILTER 100%-COMPLIANT CIRCUITS

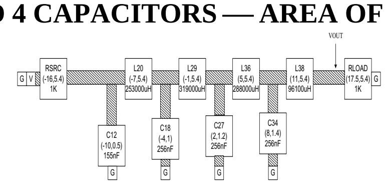
GENERATION 25 WITH 5 CAPACITORS AND 11 INDUCTORS — AREA OF 1775.2



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

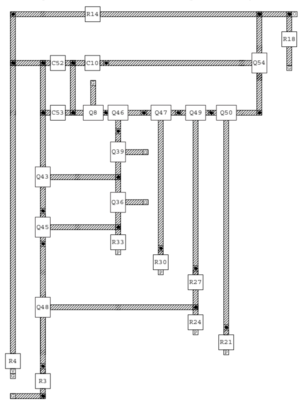


LAYOUT — 60 DB AMPLIFIER (USING TRANSISTORS)

COMPARISON

Gen	Components	Area	Four penalties	Fitness
65	27	8,234	33.034348	33.042583
101	19	4,751	0.061965	0.004751

BEST-OF-RUN CIRCUIT FROM GENERATION 101

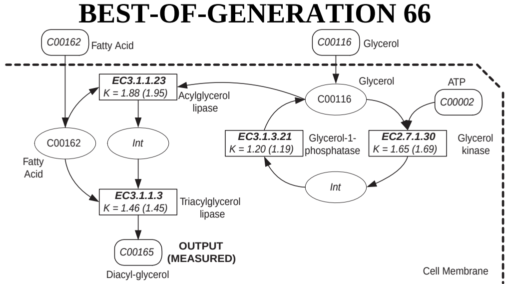


AUTOMATIC PARALLELIZATION OF SERIAL PROGRAMS USING GP

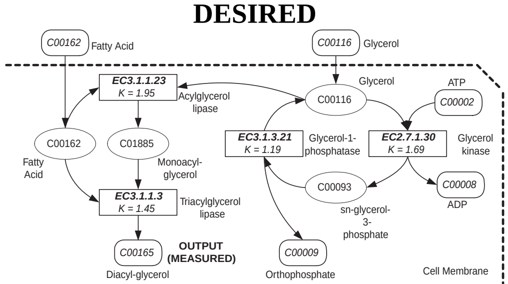
- **Ryan, Conor. 1999. *Automatic Re-engineering of Software Using Genetic Programming*. Amsterdam: Kluwer Academic Publishers.**
- **Start with working serial computer program (embryo)**
- **GP program tree contains validity-preserving functions that modify the current program. That is, the functions 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**

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

BEST-OF-GENERATION 66



DESIRED



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

73

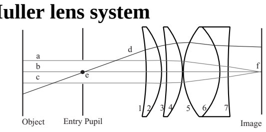
CROSS-DOMAIN FEATURES

- **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**

74

NATIVE REPRESENTATIONS ARE SUFFICIENT WHEN WORKING WITH GENETIC PROGRAMMING

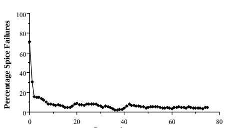
Tackaberry-Muller lens system



Lens file for Tackaberry-Muller system

Surface	Distance	Radius	Material	Aperture
Object	10^{10}	flat	air	
Entry pupil	0.88	flat	air	0.18
1	0.21900	–3.5236	BK7	0.62
2	0.07280	–1.0527	air	0.62
3	0.22500	–4.4072	BK7	0.62
4	0.01360	–1.0704	air	0.62
5	0.52100	1.02491	BK7	0.62
6	0.11800	–0.9349	SF61	0.62
7	0.47485	7.94281	air	0.62
Image		flat		

75

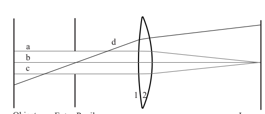
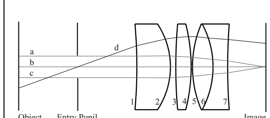
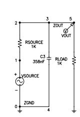
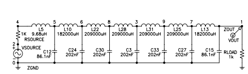
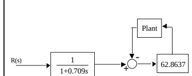
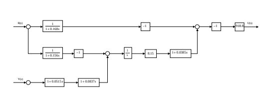
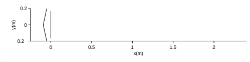
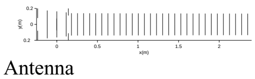


GENETIC PROGRAMMING BREEDS SIMULATABILITY

Unsimulatable individuals

76

GP STARTS SMALL

Best-of-generation 0	Best-of-run
 Optical lens system	 Optical lens system
 Lowpass filter	 Lowpass filter
 Controller	 Controller
 Antenna	 Antenna

77

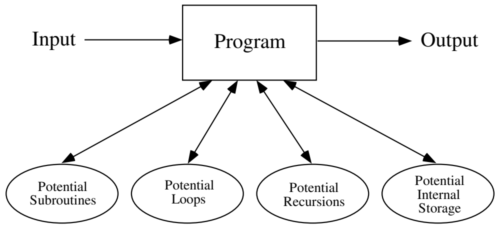
GENETIC PROGRAMMING ENGINEERS AROUND EXISTING PATENTS AND CREATES NOVEL DESIGNS MORE FREQUENTLY THAN IT CREATES INFRINGING SOLUTIONS

78

GENETIC PROGRAMMING FREQUENTLY EXPLOITS A SIMULATOR'S BUILT-IN ASSUMPTION OF REASONABLENESS

79

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**

80

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

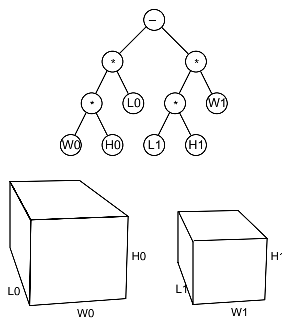
Fitness case	L_0	W_0	H_0	L_1	W_1	H_1	Dependent variable *D*
1	3	4	7	2	5	3	**54**
2	7	10	9	10	3	1	**600**
3	10	9	4	8	1	6	**312**
4	3	9	5	1	6	4	**111**
5	4	3	2	7	6	1	**–18**
6	3	3	1	9	5	4	**–171**
7	5	9	9	1	7	6	**363**
8	1	2	9	3	9	2	**–36**
9	2	6	8	2	6	10	**–24**
10	8	1	10	7	5	1	**45**

AUTOMATICALLY DEFINED FUNCTIONS (ADFS, SUBROUTINES)

SOLUTION WITHOUT ADFS

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

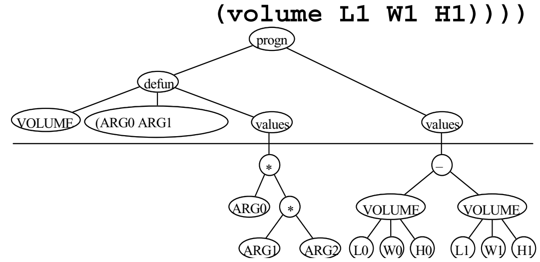
```
D = W0*L0*H0 – W1*L1*H1
```



AUTOMATICALLY DEFINED FUNCTIONS (ADFS, SUBROUTINES)

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

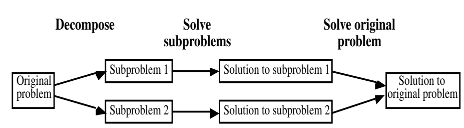
Fitness case	L_0	W_0	H_0	L_1	W_1	H_1	V_0	V_1	D
1	3	4	7	2	5	3	**84**	**30**	**54**
2	7	10	9	10	3	1	**630**	**30**	**600**
3	10	9	4	8	1	6	**360**	**48**	**312**
4	3	9	5	1	6	4	**135**	**24**	**111**
5	4	3	2	7	6	1	**24**	**42**	**–18**
6	3	3	1	9	5	4	**9**	**180**	**–171**
7	5	9	9	1	7	6	**405**	**42**	**363**
8	1	2	9	3	9	2	**18**	**54**	**–36**
9	2	6	8	2	6	10	**96**	**120**	**–24**
10	8	1	10	7	5	1	**80**	**35**	**45**

- **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

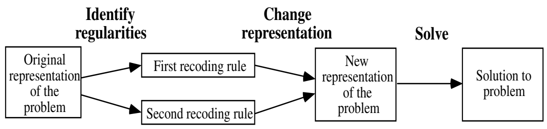


- **Decompose a problem into subproblems**
- **Solve the subproblems**
- **Assemble the solutions of the subproblems into a solution for the overall problem**

85

AUTOMATICALLY DEFINED FUNCTIONS (ADFS, SUBROUTINES)

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



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

86

AUTOMATICALLY DEFINED FUNCTIONS (ADFS, SUBROUTINES)

- **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.**

87

AUTOMATICALLY DEFINED FUNCTIONS (ADFS, SUBROUTINES)

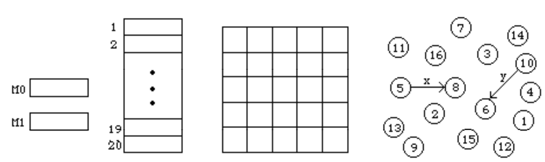
8 MAIN POINTS FROM BOOK *GENETIC PROGRAMMING II: AUTOMATIC DISCOVERY OF REUSABLE PROGRAMS* (KOZA 1994)

- **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.**

88

REUSE

MEMORY AND STORAGE



(A) (B) (C) (D)

- **(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**

- **After cross-validation**
  - **out-of-sample correlation of 0.968**
  - **out-of-sample error rate 1.6%**

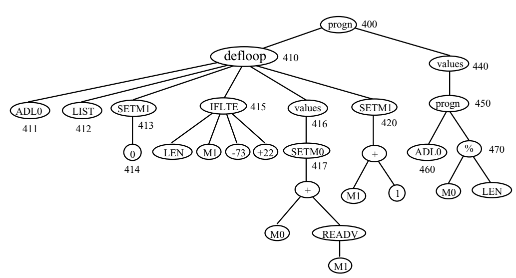
REUSE—TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM

```
(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 (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))))))
```

- **GP created the body of 3 subroutines (ADFs), 1 iteration-performing branch, and 1 result-producing branch (RPB) were created by genetic programming**

REUSE

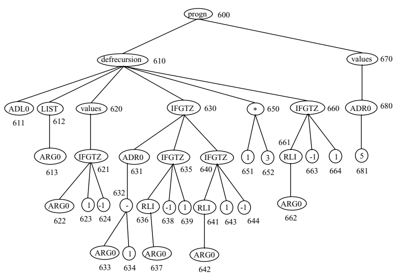
EXAMPLE OF A PROGRAM WITH A FOUR-BRANCH AUTOMATICALLY DEFINED LOOP (`ADL0`) AND A RESULT-PRODUCING BRANCH



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**



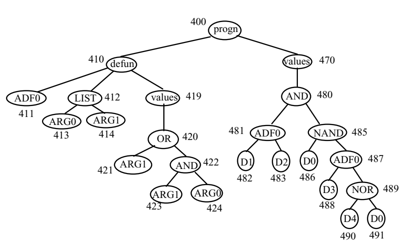
ARCHITECTURE-ALTERING OPERATIONS

PROTEIN ALIGNMENT OF "A" AND "B" PROTEINS

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

First.protein  ESSEYEIEVI KHQKEKTEKK EKEKKTHVET KKEVKKKEKK QIPCSEK 100
Second.protein DSSEYEIEVI KHQKEKTEKK EKEKKAHVEI KKKIKNKEKK FVPCSEI 100

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

First.protein  KTVKVVKPPK EKPPKKLRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTK 199
Second.protein KVVKVIKPPK EKPPKKPRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTK 196

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

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

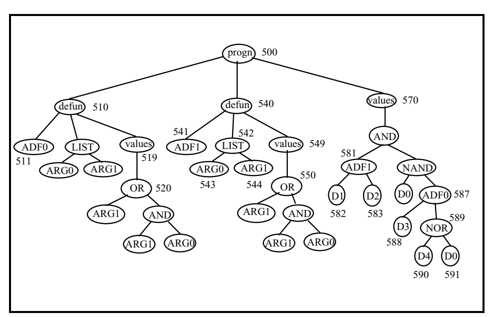

ARCHITECTURE-ALTERING OPERATIONS

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



ARCHITECTURE-ALTERING OPERATIONS

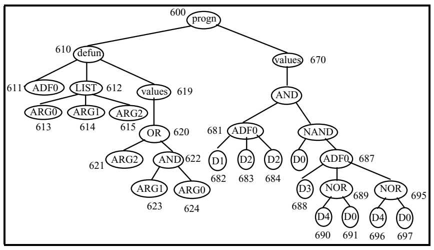
PROGRAM WITH ARGUMENT MAP OF {2, 2} CREATED USING THE OPERATION OF BRANCH DUPLICATION



97

ARCHITECTURE-ALTERING OPERATIONS

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



98

ARCHITECTURE-ALTERING OPERATIONS

SPECIALIZATION – REFINEMENT – CASE SPLITTING

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

GENERALIZATION

- **Branch deletion**
- **Argument deletion**

99

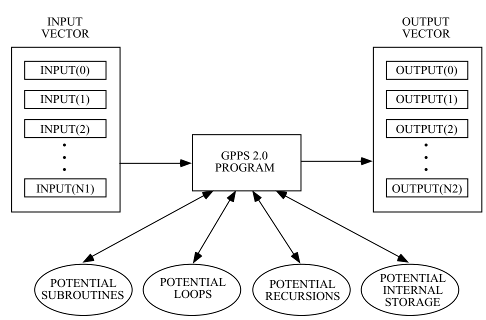
16 ATTRIBUTES OF A SYSTEM FOR AUTOMATICALLY CREATING COMPUTER PROGRAMS

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

100

ARCHITECTURE-ALTERING OPERATIONS

GENETIC PROGRAMMING PROBLEM SOLVER (GPPS) —VERSION 2.0



101

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

- **Nordin, Peter. 1997. *Evolutionary Program Induction of Binary Machine Code and its Application*. Munster, Germany: Krehl Verlag.**
- **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.**

102

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

103

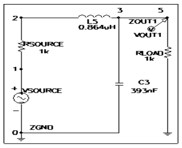
STRONG INDICATIONS FOR USING GENETIC ALGORITHM (GA) OR EVOLUTION STRATEGIES (ES)

- **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**

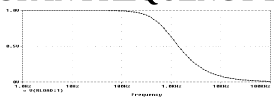
104

REUSE
LOWPASS FILTER USING ADFS

GENERATION 0 – ONE-RUNG LADDER

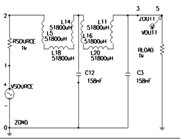


BEHAVIOR IN FREQUENCY DOMAIN

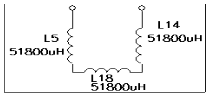


105

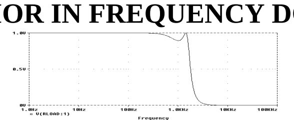
REUSE
LOWPASS FILTER USING ADFS
GENERATION 9 - TWO-RUNG LADDER



TWICE-CALLED TWO-PORTED ADF0

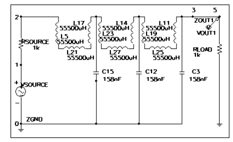


BEHAVIOR IN FREQUENCY DOMAIN

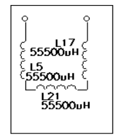


106

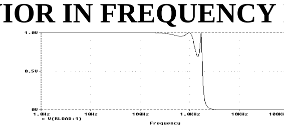
REUSE
LOWPASS FILTER USING ADFS
GEN 16 – THREE-RUNG LADDER



THRICE-CALLED TWO-PORTED ADF0

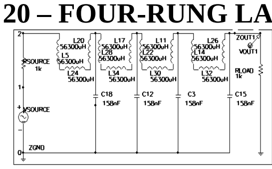


BEHAVIOR IN FREQUENCY DOMAIN

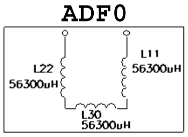


107

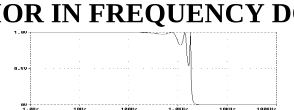
REUSE
LOWPASS FILTER USING ADFS
GEN 20 – FOUR-RUNG LADDER



QUADRUPLY-CALLED TWO-PORTED ADF0

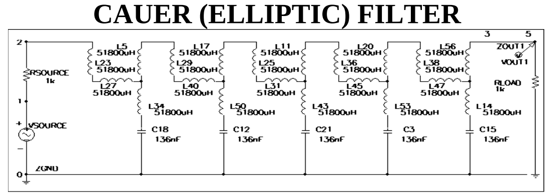


BEHAVIOR IN FREQUENCY DOMAIN

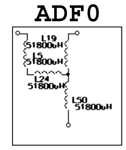


108

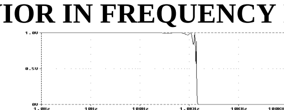
REUSE
LOWPASS FILTER USING ADFS
GENERATION 31 — TOPOLOGY OF CAUER (ELLIPTIC) FILTER



QUINTUPLY-CALLED THREE-PORTED ADF0



BEHAVIOR IN FREQUENCY DOMAIN



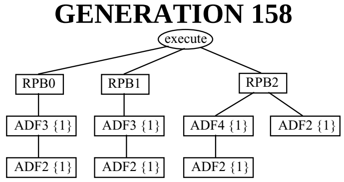
PASSING A PARAMETER TO A SUBSTRUCTURE

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

$T_{ccs\text{-}adf\text{-}potential}$ = {ARG0}

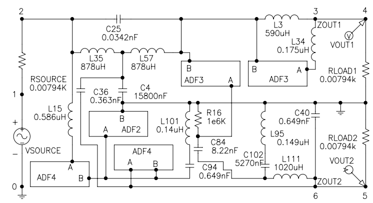
EMERGENCE OF A PARAMETERIZED ARGUMENT IN A CIRCUIT SUBSTRUCTURE

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



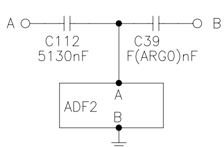
PASSING A PARAMETER TO A SUBSTRUCTURE

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



THE FIRST RESULT-PRODUCING BRANCH, RPB0, CALLING ADF3

```
(PARALLEL0 (L (+ (- 1.883196E-01 (- -9.095883E-02 5.724576E-01)) (- 9.737455E-01 -9.452780E-01)) (FLIP END)) (SERIES (C (+ (+ -6.668774E-01 -8.770285E-01) 4.587758E-02) (NOP END)) (SERIES END END (PARALLEL1 END END END END)) (FLIP (SAFE_CUT))) (PAIR_CONNECT_0 END END END) (PAIR_CONNECT_0 (L (+ -7.220122E-01 4.896697E-01) END) (L (- -7.195599E-01 3.651142E-02) (SERIES (C (+ -5.111248E-01 (- (- -6.137950E-01 -5.111248E-01) (- 1.883196E-01 (- -9.095883E-02 5.724576E-01)))) END) (SERIES END END (adf3 6.196514E-01)) (NOP END))) (NOP END)))
```

AUTOMATICALLY DEFINED FUNCTION ADF3

```
(C (+ (- (+ (+ (+ 5.630820E-01 (- 9.737455E-01 -9.452780E-01)) (+ ARG0 6.953752E-02)) (- (- 5.627716E-02 (+ 2.273517E-01 (+ 1.883196E-01 (+ 9.346950E-02 (+ -7.220122E-01 (+ 2.710414E-02 1.397491E-02)))))) (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 -8.554120E-01)) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02))))) (+ (+ 1.883196E-01 (+ (+ (+ (+ 9.346950E-02 (+ -7.220122E-01 (+ 2.710414E-02 1.397491E-02))) (- 4.587758E-02 -2.340137E-01)) 3.226026E-01) (+ -7.220122E-01 (- -9.131658E-01 6.595502E-01)))) 3.660116E-01)) 9.496355E-01) (THREE_GROUND_0 (C (+ (- (+ (+ (+ 5.630820E-01 (- 9.737455E-01 -9.452780E-01)) (+ (- (- -7.195599E-01 3.651142E-02) -9.761651E-01) (- (+ (- (- -7.195599E-01 3.651142E-02) -9.761651E-01) 6.953752E-02) 3.651142E-02))) (- (- 5.627716E-02 (- 1.883196E-01 (- -9.095883E-02 5.724576E-01))) (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 (+ ARG0 6.953752E-02))) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02))))) (+ (+ 1.883196E-01 -7.195599E-01) 3.660116E-01)) 9.496355E-01) (NOP (FLIP (PAIR_CONNECT_0 END END END)))) (FLIP (SERIES (FLIP (FLIP (FLIP END))) (C (- (+ 6.238477E-01 6.196514E-01) (+ (+ (- (- 4.037348E-01 4.343444E-01) (+ -7.788187E-01 (+ (+ (- -8.786904E-01 1.397491E-02) (- -6.137950E-01 (- (+ (- 2.710414E-02 -2.807583E-01) (+ -6.137950E-01 -8.554120E-01)) (- -8.770285E-01 (- -4.049602E-01 -2.192044E-02))))) (+ (+ 7.215142E-03 1.883196E-01) (+ 7.733750E-01 4.343444E-01))))) (- (- -9.389297E-01 5.630820E-01) (+ -5.840433E-02 3.568947E-01))) -8.554120E-01)) (NOP END)) END)) (FLIP (adf2 9.737455E-01))))
```

113

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`.

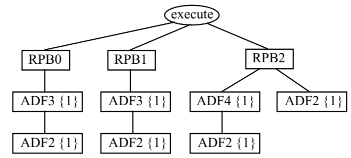
• The combined effect of `ADF3` is to insert the following three components:

- 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.

114

EMERGENCE OF A PARAMETERIZED ARGUMENT IN A CIRCUIT SUBSTRUCTURE

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



115

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	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 <u>ONE INSTANCE</u> of the quadratic equation problem.

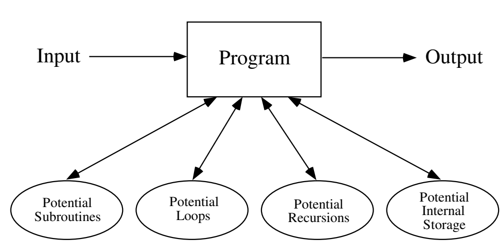
116

SOLVING A QUADRATIC EQUATION USING GENETIC PROGRAMMING (GP)

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

$$ax^2 + bx + c = 0$$

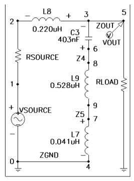
by searching the space of computer programs for a program that takes ***a***, ***b***, and ***c*** as inputs



• The result is a solution to <u>ALL INSTANCES</u> of the quadratic equation problem

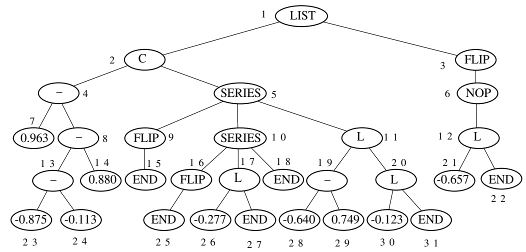
GENERAL APPEARANCE OF ONE POSSIBLE CHROMOSOME ENCODING USED TO SOLVE ONE INSTANCE OF A CIRCUIT PROBLEM USING THE GENETIC ALGORITHM (GA) OPERATING ON FIXED-LENGTH CHARACTER STRINGS

EXAMPLE CIRCUIT



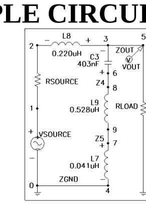
1st Component				2nd Component				3rd Component				4th Component			
L	.220	2	3	C	403.	3	6	L	.528	6	9	L	.041	9	0

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



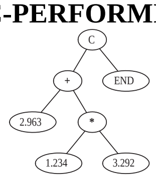

```
(LIST (C (- 0.963 (- (- -0.875 -0.113) 0.880)) (series (flip
end) (series (flip end) (L -0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip (nop (L -0.657 end)))))
```

EXAMPLE CIRCUIT (GEN 0)

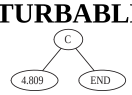


VALUE-SETTING SUBTREES—3 WAYS

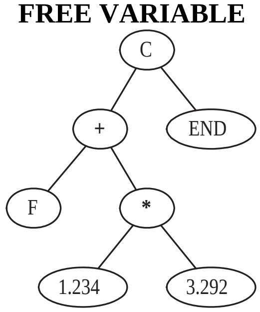
ARITHMETIC-PERFORMING SUBTREE



SINGLE PERTURBABLE CONSTANT



FREE VARIABLE

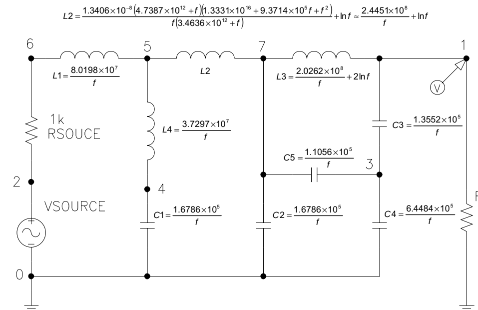


PARAMETERIZED TOPOLOGY FOR "GENERALIZED" LOWPASS FILTER

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

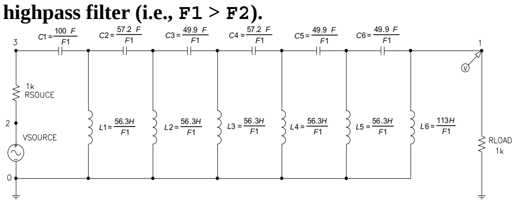
$$L2 = \frac{1.3406\times10^{-8}(4.7387\times10^{12}+f)(1.3331\times10^{16}+9.3714\times10^{5}f+f^{2})}{f(3.4636\times10^{12}+f)} + \ln f \approx \frac{2.4451\times10^{8}}{f} + \ln f$$



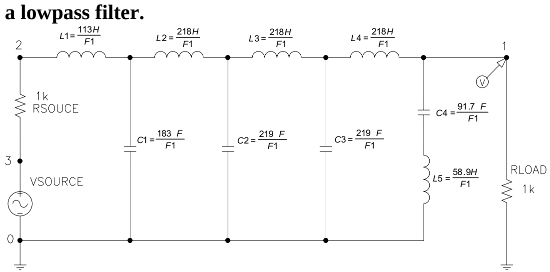
PARAMETERIZED TOPOLOGY USING CONDITIONAL DEVELOPMENTAL OPERATORS (GENETIC SWITCH)

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT

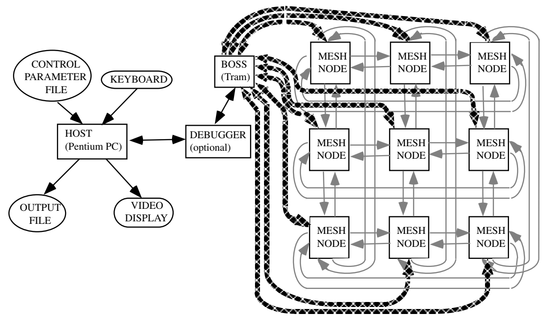
- **Best-of-run circuit from generation 93 when inputs call for a highpass filter (i.e., `F1` > `F2`).**



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



PARALLELIZATION BY SUBPOPULATIONS ("ISLAND" OR "DEME" MODEL OR "DISTRIBUTED GENETIC ALGORITHM")



- **Like Hormel, Get Everything Out of the Pig, Including the Oink**
- **Keep on Trucking**
- **It Takes a Licking and Keeps on Ticking**
- **The Whole is Greater than the Sum of the Parts**

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

System	Period of usage	Petacycles (10^{15} cycles) per day for entire system	Speed-up over previous system	Speed-up over first system in this table	Human-competitive results
Serial Texas Instruments LISP machine	1987–1994	0.00216	1 (base)	1 (base)	0
64-node Transtech transputer parallel machine	1994–1997	0.02	9	9	2
64-node Parsytec parallel machine	1995–2000	0.44	22	204	12
70-node Alpha parallel machine	1999–2001	3.2	7.3	1,481	2
1,000-node Pentium II parallel machine	2000–2002	30.0	9.4	13,900	12

PROGRESSION OF RESULTS

System	Period	Speed-up	Qualitative nature of the results produced by genetic programming
Serial LISP machine	1987–1994	1 (base)	• Toy problems of the 1980s and early 1990s from the fields of artificial intelligence and machine learning
64-node Transtech 8-biy transputer	1994–1997	9	•Two human-competitive results involving one-dimensional discrete data (not patent-related)
64-node Parsytec parallel machine	1995–2000	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 20^{th}-century patented inventions
70-node Alpha parallel machine	1999–2001	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)
1,000-node Pentium II parallel machine	2000–2002	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 21^{st}-century patented inventions
Long (4-week) runs of 1,000-node Pentium II parallel machine	2002	9.3	• Generation of two patentable new inventions

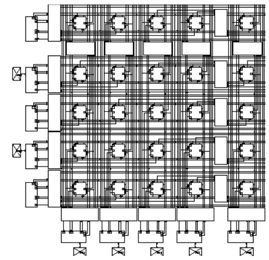
PROGRESSION OF QUALITATIVELY MORE SUBSTANTIAL RESULTS PRODUCED BY GENETIC PROGRAMMING IN RELATION TO FIVE ORDER-OF-MAGNITUDE INCREASES IN COMPUTATIONAL POWER

- **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



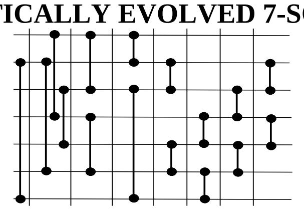
EVOLVABLE HARDWARE

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



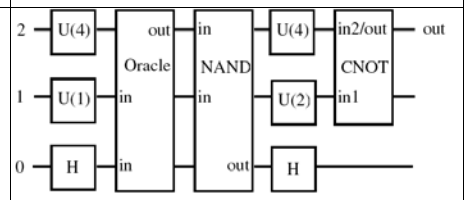
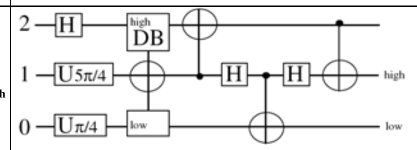
FUNDAMENTAL DIFFERENCES BETWEEN GP AND OTHER APPROACHES TO AI AND ML

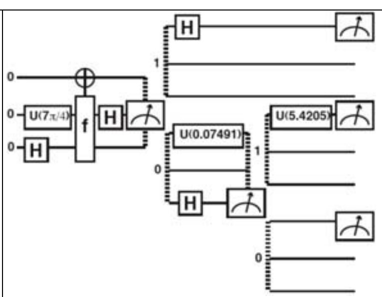
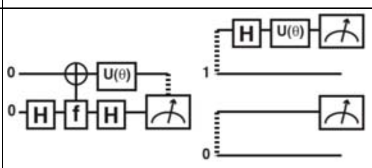
(1) Representation: Genetic programming overtly conducts it search for a solution to the given problem in program space.
(2) Role of point-to-point transformations in the search: Genetic programming does not conduct its search by transforming a single point in the search space into another single point, 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.

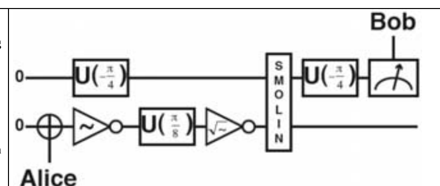
EIGHT CRITERIA FOR HUMAN-COMPETITIVENESS

	Criterion
A	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.
B	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.
C	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.
D	The result is publishable in its own right as a new scientific result—independent of the fact that the result was mechanically created.
E	The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
F	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.
G	The result solves a problem of indisputable difficulty in its field.
H	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).

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

Claimed instance	Picture
Creation of a better-than-classical quantum algorithm for the Deutsch-Jozsa "early promise" problem Spector, Barnum, and Bernstein 1998	
Creation of a better-than-classical quantum algorithm for Grover's database search problem Spector, Barnum, and Bernstein 1999	

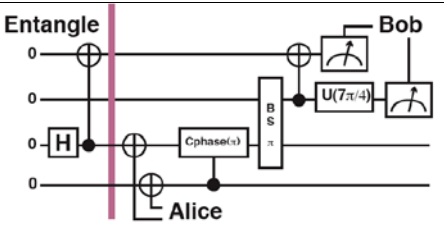
Creation of a quantum algorithm for the depth-two AND/OR query problem that is better than any previously published result Spector, Barnum, Bernstein, and Swamy 1999; Barnum, Bernstein, and Spector 2000	
Creation of a quantum algorithm for the depth-one OR query problem that is better than any previously 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 $$\text{Smolin} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

133

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 to bs-gate.jpg

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

134

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

Luke 1998

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

135

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

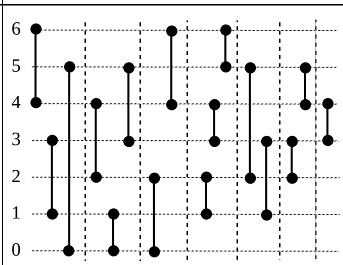
Sections 18.8 and 18.10 of *Genetic Programming II* and sections 16.5 and 17.2 of *Genetic Programming III*

"0-2-4 rule" from section 16.5 of *Genetic Programming III*

Residue	Increment
A, F, I, L, M, or V	0
C, D, G, H, K, N, P, Q, R, S, T, W, or Y	+2
E	

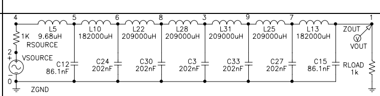
Creation of a sorting network for seven items using only 16 steps

Sections 21.4.4, 23.6, and 57.8.1 of *Genetic Programming III*



Rediscovery of the Campbell ladder topology for lowpass and highpass filters

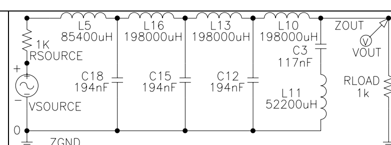
Section 25.15.1 of *Genetic Programming III*



136

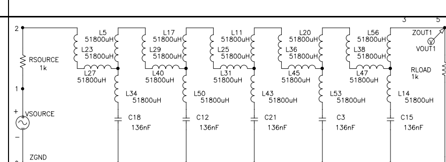
Rediscovery of the Zobel "*M*-derived half section" and "constant *K*" filter sections

Section 25.15.2 of *Genetic Programming III*



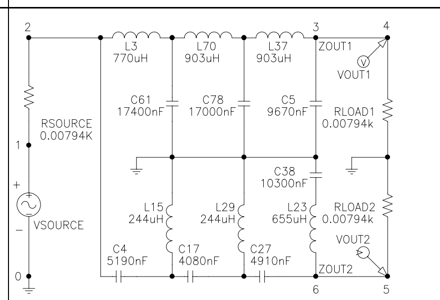
Rediscovery of the Cauer (elliptic) topology for filters

Section 27.3.7 of *Genetic Programming III*



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

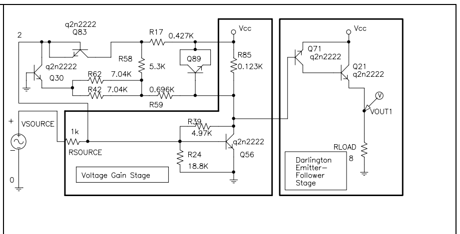
Section 32.3 of *Genetic Programming III*



137

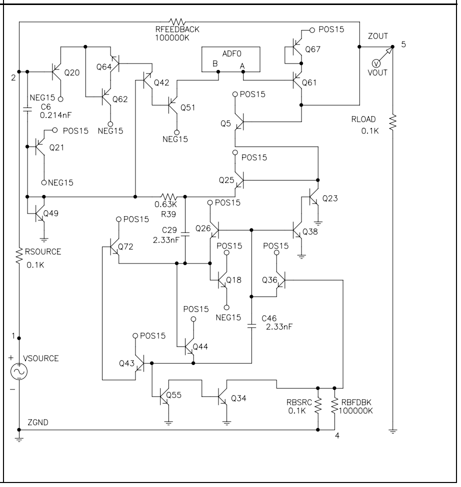
Rediscovery of a recognizable voltage gain stage and a Darlington emitter-follower section of an amplifier and other circuits

Section 42.3 of *Genetic Programming III*



Synthesis of 60 and 96 decibel amplifiers

Section 45.3 of *Genetic Programming III*

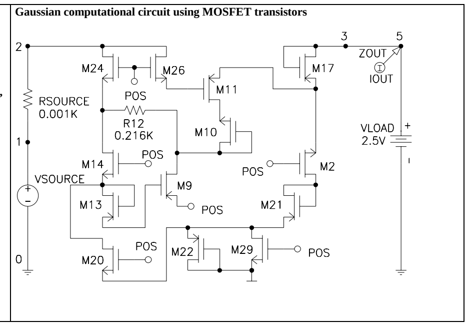


138

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

Section 47.5.3 of *Genetic Programming III*

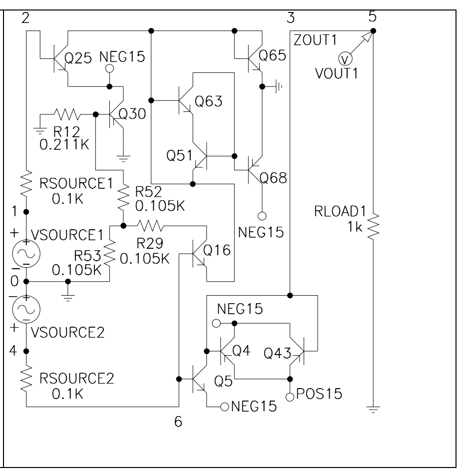
Gaussian computational circuit using MOSFET transistors



139

Synthesis of a real-time analog circuit for time-optimal control of a robot

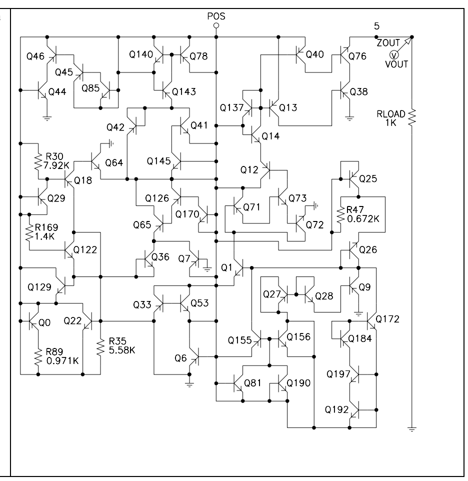
Section 48.3 of *Genetic Programming III*



140

Synthesis of an electronic thermometer

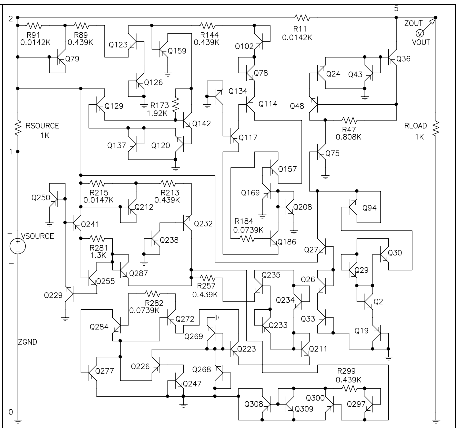
Section 49.3 of *Genetic Programming III*



141

Synthesis of a voltage reference circuit

Section 50.3 of *Genetic Programming III*



142

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 00000000 01011111 11111111 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 11111111 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

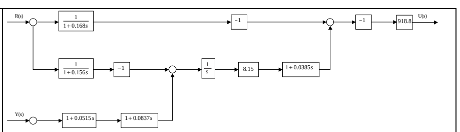
Section 59.8 of *Genetic Programming III*

`[IV]-[lim]-D-E-[AI]-D-[rnek]-[lim]-[lim]-[limeqdnrsk]`

143

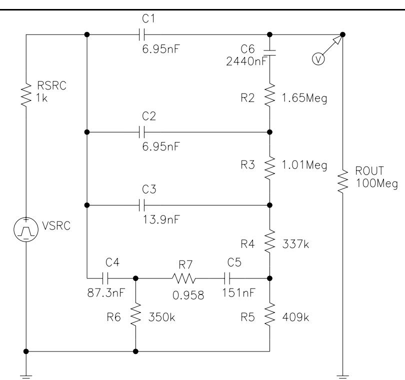
Synthesis of topology for a PID-D2 (proportional, integrative, derivative, and second derivative) controller

Section 3.7 of *Genetic Programming IV*



Synthesis of an analog circuit equivalent to Philbrick circuit

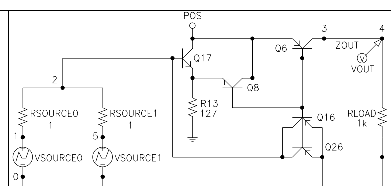
Section 4.3 of *Genetic Programming IV*



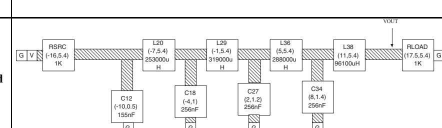
144

Synthesis of a NAND circuit

Section 4.4 of *Genetic Programming IV*

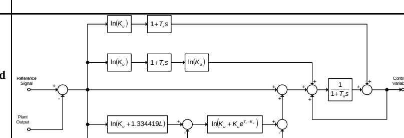


Simultaneous synthesis of topology, sizing, placement, and routing of analog electrical circuits



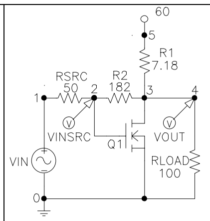
Synthesis of topology for a PID (proportional, integrative, and derivative) controller

Chapter 5 of *Genetic Programming IV*



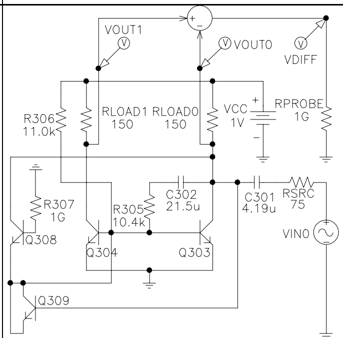
Rediscovery of negative feedback

Chapter 14 of *Genetic Programming IV*



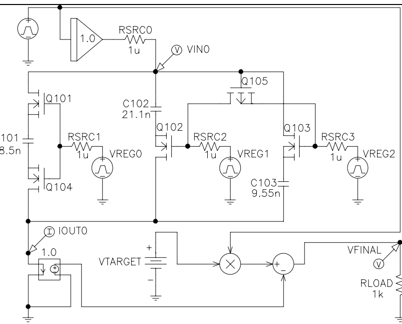
Synthesis of a low-voltage balun circuit

Section 15.4.1 of *Genetic Programming IV*



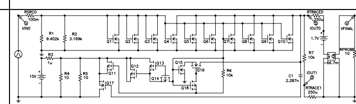
Synthesis of a mixed analog-digital variable capacitor circuit

Section 15.4.2 of *Genetic Programming IV*



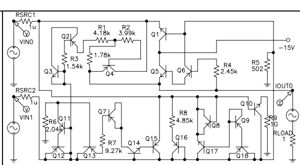
Synthesis of a high-current load circuit

Section 15.4.3 of *Genetic Programming IV*



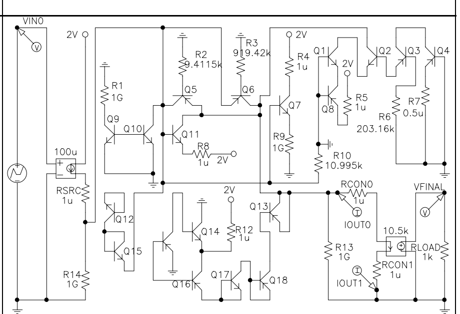
Synthesis of a voltage-current conversion circuit

Section 15.4.4 of *Genetic Programming IV*



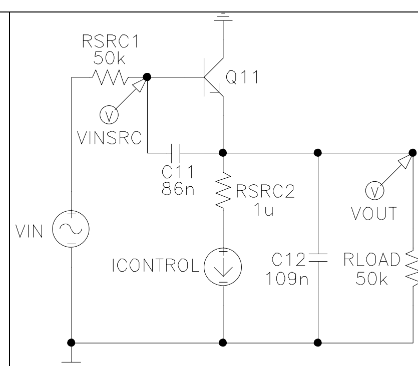
Synthesis of a cubic function generator

Section 15.4.5 of *Genetic Programming IV*



Synthesis of a tunable integrated active filter

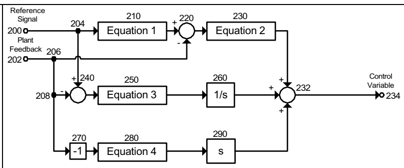
Section 15.4.6 of *Genetic Programming IV*



149

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 $K_{p\text{-}final}$, $K_{i\text{-}final}$, $K_{d\text{-}final}$, and b_{final}:

$$K_{p\text{-}final} = 0.72 * K_u * e^{\frac{-1.6}{K_u}+\frac{1.2}{K_u^2}} - .0012340 * T_u - 6.1173 * 10^{-6}$$

$$K_{i\text{-}final} = \frac{0.72 * K_u * e^{\frac{-1.6}{K_u}+\frac{1.2}{K_u^2}}}{0.59 * T_u * e^{\frac{-1.3}{K_u}+\frac{0.38}{K_u^2}}} - .068525 * \frac{K_u}{T_u}$$

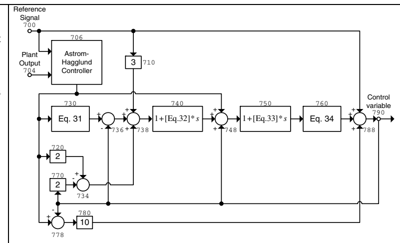
$$K_{d\text{-}final} = 0.108 * K_u * T_u * e^{\frac{-1.6}{K_u}+\frac{1.2}{K_u^2}} * e^{\frac{-1.4}{K_u}+\frac{0.56}{K_u^2}} - 0.0026640\left(e^{T_u}\right)^{\log\left(1.6342^{\log K_u}\right)}$$

$$b_{final} = 0.25 * e^{\frac{0.56}{K_u}+\frac{-0.12}{K_u^2}} + \frac{K_u}{e^{K_u}}$$

150

Creation of three non-PID controllers that outperform a PID controller that uses the Ziegler-Nichols or Åström-Hägglund tuning rules

Chapter 13 of *Genetic Programming IV*



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

$$\left|\log\left|T_r - T_u + \log\left|\frac{\log\left(|L|^L\right)}{T_u+1}\right|\right|\right| \qquad [31]$$

$$\left|\log\left|T_r + 1\right|\right| \qquad [34]$$

$$NLM\left(\log|L| - \left(\text{abs}(L)^L\right)^2 T_u^3 \left(T_u+1\right) T_r e^L - 2T_u e^L\right) \qquad [32]$$

$$NLM\left(\log|L| - 2T_u e^L \left(2K_u \left(\log\left|K_u e^L\right| - \log|L|\right) T_u + K_u e^L\right)\right) \qquad [33]$$

Antenna that satisfied NASA specs and that will be launched into space in 2004

Lohn et al. 2003

151

EVOLUTIONARY SYNTHESIS OF KINEMATIC MECHANISMS (LIPSON 2004)

152

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).

TURING'S THREE APPROACHES TO MACHINE INTELLIGENCE — CONTINUED

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"

17 AUTHORED BOOKS ON GP

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. 1998. *Genetic Programming - An Introduction*. San Francisco, CA: Morgan Kaufman Publishers and Heidelberg, Germany: dpunkt.verlag.

Babovic, Vladan. 1996. *Emergence, Evolution, Intelligence: Hydroinformatics*. Rotterdam, The Netherlands: Balkema Publishers.

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.

Jacob, Christian. 1997. *Principia Evolvica: Simulierte Evolution mit Mathematica*. Heidelberg, Germany: dpunkt.verlag. In German. English translation forthcoming in 2000 from Morgan Kaufman Publishers.

Jacob, Christian. 2001. *Illustrating Evolutionary Computation with Mathematica*. San Francisco: Morgan Kaufmann.

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

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

Koza, John R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: The MIT Press

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving.* San Francisco, CA: Morgan Kaufmann Publishers.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido. 2003. *Genetic Programming IV. Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

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

Langdon, William B. and Poli, Riccardo. 2002. *Foundations of Genetic Programming*. Berlin: Springer-Verlag.

Nordin, Peter. 1997. *Evolutionary Program Induction of Binary Machine Code and its Application*. Munster, Germany: Krehl Verlag.

O'Neill, Michael and Ryan, Conor. 2003. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Boston: Kluwer Academic Publishers.

Ryan, Conor. 1999. *Automatic Re-engineering of Software Using Genetic Programming*. Amsterdam: Kluwer Academic Publishers.

Spector, Lee. 2004. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Boston: Kluwer Academic Publishers.

Wong, Man Leung and Leung, Kwong Sak. 2000. *Data Mining Using Grammar Based Genetic Programming and Applications*. Amsterdam: Kluwer Academic Publishers.

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

Book	Main Points
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.

VARIOUS CONFERENCES

ASPGP

Cho, Sung-Bae, Nguyen, Hoai Xuan, and Shan, Yin (editors). 2003. *Proceedings of the First Asian-Pacific Workshop on Genetic Programming*. ISBN 0975172409. www.aspgp.org

GECCO

Beyer, H.-G.; O'Reilly, U.-M.; Arnold, D.V.; Banzhaf, W.; Blum, C.; Bonabeau, E.W.; Cantu-Paz, E.; Dasgupta, D.; Deb, K.; Foster, J.A.; de Jong, E.D.; Lipson, H.; Llora, X.; Mancoridis, S.; Pelikan, M.; Raidl, G.R.; Soule, T.; Tyrrell, A.; Watson, J.-P.; Zitzler, E. (editors). *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2005*. New York, NY: ACM Press.

EURO-GP

Keijzer, Maarten, Tettamanzi, Andrea, Collet, Pierre, van Hemert, Jano, Tomassini, Marco (editor). *Genetic Programming: 8th European Conference, EuroGP 2005, Lausanne, Switzerland, March 30-April 1, 2005, Proceedings*. Lecture Notes in Computer Science 3447. Heidelberg: Springer-Verlag.

GP Conference (Now part of GECCO)

Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, 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*. San Francisco, CA: Morgan Kaufmann.

GPTP

Yu, Gwoing, Worzel, William, and Riolo, Rick (editors). *Genetic Programming Theory and Practice III*. New York: Springer.

3 EDITED *ADVANCES IN GENETIC PROGRAMMING* BOOKS

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press.

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999. *Advances in Genetic Programming 3*. Cambridge, MA: The MIT Press.

4 VIDEOTAPES ON GP

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie.* Cambridge, MA: The MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave, Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann Publishers.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, Lanza, Guido, and Fletcher, David. 2003. *Genetic Programming IV Video: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

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/

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**