Foundations of Systems Biology

First Edition

edited by
Hiroaki Kitano

This book was set in Times Roman by the author using the LaTeX document preparation system.
Printed on recycled paper and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

The political economy of trade policy: papers in honor of Jagdish Bhagwati / edited by Robert C. Feenstra, Gene M. Grossman, Douglas A. Irwin
p. cm.
Includes bibliographical references and index.
ISBN 0-262-06186-4 (alk. paper)

Contents

Contributors

Mutsuki Amano
Division of Signal Transduction,
Nara Institute of Science and Technology.

Katja Bettenbrock
bettenbrock@mpi-magdeburg.mpg.de
Max-Planck-Institute for Dynamics of Complex Technical Systems.

Hamid Bolouri
hbolouri@caltech.edu
JST/ERATO Kitano Systems Biology Project,
Control and Dynamical Systems, California Institute of Technology, and
Division of Biology, California Institute of Technology, and
Science and Technology Research Centre, University of Hertfordshire.

Dennis Bray
d.bray@zoo.cam.ac.uk
Department of Zoology, University of Cambridge.

Jehoshua Bruck
bruck@paradise.caltech.edu
Division of Engineering and Applied Science, California Institute of Technology.

John Doyle
doyle@cds.caltech.edu
Control and Dynamical Systems, California Institute of Technology.

Andrew Finney
afinney@cds.caltech.edu
JST/ERATO Kitano Systems Biology Project,
Control and Dynamical Systems, California Institute of Technology.

Ernst Dieter Gilles
gilles@mpi-magdeburg.mpg.de
Max-Planck-Institute for Dynamics of Complex Technical Systems.

Martin Ginkel
ginkel@mpi-magdeburg.mpg.de
Max-Planck-Institute for Dynamics of Complex Technical Systems.

Shugo Hamahashi
shugo@symbio.jst.go.jp
JST/ERATO Kitano Symbiotic Systems Project,
Systems Biology Institute, and
Department of Computer Science, Keio University.

Michael Hucka
mhucka@cds.caltech.edu
JST/ERATO Kitano Systems Biology Project,
Control and Dynamical Systems, California Institute of Technology.

Kozo Kaibuchi
Division of Signal Transduction,

Nara Institute of Science and Technology.,
Department of Cell Pharmacology, Nagoya University.

Mitsuo Kawato
JST/ERATO Kawato Dynamic Brain Project,
Human Information Processing Research Laboratories,
ATR.

Martin A. Keane
makeane@ix.netcom.com
Econometric Inc.

Hiroaki Kitano
kitano@symbio.jst.go.jp
JST/ERATO Kitano Symbiotic Systems Project, Systems Biology Institute.,
Control and Dynamical Systems, California Institute of Technology, and
Sony Computer Science Laboratories, Inc.,

John R. Koza
koza@stanford.edu
Biomedical Informatics,
Department of Medicine,
Department of Electrical Engineering,
Stanford University.

Andreas Kremling
kre@mpi-magdeburg.mpg.de
Max-Planck-Institute for Dynamics of Complex Technical Systems.

Shinya Kuroda
kuroda@fido.cpmc.columbia.edu
JST/ERATO Kawato Dynamic Brain Project,
Division of Signal Transduction, Nara Institute of Science and Technology,
Present address: Center for Neurobiology and Behavior, Columbia University.

Koji M. Kyoda
kyoda@symbio.jst.go.jp
JST/ERATO Kitano Symbiotic Systems Project,
Systems Biology Institute,
and
Department of Fundamental Science and Technology,
Keio University.

Guido Lanza
guido@pharmix.com
Genetic Programming Inc.

Andre Levchenko
andre@paradise.caltech.edu
Division of Engineering and Applied Science,
California Institute of Technology.

Pedro Mendes
mendes@vt.edu
Virginia Bioinformatics Institute,
Virginia Polytechnic Institute and State University.

Satoru Miyano
miyano@ims.u-tokyo.ac.jp
JST/ERATO Kitano Symbiotic Systems Project,
Human Genome Center, Institute of Medical Science,
University of Tokyo.

Eric Mjolsness
mjolsness@jpl.nasa.gov
Jet Propulsion Laboratory,
Division of Biology, California Institute of Technology
and
JST/ERATO Kitano Symbiotic Systems Project,

Mineo Morohashi
moro@symbio.jst.go.jp

JST / ERATO Kitano Symbiotic Systems Project,
Systems Biology Institute,
and
Department of Fundamental Science and Technology,
Keio University.

William Mydlowec
myd@cs.stanford.edu
Genetic Programming Inc.

Masao Nagasaki
masao@ims.u-tokyo.ac.jp
JST / ERATO Kitano Symbiotic Systems Project,
Department of Information Science,
Human Genome Center, Institute of Medical Science,
University of Tokyo.

Yoichi Nakayama
ynakayam@sfc.keio.ac.jp
Institute for Advanced Biosciences,
Keio University.

Shuichi Onami
sonami@symbio.jst.go.jp
JST / ERATO Kitano Symbiotic Systems Project,
Systems Biology Institute.,
Control and Dynamical Systems,
California Institute of Technology.

Herbert Sauro
hsauro@cds.caltech.edu
JST / ERATO Kitano Systems Biology Project,
Control and Dynamical Systems,
California Institute of Technology.

Nicolas Schweighofer
JST / ERATO Kawato Dynamic Brain Project,
Present address: Learning Curve.

Bruce Shapiro
bshapiro@jpl.nasa.gov
Jet Propulsion Laboratory,
California Institute of Technology.

Thomas Simon Shimizu
tss26@cam.ac.uk
Department of Zoology,
University of Cambridge.

Jörg Stelling
stelling@mpi-magdeburg.mpg.de
Max-Planck-Institute for Dynamics of Complex Technical Systems.

Paul W. Sternberg
pws@its.caltech.edu
Division of Biology and Howard Hughes Medical Institute,
California Institute of Technology.

Zoltan Szallasi
zszallas@mxc.usuhs.mil
Department of Pharmacology,
Uniformed Services University of the Health Sciences.

Masaru Tomita
mt@sfc.keio.ac.jp
Institute for Advanced Biosciences,
Keio University.

Mattias Wahde
mwahde@me.chalmers.se
Division of Mechatronics,
Chalmers University of Technology.

Tau-Mu Yi
tmy@caltech.edu
JST / ERATO Kitano Symbiotic Systems Project,
Division of Biology,
California Institute of Technology.

Jessen Yu
jyu@cs.stanford.edu

Part I

Software Platform and Data Resources

1 Automated Reverse Engineering of Metabolic Pathways from Observed Data by Means of Genetic Programming

John R.Koza, William Mydlowec, Guido Lanza,
Jessen Yu, and Martin A. Keane

Recent work has demonstrated that genetic programming is capable of automatically creating complex networks (e.g., analog electrical circuits, 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 modeled by non-linear continuous-time differential equations. This chapter 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 substances and automatically creates both the topology of the network of chemical reactions and the rates of each reaction of a network such that the behavior of the automatically created network matches the observed time-domain data. Specifically, genetic programming automatically created a metabolic pathway involving four chemical reactions that consume glycerol and fatty acid as input, use ATP as a cofactor, and produce diacyl-glycerol as the final product. The metabolic pathway was created from 270 data points. The automatically created metabolic pathway contains three key topological features, including an internal feedback 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 topology and sizing of the entire metabolic pathway was automatically created using only the time-domain concentration values of diacyl-glycerol (the final product).

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 et al., 1997; Yuh et al., 1998; Liang et al., 1998; Mendes and Kell, 1998; D'haeseleer et al., 1999).

The concentrations of substrates, products, and catalysts (e.g., enzymes) participating in chemical reactions are modeled by non-linear continuous-time differential equations, such as the Michaelis-Menten equations (Voit, 2000).

The question arises as to whether it is possible to start with observed time-domain concentrations of substances and automatically create both the topology of the network of chemical reactions and the rates of each reaction that produced the observed data — that is, to automatically reverse engineer the network from the data.

Genetic programming (Koza et al., 1999a) is a method for automatically creating a computer program whose behavior satisfies certain high-level requirements. Recent work has demonstrated that genetic programming can automatically create complex networks that exhibit prespecified behavior in areas where the network's behavior is governed by differential equations (both linear and non-linear).

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 the topology and sizing of analog electrical circuits from the circuit's desired (or observed) behavior (Koza et al., 1999b). Seven of the automatically created circuits infringe on previously issued patents. Others duplicate the functionality of previously patented inventions in a novel way.

As another example, genetic programming is capable of automatically creating both the topology and sizing (tuning) for controllers composed of time-domain blocks such as integrators, differentiators, multipliers, adders, delays, leads, and lags merely by specifying the controller's effect on the to-be-controlled plant (Koza et al., 1999c, 2000a). This reverse engineering of controllers from data is performed by genetic programming even though there is no general mathematical method for creating the topology and sizing for controllers from the controller's behavior. Two of the automatically created controllers infringe on previously issued patents.

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 (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 electrical circuit representing a network of chemical reactions,

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

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

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

The implementation of our approach entails working with five different representations for a network of chemical reactions, namely

Reaction Network: Biochemists often use this representation (shown in Figure 1.1) to represent a network of chemical reactions. In this representation, the blocks represent chemical reactions and the directed lines represent flows of substances between reactions.

Program Tree: A network of chemical reactions can also be represented as a program tree whose internal points are functions and external points are terminals. This representation enables genetic programming to breed a population of programs in a search for a network of chemical reactions whose time-domain behavior concerning concentrations of final product substance(s) closely matches observed data.

Symbolic Expression: A network of chemical reactions can also be represented as a symbolic expression (S-expression) in the style of the LISP programming language. This representation is used internally by the run of genetic programming.

System of Non-Linear Differential Equations: A network of chemical reactions can also be represented as a system of non-linear differential equations.

Analog Electrical Circuit: A network of chemical reactions can also be represented as an analog electrical circuit (as shown in Figure 1.3). Representation of a network of chemical reactions as a circuit facilitates simulation of the network's time-domain behavior.

The goal is to automatically create (reverse engineer) *both* the topology and sizing of a network of chemical reactions.

The *topology* of a network of chemical reactions comprises (1) the number of substrates consumed by each reaction, (2) the number of products produced by each reaction, (3) the pathways supplying the substrates (either from external sources or other reactions) to the reactions, and (4) the pathways dispersing the reaction's products (either to other reactions or external outputs). The *sizing* of a network of chemical reactions consists of the numerical values representing the rates of each reaction.

We chose, as an illustrative problem, a network that incorporates three key topological features. These features include an internal feedback 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 particular chosen network is part of a phospholipid cycle, as presented in the E-CELL cell simulation model (Tomita et al., 1999). The network's external inputs are glycerol and fatty acid. The network's final product is diacyl-glycerol. 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 1.1 shows this network of chemical reactions with the correct rates of each reaction in parenthesis. The rates that are outside the parenthesis are the rates of the best individual from generation 225 of the run of genetic programming.

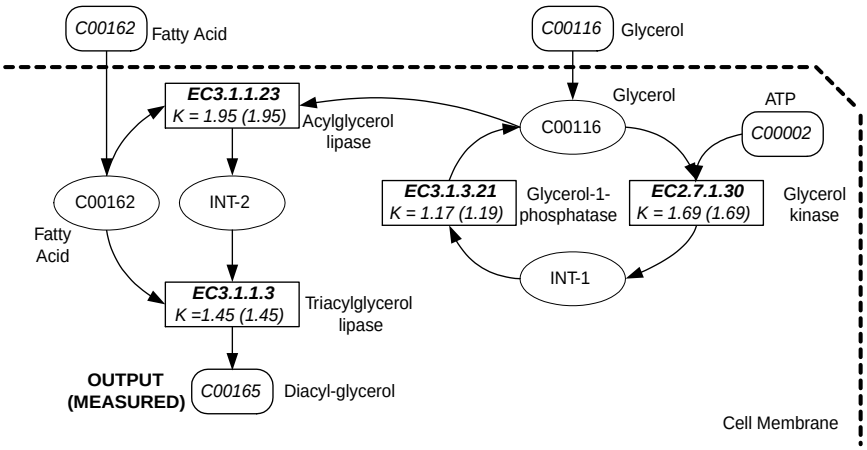

Figure 1.1 Network of chemical reactions involved in the phospholipid cycle

BACKGROUND ON GENETIC PROGRAMMING

Genetic programming (Koza, 1992, 1994a,b; Koza et al., 1999a,b; Koza and Rice, 1992) is a method for automatically creating a computer program whose behavior satisfies user-specified high-level requirements. Genetic programming is an extension of the genetic algorithm (Holland, 1992) in which the population being bred consists of computer programs. Genetic programming starts with a primordial ooze of thousands of randomly created computer 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. Although there are many mathematical algorithms that solve problems by producing a set of numerical values, a run of genetic programming can create both a graphical structure and a set of numerical values. That is, genetic programming will produce not just numerical values, but the structure in which those numerical values reside.

Genetic programming breeds computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of compositions (typically random) of the functions and terminals of the problem.

(2) Iteratively perform the following substeps (referred to herein as a generation) on the population of programs until the termination criterion has been satisfied:

(A)Execute each program in the population and assign it a fitness value using the fitness measure.

(B)Create a new population of programs by applying the following operations. The operations are applied to program(s) selected from the population with a probability based on fitness (with reselection allowed).

(i) Reproduction: Copy the selected program to the new population.

(ii) Crossover: Create a new offspring program for the new population by recombining randomly chosen parts of two selected programs.

(iii) Mutation: Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.

(iv) Architecture-altering operations: Select an architecture-altering operation from the available repertoire of such operations and create one new offspring program for the new population by applying the selected architecture-altering operation to the selected program.

(3) Designate the individual program that is identified by result designation (e.g., the best-so-far individual) as the result of the run of genetic programming. This result may be a solution (or an approximate solution) to the problem.

The individual programs that are evolved by genetic programming are typically multi-branch programs consisting of one or more result-producing branches and zero, one, or more automatically defined functions (subroutines).
The *architecture* of such a multi-branch program involves

(1) the total number of automatically defined functions,

(2) the number of arguments (if any) possessed by each automatically defined function, and

(3) if there is more than one automatically defined function in a program, the nature of the hierarchical references (including recursive references), if any, allowed among the automatically defined functions.

Architecture-altering operations enable genetic programming to automatically determine the number of automatically defined functions, the number of arguments that each possesses, and the nature of the hierarchical references, if any, among such automatically defined functions.

Additional information on genetic programming can be found in books such as (Banzhaf et al., 1998); books in the series on genetic programming from Kluwer Academic Publishers such as (Langdon, 1998); in edited collections of papers such as the *Advances in Genetic Programming* series of books from the MIT Press (Spector et al., 1999); in the proceedings of the Genetic Programming Conference (Koza et al., 1998); in the proceedings of the annual Genetic and Evolutionary Computation Conference (combining the annual Genetic Programming Conference and the International Conference on Genetic Algorithms) held starting in 1999 (Whitley et al., 2000); in the proceedings of the annual Euro-GP conferences held starting in 1998 (Poli et al., 2000); at web sites such as www.genetic-programming.org; and in the Genetic Programming and Evolvable Machines journal (from Kluwer Academic Publishers).

REPRESENTATION OF CHEMICAL REACTION NETWORKS

This section describes a method for representing a network of chemical reactions as a program tree suitable for use in a run of genetic programming. Each program tree represents an interconnected network of chemical 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 representing enzyme that catalyze a reaction, and
- external points 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.

Repertoire of Functions

There are four chemical reaction functions and two selector functions.

The first argument of each chemical reaction (CR) 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 1.1 shows the number of substrate(s) and product(s) and overall arity for each of the four chemical reaction functions. The runs in this chapter use a first-order and second-order rate law.

Table 1.1 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 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 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).

Repertoire of Terminals

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

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 chemical reaction function must be a substance terminal. The argument to a one-argument selector function (FIRST or SECOND) is always a chemical reaction function.

Example

The chemical reactions, enzymes, and substances of a network of chemical reactions may be completely 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 representing enzymes that catalyze a reaction, and
- external points representing numerical constants (reaction rates).

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

Figure 1.2 shows a program tree that corresponds to the metabolic pathway of figure 1.1. The program tree is presented in the style of the LISP programming language. The program tree (Figure 1.2) has two result-producing branches, RPB0 and RPB1. These two branches are connected by means of a connective PROGN function.

As can be seen, there are four chemical reaction functions in Figure 1.2. The first argument of each chemical reaction function is constrained to be an enzyme and the second argument is constrained to be a numerical rate. The remaining arguments are substances, such as externally supplied input substances, intermediate substances produced by reactions within the network, and the final output substance produced by the network. The remaining arguments of each chemical reaction function are marked, purely as a visual aid to the reader, by an arrow. An upward arrow indicates that the substance at the tail of the arrow points to a substrate of the reaction. An downward arrow indicates that the head of the arrow

points to a product of the reaction.

There is a two-substrate, one-product chemical reaction function CR_2_1 in the lower left part of Figure 1.2. For this reaction, the enzyme is Acylglycerol lipase (EC3.1.1.23) (the first argument of this chemical reaction function); its rate is 1.95 (the second argument); its two substrates are fatty acid (C00162) (the third argument) and Glycerol (C00116) (the fourth argument); and its product is Monoacyl-glycerol (C01885) (the fifth argument).

There is a FIRST-PRODUCT function between the two chemical reaction functions in the left half of Figure 1.2. The FIRST-PRODUCT function selects the first of the two products of the lower CR_2_1 function. The line in the program tree from the lower chemical reaction function to the FIRST-PRODUCT function and the line between the FIRST-PRODUCT function and the higher CR_2_1 reaction means that when this tree is converted into a network of chemical reactions, the first (and, in this case, only) substance produced by the lower CR_2_1 reaction is a substrate to the higher reaction. In particular, the product of the lower reaction function (i.e., an intermediate substance called Monoacyl-glycerol) is the second of the two substrates to the higher chemical reaction function (i.e., the fourth argument of the higher function). Thus, although there is no return value for any branch or for the program tree as a whole, the return value(s) of all but the top chemical reaction function of a particular branch (as well the return values of a FIRST-PRODUCT function and a SECOND-PRODUCT function) define the flow of substances in the network of chemical reactions represented by the program tree.

Notice that the fatty acid (C00162) substance terminal appears as a substrate argument to both of these chemical reaction functions (in the left half of Figure 1.2 and also in the left half of Figure 1.1). The repetition of a substance terminal as a substrate argument in a program tree means that when the tree is converted into a network of chemical reactions, the available concentration of this particular substrate is distributed to two reactions in the network. That is, the repetition of a substance terminal as a substrate argument in a program tree corresponds to a bifurcation point where one substance is distributed to two different reactions in the network of chemical reactions represented by the program tree. There is another bifurcation point in this network of chemical reactions where Glycerol (C00116) appears as a substrate argument to both the two-substrate, one-product chemical reaction function CR_2_1 (in the lower left of Figure 1.2 and in the upper left part of Figure 1.1) and the two-substrate, two-product chemical reaction function CR_2_2 (in the upper right part of Figure 1.2 and in the upper right part of Figure 1.1).

Glycerol (C00116) has two sources in this network of chemical reactions. First, it is externally supplied (shown at the top right of Figure 1.1). Second, this substance is the product of the one-substrate, two-product chemical reaction function CR_1_2 (in the middle of Figure 1.1 and in the

lower right of Figure 1.2). When a substance in a network has two or more sources (by virtue either of being externally supplied, by virtue of being a product of a reaction of a network, or any combination thereof), the substance is accumulated. When the program tree is converted into a network, all the sources of this substance are pooled. That is, there is an accumulation point for the substance.

Also, Glycerol (C00116) appears as part of an internal feedback loop consisting of two reactions, namely

- the one-substrate, two-product chemical reaction function CR_1_2 catalyzed by EC3.1.3.21 (in the middle of Figure 1.1 and in the lower right of Figure 1.2) and
- the two-substrate, two-product chemical reaction function CR_2_2 catalyzed by EC2.7.1.30 (in the upper right part of Figure 1.2 and in the right part of Figure 1.1).

The presence of an internal feedback loop is established in this network because of the following two features of this program tree:

- There exists a substance, namely sn-Glycerol-3-Phosphate (C00093) such that this substance
  - –is a product (sixth argument) that is produced by the two-substrate, two-product chemical reaction function CR_2_2 (catalyzed by EC2.7.1.30) in the upper right part of Figure 1.2, and
  - –is also a substrate that is consumed by the one-substrate, two-product chemical reaction function CR_1_2 (catalyzed by EC3.1.3.21) in the lower right part of Figure 1.2 that lies beneath the CR_2_2 function.
- There exists a second substance, namely glycerol (C00116), that
  - –is a product that is produced by the chemical reaction function CR_1_2 (catalyzed by EC3.1.3.21) and
  - –is a substrate that is consumed by the chemical reaction function CR_2_2 (catalyzed by EC2.7.1.30).

In summary, the network of Figure 1.2 contains the following three noteworthy topological features:

- an internal feedback loop in which Glycerol (C00116) is both consumed and produced in the loop,
- two bifurcation points (one where Glycerol is distributed to two different reactions and one where and fatty acid is distributed to two different reactions), and
- an accumulation point where one substance, namely Glycerol, is accumulated from two sources.

A Stanford University technical report provides additional details and explanatory figures (Koza, Mydlowec, Lanza, Yu, and Keane 2000).

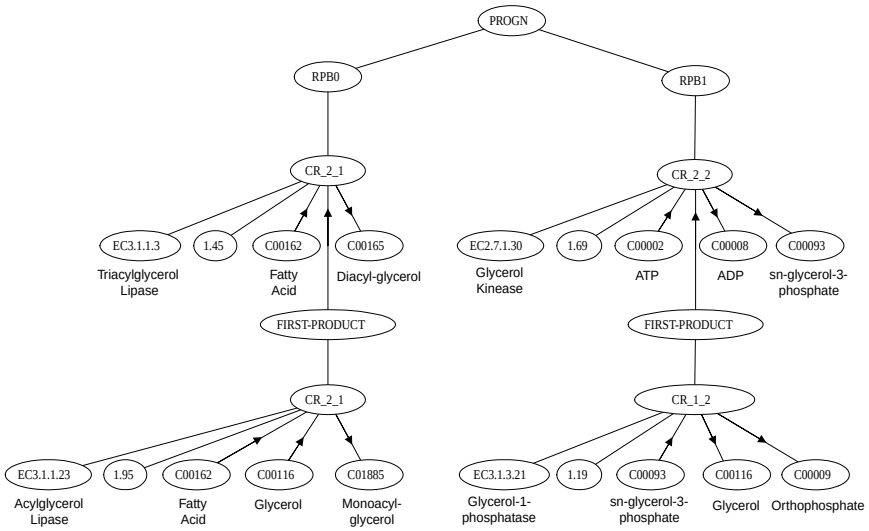

Figure 1.2 Program tree corresponding to metabolic pathway of Figure 1.1.

Figure 1.3 shows the electrical circuit corresponding to the network of Figure 1.1. The triangles in the figure represent integrators.

PREPARATORY STEPS

Six major preparatory steps are required before applying genetic programming: (1) determine the architecture of the program trees, (2) identify the functions, (3) identify the terminals, (4) define the fitness measure, (5) choose control parameters for the run, and (6) choose the termination criterion and method of result designation. For additional details, see (Koza et al., 2000b).

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.

Function Set

The function set, F, consists of six functions.

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

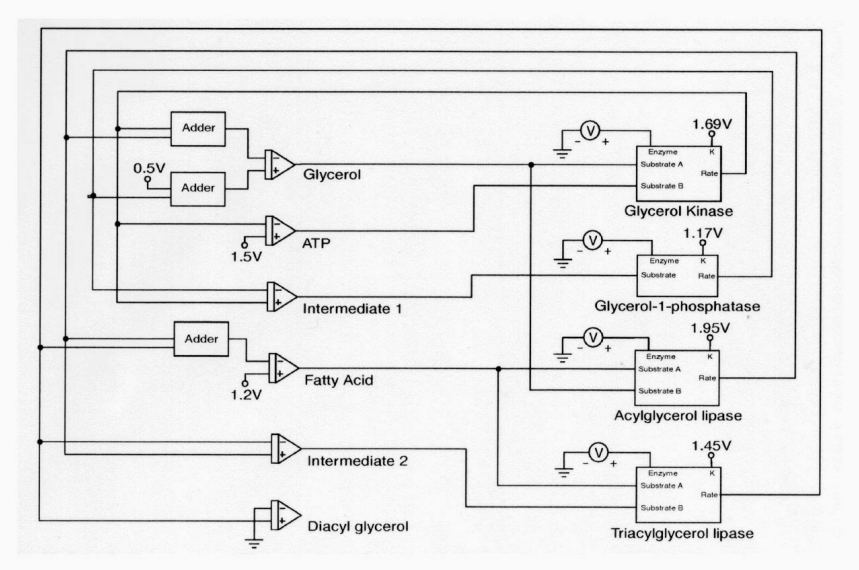

Figure 1.3 Electrical circuit corresponding to the chemical reaction network of Figure 1.1.

Terminal Set

The terminal set, T, is

$$T = \{\Re, C00116, C00162, C00002, C00165, INT\_1, INT\_2, INT\_3, EC2\_7\_1\_30, EC3\_1\_3\_21, EC3\_1\_1\_23, EC3\_1\_1\_3\}.$$

$\Re$ 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).

In the illustrative problem herein, C00116 is the concentration of glycerol. C00162 is the concentration of fatty acid. These two substances are inputs to the illustrative overall network of interest herein. C00002 is the concentration of the cofactor ATP. C00165 is the concentration of diacylglycerol. This substance is the final product of the illustrative network herein. INT_1, INT_2, and INT_3 are the concentrations of intermediate substances 1, 2, and 3 (respectively).

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

EC2_7_1_30, EC3_1_3_21, EC3_1_1_23, and EC3_1_1_3 are enzymes.

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. The result-producing branches are executed from left to right. The functions in a particular result-producing branch are executed in a depth-first manner. 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, the pathway is converted into an electrical circuit. A SPICE netlist is then constructed to represent the electrical 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 analysis in the time domain (described below). The electrical circuit is then simulated using our modified version of the original 217,000-line SPICE3 simulator (Quarles et al., 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 time-domain signals (table 2) representing the time-varying concentrations 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. None of these time series patterns are extreme. Each has been structured so as to vary the concentrations between 0 and 2.0 in a pattern to which a living cell might conceivably be exposed.

There are a total of 270 data points. The data was obtained from the E-CELL cell simulation model (Tomita et al., 1999; Voit, 2000).

The concentrations of all intermediate substances and the network's final product are 0 at time step 0.

For the runs in this paper, Glycerol (C00116), Fatty acid (C00162), and ATP (C00002) are externally supplied at a constant rate (table 3). That is, these values are not subject to evolutionary change during the run.

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 and the observed concentration of diacylglycerol (C00165). The smaller the fitness, the better. An individual that cannot be simulated by SPICE is assigned a high penalty value of fitness

Table 1.2 Variations in the levels of the four enzymes

Signal	EC2.7.1.30	EC3.1.3.21	EC.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	Step-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

Table 1.3 Rates for three externally supplied substances

Substance	Rate
Glycerol (C00116)	0.5
Fatty acid (C00162)	1.2
ATP (C00002)	1.5

(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.

See Koza, Mydlowec, Lanza, Yu, and Keane 2000 for additional details.

Control Parameters for the Run

The population size, M, is 100,000. A generous maximum size of 500 points (for functions and terminals) was established for each result-producing branch. The percentages of the genetic operations for each generation is 58.5% one-offspring crossover on internal points of the program tree other than perturbable numerical values, 6.5% one-offspring crossover on points of the program tree other than perturbable numerical values, 1% mutation on points of the program tree other than perturbable numerical values, 20% mutation on perturbable numerical values, 10% reproduction, 3% branch creation, and 2% subroutine deletion. The other parameters are the default values that we apply to a broad range of problems (Koza et al., 1999a).

Termination

The run was manually monitored and manually terminated when the fitness of many successive best-of-generation individuals appeared to have reached a plateau.

Implementation on Parallel Computing System

We used a home-built Beowulf-style (Sterling et al., 1999; Koza et al., 1999a) parallel cluster computer system consisting of 1,000 350 MHz Pentium II processors (each accompanied by 64 megabytes of RAM). The system has a 350 MHz Pentium II computer as host. The processing nodes are connected with a 100 megabit-per-second Ethernet. The processing nodes and the host use the Linux operating system. The distributed genetic algorithm with unsynchronized generations and semi-isolated subpopulations was used with a subpopulation size of Q = 500 at each of D = 1,000 demes. As each processor (asynchronously) completes a generation, four boatloads of emigrants from each subpopulation are dispatched to each of the four toroidally adjacent processors. The 1,000 processors are hierarchically organized. There are $5 \times 5 = 25$ high-level groups (each containing 40 processors). If the adjacent node belongs to a different group, the migration rate is 2% and emigrants are selected based on fitness. If the adjacent node belongs to the same group, emigrants are selected randomly and the migration rate is 5% (10% if the adjacent node is in the same physical box).

RESULTS

The population for the initial random generation (generation 0) of a run of genetic programming is created at random. The fitness of the best individual (Figure 1.4) from generation 0 is 86.4. This individual scores 126 hits (out of 270). Substance C00162 (fatty acid) is used as an input substance to this metabolic pathway; however, glycerol (C00116) and ATP (C00002) are not. Two of the four available reactions (EC 3.1.1.23 and EC 3.1.1.3) are used. However; a third reaction (EC 3.1.3.21) consumes a non-existent intermediate substance (INT_2) and the fourth reaction (EC 2.7.1.30) is not used at all. This metabolic pathway contains one important topological feature, namely the bifurcation of C00162 to two different reactions. However, this metabolic pathway does not contain any of the other important topological features of the correct metabolic pathway.

In generation 10, the fitness of the best individual (Figure 1.5) is 64.0. This individual scores 151 hits. This metabolic pathway is superior to the best individual of generation 0 in that it uses both C00162 (fatty acid) and glycerol (C00116) as external inputs. However, this metabolic pathway does not use ATP (C00002). This metabolic pathway is also defective in that it contains only two of the four reactions.

In generation 25, the fitness of the best individual (figure 1.6) is 14.3. This individual scores 224 hits. This metabolic pathway contains all four of the available reactions. This metabolic pathway is more complex than previous best-of-generation individuals in that it contains two topological features not previously seen. First, this metabolic pathway contains an internal feedback loop in which one substance (glycerol C00116) is con-

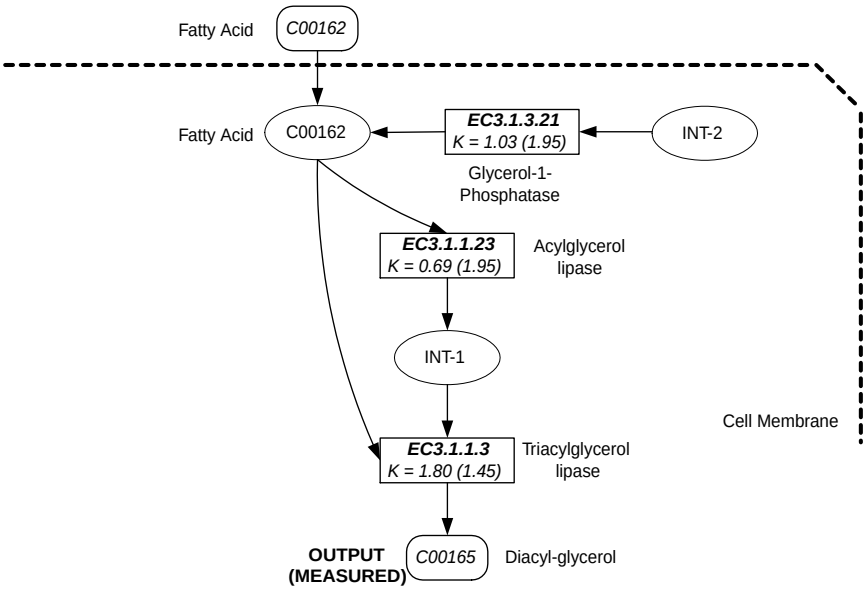

Figure 1.4 Best of generation 0

sumed by one reaction (catalyzed by enzyme EC 2.7.1.30), produced by another reaction (catalyzed by enzyme EC 3.1.3.21), and then supplied as a substrate to the first reaction. Second, this metabolic pathway contains a place where there is an addition of quantities of one substance. Specifically, glycerol (C00116) comes from the reaction catalyzed by enzyme EC 3.1.3.21 and is also externally supplied. This metabolic pathway also contains two substances (C00116 and C00162) where a substance is bifurcated to two different reactions.

$$\frac{d[ATP]}{dt} = 1.5 - 1.69[C00116][C00002][EC2.7.1.30] \qquad (1.1)$$

In generation 120, the fitness of the best individual (Figure 1.7) is 2.33. The cofactor ATP (C00002) appears as an input to this metabolic pathway. This pathway has the same topology as the correct network.. However, the numerical values (sizing) is not yet correct and this individual scores only 255 hits.

The best-of-run individual (Figure 1.1) appears in generation 225. Its fitness is almost zero (0.054). This individual scores 270 hits (out of 270). In addition to having the same topology as the correct metabolic pathway, the rate constants of three of the four reactions match the correct rates (to three significant digits) while the fourth rate differs by only about 2% from the correct rate (i.e., the rate of EC 3.1.3.21 is 1.17 compared with 1.19 for the correct network).

In the best-of-run network from generation 225, the rate of production

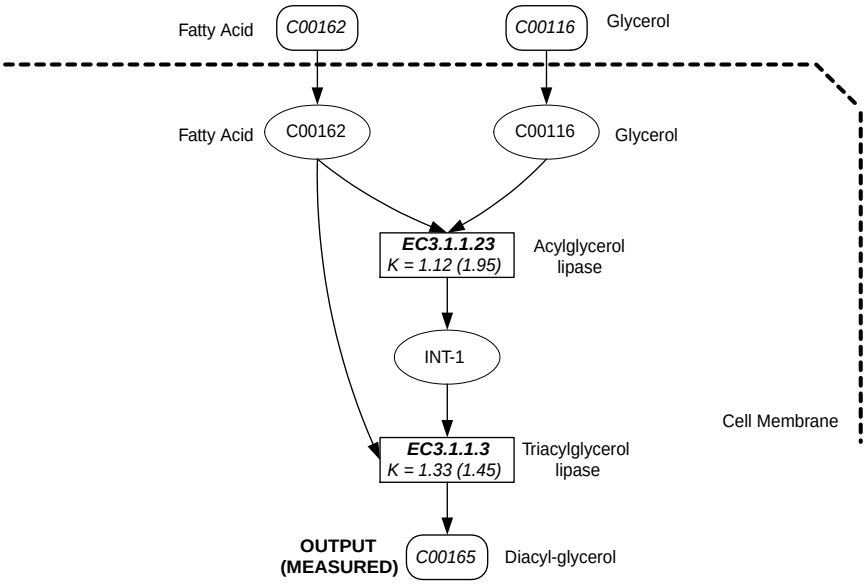

Figure 1.5 Best of generation 10

of the network's final product, diacyl-glycerol (C00165), is given by

$$\frac{d[C00165]}{dt} = 1.45[C00162][INT_2][EC3.1.1.3] \tag{1.2}$$

Note that genetic programming has correctly determined that the reaction that produces the network's final product diacyl-glycerol (C00165) has two substrates and one product; it has correctly identified enzyme EC3.1.1.3 as the catalyst for this final reaction; it has correctly determined the rate of this final reaction as 1.45; and it has correctly identified the externally supplied substance, fatty acid (C00162), as one of the two substrates for this final reaction. None of this information was supplied *a priori* to genetic programming.

Of course, genetic programming has no way of knowing that biochemists call the intermediate substance (INT_2) by the name Monoacylglycerol (C01885) (as indicated in Figure 1.1). It has, however, correctly determined that an intermediate substance is needed as one of the two substrates of the network's final reaction and that this intermediate substance should, in turn, be produced by a particular other reaction (described next).

In the best-of-run network from generation 225, the rate of production and consumption of the intermediate substance INT_2 is given by

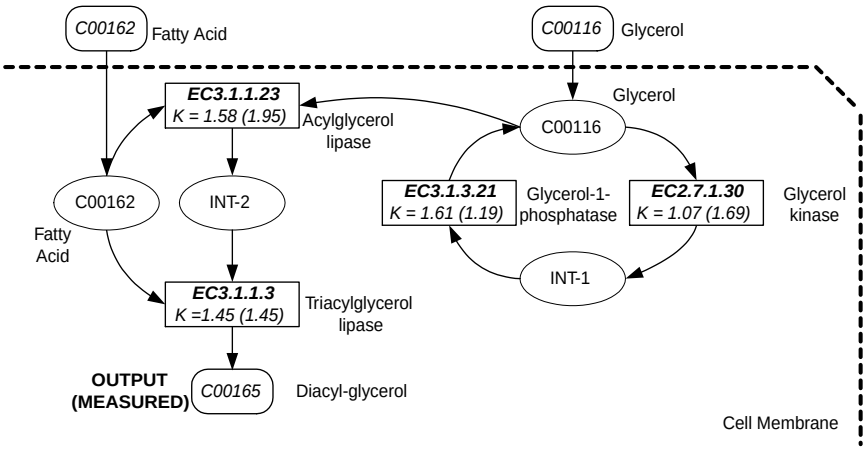

Figure 1.6 Best of generation 25

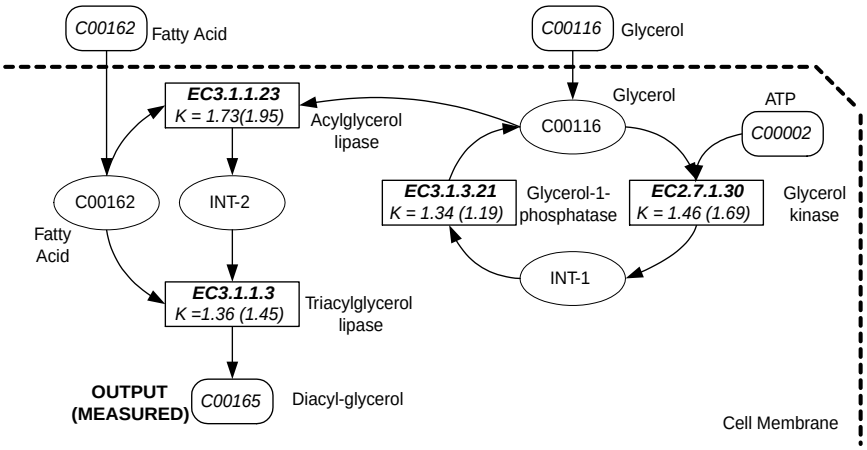

Figure 1.7 Best of generation 120

$$\frac{d[INT_2]}{dt} = 1.95[C00162][C00116][EC3.1.1.23] - 1.45[C00162][INT_2][EC3.1.1.3] \quad (1.3)$$

Again, genetic programming has correctly determined that the reaction that produces the intermediate substance (INT_2) has two substrates and one product; it has correctly identified enzyme EC3.1.1.23 as the catalyst for this reaction; it has correctly determined the rate of this reaction as 1.95; it has correctly identified two externally supplied substances, fatty acid (C00162) and glycerol (C00116), as the two substrates for this reaction.

In the best-of-run network from generation 225, the rate of production

and consumption of the intermediate substance INT_1 in the internal feedback loop is given by

$$\frac{d[INT_1]}{dt} = 1.69[C00116][C00002][EC2.7.1.30] - 1.17[INT_1][EC3.1.3.21] \tag{1.4}$$

Note that the numerical rate constant of 1.17 in the above equation is slightly different from the correct rate (as shown in Figure 1.1).

Here again, genetic programming has correctly determined that the reaction that produces the intermediate substance (INT_1) has two substrates and one product; it has correctly identified enzyme EC2.7.1.30 as the catalyst for this reaction; it has almost correctly determined the rate of this reaction to be 1.17 (whereas the correct rate is 1.19. as shown in Figure 1.1); it has correctly identified two externally supplied substances, glycerol (C00116) and the cofactor ATP (C00002), as the two substrates for this reaction.

Genetic programming has no way of knowing that biochemists call the intermediate substance (INT_1) by the name sn-Glycerol-3-Phosphate (C00093) (as indicated in Figure 1.1). Genetic programming has, however, correctly determined that an intermediate substance is needed as the single substrate of the reaction catalyzed by Glycerol-1-phosphatase (EC3.1.3.21) and that this intermediate substance should, in turn, be produced by the reaction catalyzed by Glycerol kinase (EC2.7.1.30).

In the best-of-run network from generation 225, the rate of supply and consumption of ATP (C00002) is

$$\frac{d[ATP]}{dt} = 1.5 - 1.69[C00116][C00002][EC2.7.1.30] \tag{1.5}$$

The rate of supply and consumption of fatty acid (C00162) in the best-of-run network is

$$\begin{aligned}\frac{d[C00162]}{dt} = 1.2 &- 1.95[C00162][C00116][EC3.1.1.23] \\ &- 1.45[C00162][INT_2][EC3.1.1.3]\end{aligned} \tag{1.6}$$

The rate of supply, consumption, and production of glycerol (C00116) in the best-of-run network is

$$\begin{aligned}\frac{d[C00116]}{dt} = 0.5 &+ 1.17[INT_1][EC3.1.3.21] \\ &- 1.69[C00116][C00002][EC2.7.1.30] \\ &- 1.95[C00162][C00116][EC3.1.1.23]\end{aligned} \tag{1.7}$$

Again, note that the numerical rate constant of 1.17 in the above equation is slightly different from the correct rate (as shown in Figure 1.1).

Notice the internal feedback loop in which C00116 is both consumed and produced.

In summary, driven only by the time-domain concentration values of the final product C00165 (diacyl-glycerol), genetic programming created both the topology and sizing for an entire metabolic pathway whose time-domain behavior closely matches that of the naturally occurring pathway, including

- the total number of reactions in the network,
- the number of substrate(s) consumed by each reaction,
- the number of product(s) produced by each reaction,
- an indication of which enzyme (if any) acts as a catalyst for each reaction,
- the pathways supplying the substrate(s) (either from external sources or other reactions in the network) to each reaction,
- the pathways dispersing each reaction's product(s) (either to other reactions or external outputs),
- the number of intermediate substances in the network,
- emergent topological features such as
  - –internal feedback loops,
  - –bifurcation points,
  - –accumulation points, and
- numerical rates (sizing) for all reactions.

Genetic programming did this using only the 270 time-domain concentration values of the final product C00165 (diacyl-glycerol).

For additional details, see (Koza et al., 2000b).

CONCLUSION

Genetic programming automatically created a metabolic pathway involving four chemical reactions that took in glycerol and fatty acid as input, used ATP as a cofactor, and produced diacyl-glycerol as its final product. The metabolic pathway was created from 270 data points. The automatically created metabolic pathway contains three key topological features, including an internal feedback 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. This example demonstrates the principle that it is possible to reverse engineer a metabolic pathway from observed data concerning the concentration values of its final product substance.

FUTURE WORK

Numerous directions for future work are suggested by the work described herein.

Improved Program Tree Representation

Although the representation used herein yielded the desired results, the authors believe that alternative representations for the program tree (i.e., the function set, terminal set, and constrained syntactic structure) would significantly improve efficiency of the search. The authors are currently contemplating a developmental approach.

Minimum Amount of Data Needed

The work in this chapter has not addressed the important question of the minimal number of data points necessary to automatically create a correct metabolic pathway or the question whether the requisite amount of data is available in practical situations.

Opportunities to Use Knowledge

There are numerous opportunities to incorporate and exploit preexisting knowledge about chemistry and biology in the application of the methods described in this chapter.

The chemical reactions functions used in this chapter (i.e., CR_1_1, CR_1_2, CR_2_1, CR_2_2) are intentionally open-ended in the sense that they permit great flexibility and variety in the networks that can be created by the evolutionary process. However, there is a price, in terms of efficiency of the run, that is paid for this flexibility and generality. Alternative chemical reaction functions that advantageously incorporate pre-

existing knowledge might be defined and included in the function set.

For example, a particular substrate, a particular product, or both might be made part of the definition of a new chemical reaction function. For example, a variant of the CR_2_2 chemical reaction function might be defined in which ATP is hard-wired as one of the substrates and ADP is hard-wired as one of products. This new chemical reaction function would have only one free substrate argument and one free product argument. This new chemical reaction function might be included in the function set in addition to (and conceivably in lieu of) the more general and open-ended CR_2_2 chemical reaction function. This new chemical reaction function would exploit the well-known fact that there are a number of biologically important and biologically common reactions that employ ATP as one of its two substrates and produce ADP as one of its products.

Similarly, a particular enzyme might be made part of the definition of a new chemical reaction function. That is, a chemical reaction function with *k* substrates and *j* products might be defined in which a particular enzyme is hard-wired. This new chemical reaction function would not possess an argument for specifying the enzyme. This new chemical reaction function would exploit knowledge of the arity of reactions catalyzed by a particular enzyme.

Also, a known rate might be made part of the definition of a new chemical reaction function. This approach might be particularly useful in combination with other alternatives mentioned above.

Designing Alternative Metabolisms

Mittenthal et al. (1998) have presented a method for generating alternative biochemical pathways. They illustrated their method by generating diverse alternatives to the non-oxidative stage of the pentose phosphate pathway. They observed that the naturally occurring pathway is especially favorable in several respects to the alternatives that they generated. Specifically, the naturally occurring pathway has a comparatively small number of steps, does not use any reducing or oxidizing compounds, requires only one ATP in one direction of flux, and does not depend on recurrent inputs.

Mendes and Kell (1998) have also suggested that novel metabolic pathways might be artificially constructed.

It would appear that genetic programming could also be used to generate diverse alternatives to naturally occurring pathways. Conceivably, realizable alternative metabolisms might emerge from such evolutionary runs.

In one approach, the fitness measure in a run of genetic programming might be oriented toward duplicating the final output(s) of the naturally occurring pathway (as was done in this chapter). However, instead of harvesting only the individual from the population with the very best

value of fitness, individuals that achieve a slightly poorer value of fitness could be examined to see if they simultaneously possess other desirable characteristics.

In a second approach, the fitness measure in a run of genetic programming might be specifically oriented to factors such as the pathway's efficiency or use or non-use of certain specified reactants or enzymes.

In a third approach, the fitness measure in a run of genetic programming might be specifically oriented toward achieving novelty. Genetic programming has previously been used as an invention machine by employing a two-part fitness measure that incorporates both the degree to which an individual in the population satisfies the certain performance requirements and the degree to which the individual does not possess the key characteristics of previously known solutions (Koza et al., 1999a,c).

ACKNOWLEDGEMENTS

Douglas B. Kell of the University of Wales made helpful comments on a draft of this material.

References

Arkin, A., Peidong, S., and Ross, J. (1997). A test case of correlation metric construction of a reaction pathway from measurements. *Science* 277:1275–1279.

Banzhaf, W., Nordin, P., Keller, R.E., and Francone, F.D. (1998). *Genetic Programming – An Introduction*. San Francisco, CA: Morgan Kaufmann and Heidelberg: dpunkt.

Comisky, W., and Yu, J., and Koza, J. (2000). Automatic synthesis of a wire antenna using genetic programming. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada*.

D'haeseleer, P., Wen, X., Fuhrman, S., and Somogyi, R. (1999). Linear modeling of mRNA expression levels during CNS development and injury. *Proc. Paficic Symposium on Biocomputing'99* pp.41–52.

Holland, J.H. (1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press 1975. Second edition. Cambridge, MA: The MIT Press 1992.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J.R. (1994a). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Koza, J.R. (1994b). *Genetic Programming II Videotape: The Next Generation*. MIT Press.

Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., and Riolo, R. (editors). (1998). *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Bennett III, F.H, Andre, D, and Keane, M.A. (1999a). *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Bennett III, F.H, Andre, D., Keane, M.A., and Brave, S. (1999b). *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

Koza, J.R., Keane, M.A., Yu, J., Bennett III, F.H., Mydlowec, W., and Stiffelman, O. (1999c). 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* pp.5292–5300.

Koza, J.R., Keane, M.A., Yu, J., Bennett III, F.H., and Mydlowec, W. (2000a). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines* 1:121–164.

Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., and Keane, M.A. (2000b). *Reverse Engineering and Automatic Synthesis of Metabolic Pathways from Observed Data Using Genetic Programming*. Stanford Medical Informatics Technical Report SMI-2000-0851.

Koza, J.R. and Rice, J.P. (1992). *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Liang, S., Fuhrman, S., and Somogyi, R. (1998). REVEAL: A general reverse engineering algorithm for inference of genetic network architecture. *Proc. Pacific Symposium on Biocomputing '98* pp.18-29.

Langdon, W.B. (1998). *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Amsterdam: Kluwer.

Loomis, W.F. and Sternberg, P.W. (1995). Genetic networks. *Science* 269:649.

McAdams, H.H. and Shapiro, L. (1995). Circuit simulation of genetic networks. *Science* 269:650-656.

Mendes, P. and Kell, D.B. (1998). Non-linear optimization of biochemical pathways: Applications to metabolic engineering and parameter estimation. *Bioinformatics* 14(10):869–883.

Mittenthal, J.E., Ao, Y., Bertrand C., and Scheeline, A. (1998). Designing metabolism: Alternative connectivities for the pentose phosphate pathway. *Bulletin of Mathematical Biology* 60:815–856.

Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., and Fogarty, T.C. (2000). *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 2000, Proceedings*. Lecture Notes in Computer Science. Volume 1802. Berlin, Germany: Springer-Verlag.

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

Quarles, T., 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.

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

Sterling, T.L., Salmon, J., and Becker, D.J., and Savarese, D.F. (1999). *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters*. Cambridge, MA: MIT Press.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T.S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J.C., Hutchison, C.A. (1999). E-CELL: Software environment for whole cell simulation. *Bioinformatics* 15(1):72–84.

Voit, E.O. (2000). *Computational Analysis of Biochemical Systems*. Cambridge: Cambridge University Press.

Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G. (editors). (2000). *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10 - 12, 2000, Las Vegas, Nevada*. San Francisco: Morgan Kaufmann Publishers.

Yuh, C.-H., Bolouri, H., and Davidson, E.H. (1998). Genomic cis-regulatory logic: Experimental and computational analysis of a sea urchin gene. *Science* 279:1896–1902.