Pacific Symposium on Biocomputing 6:434-445 (2001)

REVERSE ENGINEERING OF METABOLIC PATHWAYS FROM OBSERVED DATA USING GENETIC PROGRAMMING

John R. Koza
Biomedical Informatics, Department of Medicine
Department of Electrical Engineering
Stanford University, Stanford, California, koza@stanford.edu
William Mydlowec
Genetic Programming Inc., Los Altos, California, myd@cs.stanford.edu
Guido Lanza
Genetic Programming Inc., Los Altos, California, guidissimo@hotmail.com
Jessen Yu
Genetic Programming Inc., Los Altos, California, jyu@cs.stanford.edu
Martin A. Keane
Econometrics Inc., Chicago, Illinois, makeane@ix.netcom.com

Recent work has demonstrated that genetic programming is capable of automatically creating complex networks (such as analog electrical circuits and controllers) whose behavior is modeled by linear and non-linear continuous-time differential equations and whose behavior matches prespecified output values. The concentrations of substances participating in networks of chemical reactions are also modeled by non-linear continuous-time differential equations. This paper demonstrates that it is possible to automatically create (reverse engineer) a network of chemical reactions from observed time-domain data. Genetic programming starts with observed time-domain concentrations of input substances and automatically creates both the topology of the network of chemical reactions and the rates of each reaction within the network such that the concentration of the final product of the automatically created network matches the observed time-domain data. Specifically, genetic programming automatically created metabolic pathways involved in the phospholipid cycle and the synthesis and degradation of ketone bodies.

1. Introduction

A living cell can be viewed as a dynamical system in which a large number of different substances react continuously and non-linearly with one another. In order to understand the behavior of a continuous non-linear dynamical system with numerous interacting parts, it is usually insufficient to study behavior of each part in isolation. Instead, the behavior must usually be analyzed as a whole (Tomita et al. 1999).

Considerable amounts of time-domain data are now becoming available concerning the concentration of biologically important chemicals in living organisms. Such data include both gene expression data (obtained from microarrays) and data on the concentration of substances participating in metabolic pathways (Ptashne 1992; McAdams. and Shapiro 1995; Loomis and Sternberg 1995; Arkin, Shen, and Ross 1997; Yuh, Bolouri, and Davidson 1998; Laing, Fuhrman, and Somogyi 1998; D'haeseleer, Wen, Fuhrman, and Somogyi 1999).

The concentrations of substrates, products, intermediate substances, and catalysts (e.g., enzymes) participating in chemical reactions are modeled by non-linear continuous-time differential equations, including various first-order and second-order rate laws, power laws, and the Michaelis-Menten equations (Voit 2000).

The question arises as to whether it is possible to start with observed time-domain concentrations of product substances and automatically create both the topology of the network of chemical reactions and the numerical rates for each reaction of the network. In other words, is it possible to reverse engineer a network from data?

Intuitively, it might seem difficult or impossible to automatically infer both the topology and numerical parameters for a complex network from observed data. However, such intuition may be misleading.

Genetic programming (Koza 1992; Koza, Bennett, Andre, and Keane 1999) is a method for automatically creating a computer program whose behavior satisfies certain high-level requirements. Genetic programming starts with a primordial ooze of thousands of randomly created programs (program trees) and uses the Darwinian principle of natural selection, crossover (sexual recombination), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology to breed a population of programs over a series of generations.

Recent work has demonstrated that genetic programming can automatically create complex networks that exhibit prespecified behavior in fields where the network's behavior is modeled by differential equations (both linear and non-linear) or by other equations (e.g., Maxwell's equations relevant to antennas).

For example, genetic programming is capable of automatically creating both the topology and sizing (component values) for analog electrical circuits (e.g., filters, amplifiers, computational circuits) composed of transistors, capacitors, resistors, and other components merely by specifying the circuit's output — that is, the output data values that would be observed if one already had the circuit. This reverse engineering of circuits from data is performed by genetic programming even though there is no general mathematical method for creating (synthesizing) both the topology and sizing (component values) of analog electrical circuits from the circuit's desired (or observed) behavior (Koza, Bennett, Andre, and Keane 1999). Seven of the circuits that were created using genetic programming infringe on previously issued patents. Others duplicate the functionality of previously patented inventions in novel ways.

As another example, genetic programming is capable of automatically creating both the topology and sizing (tuning) for controllers composed of time-domain blocks (e.g., integrators, differentiators, multipliers, adders, delays, gains, leads, and lags) merely by specifying the controller's effect on the to-be-controlled plant (Koza, Keane, Yu, Bennett, Mydlowec, and Stiffelman 1999; Koza, Keane, Yu, Bennett, and Mydlowec 2000). This reverse engineering of controllers is performed by genetic programming even though there is no general mathematical method for creating both the topology and sizing for controllers from a high-level statement of the design goals for the controller. Two of the controllers that were created using genetic programming infringe on previously issued patents. Several other controllers designed using genetic programming outperform controllers designed by humans.

As yet another example, it is possible to automatically create antennas composed of a network of wires merely by specifying the antenna's high-level specifications. One such antenna has the key features of a type of antenna invented in the early years of the field of antenna design (Comisky, Yu, and Koza 2000).

Our approach to the problem of automatically creating both the topology and sizing of a network of chemical reactions involves

(1) establishing a representation involving program trees (composed of functions and terminals) for chemical networks,

(2) converting each individual program tree in the population into an analog electrical circuit representing the network of chemical reactions,

(3) obtaining the behavior of the individual network of chemical reactions by simulating the electrical circuit,

(4) defining a fitness measure that measures how well the behavior of an individual network matches the observed data, and

(5) applying genetic programming to breed a population of improving program trees using the fitness measure.

Since the description herein of our methods and results is necessarily severely limited by space, the authors are simultaneously publishing a considerably longer technical report that provides additional details and explanatory figures (Koza, Mydlowec, Lanza, Yu, and Keane 2000).

Section 2 states two illustrative "proof of principle" problems. Section 3 presents a method of representing networks of chemical reactions with program trees. Section 4 presents the preparatory steps for applying genetic programming to the illustrative problem. Section 5 presents the results. Section 6 is the conclusion.

2. Statement of Two Illustrative Problems

The goal in the two problems herein is to automatically create (reverse engineer) *both* the topology and sizing (defined below) of a network of chemical reactions.

The *topology* of a network of chemical reactions comprises

- the number of substrates consumed by each reaction,
- the number of products produced by each reaction,
- the pathways supplying the substrates (either from external sources or other reactions in the network) to each reaction, and
- the pathways dispersing each reaction's products (either to other reactions or external outputs).

We use the term *sizin*g for a network of chemical reactions to encompass the determination of all the numerical values specifying the rates of each reaction.

We chose, for the two illustrative problems, two networks that each incorporate all three of the following noteworthy topological features:

- an internal feedback loop (in which a substance is both consumed and produced by the reactions in the loop),
- a bifurcation point (where one substance is distributed to two different reactions), and
- an accumulation point (where one substance is accumulated from two sources).

The first network (figure 1) consists of four reactions that are part of phospholipid cycle, as presented in the E-CELL cell simulation model (Tomita et al. 1999). This network's external inputs are glycerol (`C00116`) and fatty acids (`C00162`). The network's final product is diacyl-glycerol (`C00165`). The network's four reactions are catalyzed by Glycerol kinase (EC2.7.1.30), Glycerol-1-phosphatase (EC3.1.3.21), Acylglycerol lipase (EC3.1.1.23), and Triacylglycerol lipase (EC3.1.1.3).

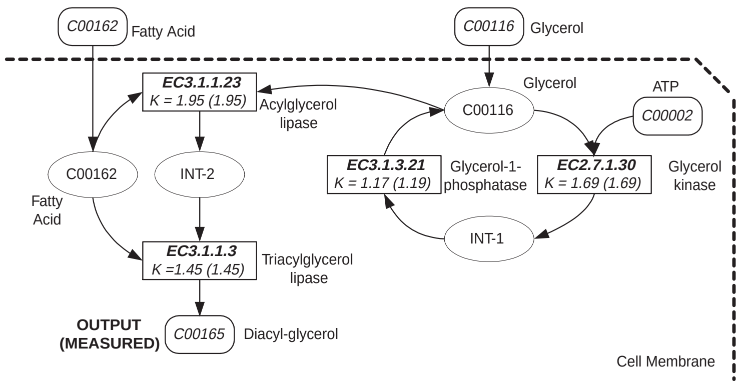

Figure 1 Four reactions from the phospholipid cycle.

To save space, figure 1 shows both the actual network (from the E-CELL model) and the outcome of the run of genetic programming reported in section 5. The actual rates (from the E-CELL model) of each reaction are in parenthesis while the (equal or nearly equal) genetically evolved rates are outside the parenthesis.

The second network (figure 2) consists of three reactions that are involved in the synthesis and degradation of ketone bodies. This network's external inputs are acetoacetyl-CoA and Acetyl-CoA. The network's final product is Acetoacetate. The network's three reactions are catalyzed by 3-oxoacid CoA-transferase (EC 2.8.3.5), Hydroxymethylglutaryl-CoA synthase (EC 4.1.3.5), and Hydroxymethylglutaryl-CoA lyase (EC 4.1.3.4). Again, to save space, figure 2 shows both the actual network and the result of the run reported in section 5.

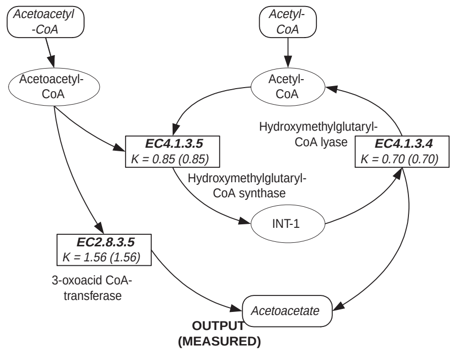

Figure 2 Three reactions for synthesis and degradation of ketone bodies.

3. Representation of Chemical Reaction Networks

Networks of chemical reactions are represented as program trees. Each program tree represents an interconnected network of reactions involving various substances. A chemical reaction may consume one or two substances and produce one or two substances. The consumed substances may be external input substances or intermediate substances produced by reactions. The chemical reactions, enzymes, and substances of a network may be represented by a program tree that contains

- internal nodes representing chemical reaction functions,
- internal nodes representing selector functions that select the reaction's first versus the reaction's second (if any) product,
- external points (leaves) representing substances that are consumed and produced by a reaction,
- external points (leaves) representing enzyme that catalyzes a reaction, and.
- external points (leaves) representing numerical constants (reaction rates).

Each program tree in the population is a composition of functions from the problem's function set and terminals from the problem's terminal set.

3.1. Repertoire of Functions

There are four chemical reaction functions and two selector functions.

The first argument of each chemical reaction function identifies the enzyme that catalyzes the reaction. The second argument specifies the reaction's rate. In addition, there are two, three, or four arguments specifying the substrate(s) and product(s) of the reaction. Table 1 shows the number of substrate(s) and product(s) and overall arity for each of the four chemical reaction functions. The runs in this paper use a first-order and second-order rate law.

Table 1 Four chemical reaction functions.

Function	**Substrates**	**Products**	**Arity**
`CR_1_1`	1	1	4
`CR_1_2`	1	2	5
`CR_2_1`	2	1	5
`CR_2_2`	2	2	6

Each chemical reaction function returns a list composed of the reaction's one or two products. The one-argument `FIRST` function returns the first of the one or two products produced by the chemical reaction function designated by its argument. The one-argument `SECOND` function returns the second of the two products (or, the first product, if the reaction produces only one product).

3.2. Repertoire of Terminals

Some terminals represent substances (input substances, intermediate substances created by reactions, and output substances). Other terminals represent the enzymes that catalyze the chemical reactions. Still other terminals represent numerical constants for the rates of the reactions.

3.3. Constrained Syntactic Structure

The trees are constructed in accordance with a constrained syntactic structure. The root of every result-producing branch must be a chemical reaction function. The enzyme that catalyzes a reaction always appears as the first argument of its chemical reaction function. A numerical value representing a reaction's rate always appears as the second argument of its chemical reaction function. The one or two input arguments to a chemical reaction function can be either a substance terminal or selector function (`FIRST` or `SECOND`). The result of having a selector function as an input argument is to create a cascade of reactions. The one or two output arguments to a function must be a substance terminal. The argument to a one-argument selector function (`FIRST` or `SECOND`) is always a function.

For additional details, see Koza, Mydlowec, Lanza, Yu, and Keane 2000.

4. Preparatory Steps

Before applying genetic programming to a problem of network synthesis, we must specify (1) the architecture of the program trees, (2) the functions, (3) the terminals, (4) the fitness measure, (5) control parameters and termination procedure.

4.1. Program Architecture

Each program tree in the initial random population (generation 0) has one result-producing branch. In subsequent generations, the architecture-altering operations (patterned after gene duplication and gene deletion in nature) may insert and delete result-producing branches to particular individual program trees in the population. Each program tree may have four result-producing branches.

4.2. Function Set

The function set, `F`, is

`F` = {`CR1_1`, `CR1_2`, `CR2_1`, `CR2_2`, `FIRST`, `SECOND`}.

4.3. Terminal Set

For the first network, the terminal set, `T`, is

`T` = {$\Re$, `INT_1`, `INT_2`, `INT_3`, `C00116`, `C00162`, `C00002`, `C00165`}.

$\Re$ denotes a perturbable numerical value. In the initial random generation (generation 0) of a run, each perturbable numerical value is set, individually and separately, to a random value in a chosen range (from 0.0 and 2.0 here).

`INT_1`, `INT_2`, and `INT_3` are the concentrations of intermediate substances 1, 2, and 3 (respectively).

For the first network, `C00116` is the concentration of glycerol; `C00162` is the concentration of fatty acid; `C00002` is the concentration of the cofactor ATP; and `C00165` is the concentration of diacyl-glycerol (the network's final product).

For the second network, the external inputs are acetoacetyl-CoA and Acetyl-CoA and the output product is Acetoacetate.

4.4. Fitness Measure

Genetic programming is a probabilistic algorithm that searches the space of compositions of the available functions and terminals under the guidance of a fitness measure. In order to evaluate the fitness of an individual program tree in the population, the program tree is converted into a directed graph representing the network. One reactor (representing the concentration of the substances participating in the reaction) is inserted into the network for each chemical reaction function that is encountered in a branch. The reactor is labeled with the reaction's enzyme and rate. A directed line entering the reactor is added for each of the reaction's one or two substrate(s). A directed line leaving the reactor is added for each of the reaction's one or two product(s). The first product of a reaction is selected whenever a `FIRST` function is encountered in a branch. The second product of a reaction is selected whenever a `SECOND` function is encountered in a branch.

After the network is constructed, it is converted into an analog electrical circuit. A SPICE netlist is then constructed to represent the circuit. We provide SPICE with subcircuit definitions to implement all the chemical reaction equations. This SPICE netlist is wrapped inside an appropriate set of SPICE commands to carry out an analysis in the time domain. The electrical circuit is then simulated using our modified version of the original 217,000-line SPICE3 simulator (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994). We have embedded our modified version of SPICE as a submodule within our genetic programming system.

Each individual chemical reaction network is exposed to nine test cases. Each test case consists of the concentrations, over time, of four enzymes (EC2.7.1.30, EC3.1.3.21, EC3.1.1.23, and EC3.1.1.3) over 30 half-second time steps (table 2).

The data is obtained from the E-CELL cell simulation model (Tomita et al. 1999).

Fitness is the sum, over the 270 fitness cases, of the absolute value of the difference between the concentration of the end product of the individual reaction network (i.e., diacyl-glycerol `C00165` for the first network and Acetoacetate for the second network) and the observed concentration (data). The smaller the fitness, the better. An individual that cannot be simulated by SPICE is assigned a high penalty value of fitness (10^8). The number of hits is defined as the number of fitness cases (0 to 270) for which the concentration of the measured substances is within 5% of the observed data value.

Table 2 Variations in the levels of the four enzymes.

Signal	EC2.7.1.30	EC3.1.3.21	EC3.1.1.23	EC3.1.1.3
1	Slope-Up	Sawtooth	Step-Down	Step-Up
2	Slope-Down	Step-Up	Sawtooth	Step-Down
3	Step-Down	Slope-Up	Slope-Down	Step-Up
4	Step-Up	Slope-Down	Step-Up	Step-Down
5	Sawtooth	Step-Down	Slope-Up	Step-Up
6	Sawtooth	Step-Down	Knock-Out	Slope-Up
7	Sawtooth	Knock-Out	Slope-Up	Step-Down
8	Knock-Out	Step-Down	Slope-Up	Sawtooth
9	Step-Down	Slope-Up	Sawtooth	Knock-Out

4.5. Control Parameters and Termination Procedure

The population size, *M*, is 100,000. The run was manually monitored and manually terminated when the fitness of many successive best-of-generation individuals appeared to have reached a plateau. For additional details, see Koza, Mydlowec, Lanza, Yu, and Keane 2000.

5. Results

5.1. Phospholipid Cycle

The fitness of the best individual from generation 0 (the randomly created initial population) is 86.4. This individual scores only 126 hits (out of 270). Substance `C00162` (fatty acid) is used as an input substance; however, glycerol (`C00116`) and cofactor ATP (`C00002`) are not. Only two of the four available reactions are used. This individual contains only one of the three noteworthy topological features of the actual network, namely the bifurcation of `C00162` to two reactions.

The best individual of generation 10 has a fitness of 64.0 and scores 151 hits. It improves on generation 0 in that it consumes both `C00162` (fatty acid) and glycerol (`C00116`); however, it contains only two reactions and does not use ATP (`C00002`).

In generation 120, the fitness of the best individual is 2.33. This individual has the same topology as the correct network; however, the numerical rates (sizing) are not yet correct. Consequently, this individual scores only 255 hits.

The best-of-run individual (shown in figure 1 and the electrical circuit of figure 3) appears in generation 225, has near-zero fitness (0.054) and scores 270 hits. It has the correct topology. The rates of three of the four reactions match the correct rates to three significant digits, while the fourth rate is within less than 2% of the correct rate.

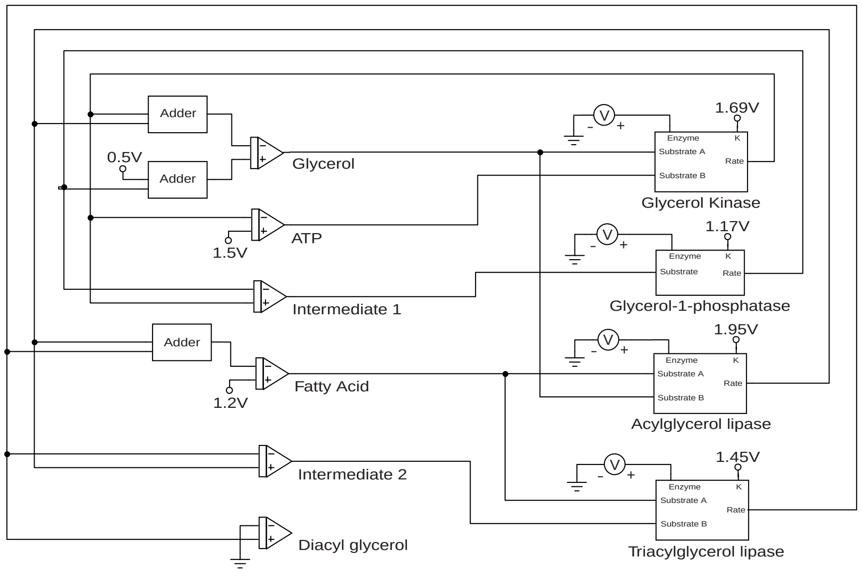

Figure 3 Electrical circuit corresponding to the chemical reaction network.

The six triangles in figure 3 represent integrators. The four rectangles at the right represent chemical reaction functions (all equivalent here to voltage multipliers).

In the best-of-run network, the rate of the two-substrate, one-product reaction catalyzed by Triacylglycerol lipase (EC 3.1.1.3) (found at the very bottom of figures 1 and 3) that produces the final product diacyl-glycerol (`C00165`) is given by

$$\frac{d[C00162]}{dt} = 1.45[C00162][INT\_2][\mathrm{EC}\,3.1.1.3]\ .$$

The rate of the two-substrate, one-product reaction catalyzed by Acylglycerol lipase (EC3.1.1.23) that produces intermediate substance `INT_2` is

$$\frac{d[INT\_2]}{dt} = 1.95[C00162][C00116][\mathrm{EC}\,3.1.1.23] - 1.45[C00162][INT\_2][\mathrm{EC}\,3.1.1.3]\,.$$

The rate of the two-substrate, one-product reaction catalyzed by Glycerol kinase (EC2.7.1.30) that produces intermediate substance `INT_1` in the internal loop is

$$\frac{d[INT\_1]}{dt} = 1.69[C00116][C00002][\mathrm{EC}\,2.7.1.30] - 1.17[INT\_1][\mathrm{EC}\,3.1.3.21]\,.$$

The rate of supply and consumption of cofactor ATP (`C00002`) is

$$\frac{d[ATP]}{dt} = 1.5 - 1.69[C00116][C00002][\mathrm{EC}\,2.7.1.30]$$

The rate of supply and consumption of fatty acids (`C00162`) is

$$\frac{d[C00162]}{dt} = 1.2 - 1.95[C00162][C00116][\mathrm{EC}\,3.1.1.23] - 1.45[C00162][INT\_2][\mathrm{EC}\,3.1.1.3]\,.$$

The rate of supply, consumption, and production of glycerol (`C00116`) is

$$\frac{d[C00116]}{dt} = 0.5 + 1.17[INT\_1][\mathrm{EC}\,3.1.3.21] - 1.69[C00116][C00002][\mathrm{EC}\,2.7.1.30] - 1.95[C00162][C00116][\mathrm{EC}\,3.1.1.23]$$

Notice that genetic programming created the entire metabolic pathway, including topological features (such as the internal feedback loop, the bifurcation point, and the accumulation point) and all numerical rate parameter values (sizing). Genetic programming also determined that two intermediate substances (INT_1 and INT_2) would be used. Genetic programming did this using only the time-domain concentration values of `C00165` (i.e., diacyl-glycerol, the final product).

Both the topology and sizing of the metabolic pathway were created by using 270 time-domain values of the final product. This example (and the one below) demonstrate the principle that it is possible to reverse engineer metabolic pathways from observed data. Note that we have not, at this time, addressed the question of the minimal number of data points necessary to automatically create a correct metabolic pathway or whether the requisite amount of data is available in practical situations.

5.2. Synthesis and Degradation of Ketone Bodies

The best individual from the randomly created population of generation 0 has a fitness value of 76.96 and scores 164 hits. This network contains a bifurcation of one substance (acetoacetyl-CoA) and two double accumulation points. It employs one intermediate substance (INT_1).

The best individual from generation 5 has a fitness value of 22.89 and scores 209 hits. This network has the same topology (but incorrect sizing) of the correct metabolic pathway. In particular, it has the internal loop in which Acetyl-CoA is produced by the reaction catalyzed by Hydroxymethylglutaryl-CoA lyase (EC 4.1.3.4) and, in turn, consumed by the reaction catalyzed by Hydroxymethylglutaryl-CoA synthase (EC 4.1.3.5). It employs one intermediate substance (INT_1).

The fitness of the best network of generation 97 has a fitness of 0.000 (and scores 270 hits). This individual has the same topology as the correct metabolic pathway and the same rates (to three significant digits) for each of the three reactions.

For additional details, see Koza, Mydlowec, Lanza, Yu, and Keane 2000.

6. Conclusion

In summary, driven only by the time-domain concentration values of the final product (Diacyl-glycerol for the network from the phospholipid cycle and Acetoacetate for the network for the synthesis and degradation of ketone bodies), genetic programming created two metabolic pathways, including

- topological features such as the internal feedback loop,
- topological features such as a bifurcation point where one substance is distributed to two different reactions,
- topological features such as an accumulation point where one substance is accumulated from two sources, and
- numerical rates (sizing) for all reactions.

References

Arkin, Adam, Shen, Peidong, and Ross, John. 1997. A test case of correlation metric construction of a reaction pathway from measurements. *Science*. 277. Pages 1275 - 1279. August 29, 1997.

Comisky, William, Yu, Jessen, and Koza, John. 2000. Automatic synthesis of a wire antenna using genetic programming. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada*. Pages 179 - 186.

D'haeseleer, Patrik, Wen, Xiling, Fuhrman, Stefanie, and Somogyi, Roland. 1999. Linear modeling of mRNA expression levels during CNS development and injury. In Altman, Russ B. Dunker, A. Keith, Hunter, Lawrence, Klein, Teri E., and Lauderdale, Kevin (editors). *Pacific Symposium on Biocomputing '99.* Singapore: World Scientific. Pages 41 - 52.

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

Koza, John R., Keane, Martin A., Yu, Jessen, Bennett, Forrest H III, and Mydlowec, William. 2000. Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*. (1) 121 - 164.

Koza, John R., Keane, Martin A., Yu, Jessen, Bennett, Forrest H III, Mydlowec, William, and Stiffelman, Oscar. 1999. Automatic synthesis of both the topology and parameters for a robust controller for a non-minimal phase plant and a three-lag plant by means of genetic programming. *Proceedings of 1999 IEEE Conference on Decision and Control*. Pages 5292 - 5300.

Koza, John R., Mydlowec, William, Lanza, Guido, Yu, Jessen, and Keane, Martin A. 2000. *Reverse Engineering and Automatic Synthesis of Metabolic Pathways from Observed Data Using Genetic Programming*. Stanford Medical Informatics Technical Report SMI-2000-0851.

Laing, Shoudan, Fuhrman, Stefanie, and Somogyi, Roland. 1998. REVEAL: A general reverse engineering algorithm for inference of genetic network architecture. In Altman, Russ B. Dunker, A. Keith, Hunter, Lawrence, and Klein, Teri E. (editors). *Pacific Symposium on Biocomputing '98*. Singapore: World Scientific. Pages 18 - 29.

Loomis, William F. and Sternberg, Paul W. 1995. Genetic networks. *Science*. 269. Page 649. August 4, 1995.

McAdams, Harley H. and Shapiro, Lucy. 1995. Circuit simulation of genetic networks. *Science*. 269. Pages 650-656. August 4, 1995.

Ptashne, Mark. 1992. *A Genetic Switch: Phage λ and Higher Organisms*. Second Edition. Cambridge, MA: Cell Press and Blackwell Scientific Publications.

Quarles, Thomas, Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. 1994. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California. Berkeley, CA. March 1994.

Tomita, Masaru, Hashimoto, Kenta, Takahashi, Kouichi, Shimizu, Thomas Simon, Matsuzaki, Yuri, Miyoshi, Fumihiko, Saito, Kanako, Tanida, Sakura, Yugi, Katsuyuki, Venter, J. Craig, Hutchison, Clyde A. III. 1999. E-CELL: Software environment for whole cell simulation. *Bioinformatics*. Volume 15 (1) 72-84.

Voit, Eberhard O. 2000. *Computational Analysis of Biochemical Systems*. Cambridge: Cambridge University Press.

Yuh, Chiou-Hwa, Bolouri, Hamid, and Davidson, Eric H. 1998. Genomic cis-regulatory logic: Experimental and computational analysis of a sea urchin gene. *Science*. 279. Pages 1896 - 1902.