AUTOMATIC CREATION OF A GENETIC NETWORK FOR THE *lac* OPERON FROM OBSERVED DATA BY MEANS OF GENETIC PROGRAMMING

Guido Lanza
Genetic Programming Inc., Los Altos, California, guidissimo@hotmail.com
William Mydlowec
Genetic Programming Inc., Los Altos, California, myd@cs.stanford.edu
John R. Koza
Biomedical Informatics, Department of Medicine
Department of Electrical Engineering
Stanford University, Stanford, California, koza@stanford.edu

This paper demonstrates that it is possible to use genetic programming to automatically create (reverse engineer) a computer program representing the logic underlying the genetic network for the expression level of the *lac* operon as measured by its mRNA. Genetic programming starts with observed time-domain expression levels of two genes (REPRESSOR or CAP) and the concentrations of two substances (GLUCOSE or LACTOSE) and automatically creates both a topological arrangement of conditional and comparative functions of the genetic network as well as all necessary numerical parameters of the genetic network whose behavior matches the observed time-domain data.

1. Introduction

Considerable amounts of data are becoming available concerning gene expression levels (obtained from microarrays) and concentrations of other substances that participate in genetic networks (Ptashne 1992; McAdams and Shapiro 1995; Loomis and Sternberg 1995; Yuh, Bolouri, and Davidson 1998; Voit 2000).

The question arises as to whether it is possible to start with observed time-domain gene expression levels and concentrations of other substances and automatically "reverse engineer" the logic underlying a genetic network. This reverse engineering entails creating both a topological arrangement of conditional and comparative functions and all necessary numerical parameters of a genetic network whose behavior matches observed time-domain data.

2. Statement of an Illustrative Problem

Figure 1 is a schematic representation of a genetic network for the expression level of the *lac* operon (composed of the Z, Y, and A genes). The network involves of two genes (REPRESSOR or CAP) and substances (GLUCOSE or LACTOSE).

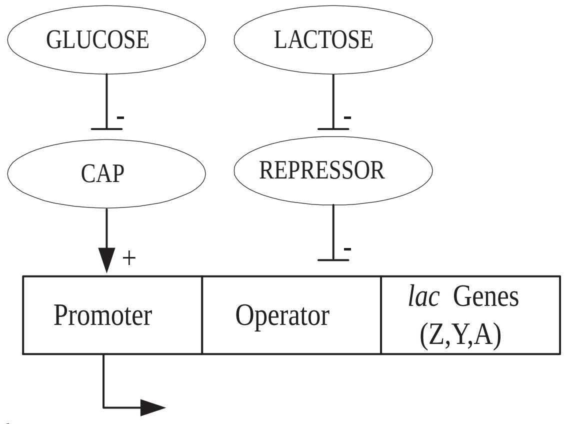

Figure 1 Genetic network for *lac* operon.

The actual performance of the genetic network shown in figure 1 is determined by the expression levels of the two genes and the concentrations of the two substances in relation to threshold values. These threshold values serve as numerical parameters of certain conditional and comparative functions. The logic underlying the genetic network for the *lac* operon can be succinctly written in C-style pseudo code as shown below. The numerical value returned by this program is the expression level of the *lac* operon (`LAC_mRNA_LEVEL`).

```
if(LACTOSE_LEVEL >= LACTOSE_THRESHOLD)
{
    if(GLUCOSE_LEVEL >= GLUCOSE_THRESHOLD)
    {
        LAC_mRNA_LEVEL = low;
    }
    else
    {
        if (CAP_LEVEL >= CAP_THRESHOLD)
        {
            LAC_mRNA_LEVEL = high;
        }
        else
        {
            LAC_mRNA_LEVEL = low;
        }
    }
}
else
{
    if(REPRESSOR_LEVEL >= REPRESSOR_THRESHOLD)
    {
        LAC_mRNA_LEVEL = 0;
    }
    else
    {
        LAC_mRNA_LEVEL = low;
    }
}
```

The goal in this paper is to automatically create (reverse engineer) *both* a topological arrangement of conditional and comparative functions and all necessary numerical parameters that represent the expression of the expression level of the *lac* operon as measured by its mRNA. In other words, we seek to automatically create logic equivalent to that shown above from the time-domain data for the expression levels of the two genes and the concentrations of the two substances.

3. Background of Genetic Programming

Genetic programming (Koza, Bennett, Andre, and Keane 1999) is a method for automatically creating a computer program whose behavior satisfies specified high-level requirements. Genetic programming starts with a primordial ooze of thousands of randomly created computer programs (program trees) and uses the Darwinian principle of natural selection, recombination (crossover), 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 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 (Koza, Bennett, Andre, and Keane 1999). Seven of the automatically created circuits infringe on previously issued patents.

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, Keane, Yu, Bennett, Mydlowec 2000). 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).

4. Representation of Genetic Networks as Computer Programs

Each program tree represents the logic of a genetic network. A program tree is a composition of functions from the function set and terminals from the terminal set and contains

- internal nodes representing conditional and comparative functions,
- external points (leaves) representing expression level of various genes, and
- external points (leaves) representing concentration of substances.

The value returned by the result-producing branch of the program tree is the expression level of the *lac* operation (called LAC_mRNA_LEVEL in the C-style pseudo code above).

4.1. Repertoire of Functions

The three-argument IF function returns the results of evaluating its third argument (the "else" clause) if its first argument is FALSE, but returns the results of evaluating its second argument (the "then" clause) if its first argument is TRUE.

The two-argument < comparative function returns a value of TRUE if its first argument is less than its second argument, but otherwise FALSE.

The two-argument > comparative function performs the opposite function.

4.2. Repertoire of Terminals

GLUCOSE_LEVEL and LACTOSE_LEVEL are substances.

REPRESSOR_LEVEL and CAP_LEVEL are expression levels of genes.

$\Re$ 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 10.0 here). These numerical values will serve as the thresholds in the overall logic of the evolved program.

4.3. Constrained Syntactic Structure

The trees are constructed in accordance with a constrained syntactic structure. The entire program tree returns a floating-point number. The first argument of an IF function must be a comparative function (< or >). The two arguments of a comparative function must be a terminal. The second and third arguments of an IF function may be another IF function or a perturbable numerical value.

5. Preparatory Steps

5.1. Program Architecture

Each program tree has one result-producing branch.

5.2. Function Set

The function set is

F = {IF, <, >}

with arity of three, two, and two, respectively.

5.3. Terminal Set

The terminal set is

T = {GLUCOSE_LEVEL, LACTOSE_LEVEL, REPRESSOR_LEVEL, CAP_LEVEL, $\Re$}.

5.4. 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.

Each individual genetic network is exposed to four time-domain scenarios representing the concentrations of substances (GLUCOSE_LEVEL or LACTOSE_LEVEL) and expression values of genes (REPRESSOR_LEVEL or CAP_LEVEL) over 20 time steps (except that there are only 19 time steps in the first scenario since time $t = 0$ is ignored).

The first of the four fitness cases is based on a high level (10) of GLUCOSE_LEVEL and a low level (0) of LACTOSE_LEVEL. In this context the network is exposed, during the 20 time steps, to all four combinations of high and low values of CAP_LEVEL and REPRESSOR_LEVEL. Broadly speaking, the expression level of CAP_LEVEL initially rises. While CAP_LEVEL is steady, REPRESSOR_LEVEL begins to rise. When REPRESSOR_LEVEL reaches it peak, CAP begins to fall.

The second fitness case is based on a high level (10) of GLUCOSE_LEVEL and a high level (10) of LACTOSE_LEVEL.

The third fitness case is based on a low level (0) of GLUCOSE_LEVEL and a high level (10) of LACTOSE_LEVEL.

The fourth fitness case is based on a low level (0) of GLUCOSE_LEVEL and a low level (0) of LACTOSE_LEVEL.

Fitness is the sum, over the 79 fitness cases, of the absolute weighted value of the difference between the value returned by the result-producing branch and the observed expression level of the *lac* operon (as measured by mRNA). If the value returned by the result-producing branch is within 5% of the observed expression level data, the weight is 1.0; otherwise it is 10. The smaller the fitness, the better.

The number of hits is defined as the number of fitness cases (time steps 1 to 79) for which the difference is within 5% of the correct value.

5.5. Control Parameters for the Run

The population size, *M*, is 10,000. A maximum size of 1,000 points (for functions and terminals) was established for the result-producing branch.

6. Results

The fitness of the best individual from the initial random generation (generation 0) is very high (namely 116.6). This individual scores 0 hits (out of 79).

The best individual evolved during the run of genetic programming appeared in generation 93. This individual has a fitness of 3.15 and scores 78 hits. This actual program is shown below:

```
   (IF (< LACTOSE_LEVEL 9.139 ) (IF (< REPRESSOR_LEVEL 6.270 ) (IF (>
GLUCOSE_LEVEL 5.491 ) 2.02 (IF (< CAP_LEVEL 0.639 ) 2.033 (IF (< CAP_LEVEL
4.858 ) (IF (> LACTOSE_LEVEL 2.511 ) (IF (> CAP_LEVEL 7.807 ) 5.586 (IF (>
LACTOSE_LEVEL 2.114 ) 1.978 2.137 ) ) 0.0 ) (IF (> REPRESSOR_LEVEL 4.015 )
0.036 (IF (< GLUCOSE_LEVEL 5.128 ) 10.0 (IF (< REPRESSOR_LEVEL 4.268 ) 2.022
9.122 ) ) ) ) ) ) (IF (> CAP_LEVEL 0.842 ) 0.0 5.97 ) ) (IF (< CAP_LEVEL 1.769
) 2.022 (IF (< GLUCOSE_LEVEL 2.382 ) (IF (> LACTOSE_LEVEL 1.256 ) (IF (>
LACTOSE_LEVEL 1.933 ) (IF (> GLUCOSE_LEVEL 2.022 ) (IF (< GLUCOSE_LEVEL 5.183
) 6.323 (IF (> CAP_LEVEL 1.208 ) 9.713 0.842 ) ) 10.0 ) (IF (> GLUCOSE_LEVEL
6.270 ) 2.109 ) 1.965 ) ) 0.665 ) 1.982 ) ) )
```

When the logic of this program is simplified and the program is rewritten in C-style pseudo code, the result is as follows:

```
if(LACTOSE_LEVEL < 9.139)
{
    if(REPRESSOR_LEVEL < 6.270)
    {
        LAC_mRNA_LEVEL = 2.022;
    }
    else
    {
        LAC_mRNA_LEVEL = 0.0;
    }
}
else
{
    if(CAP_LEVEL < 1.769)
    {
        LAC_mRNA_LEVEL = 2.022;
    }
```

```
        else
        {
            if(GLUCOSE_LEVEL < 2.382)
            {
                LAC_mRNA_LEVEL = 10.0;
            }
            else
            {
                LAC_mRNA_LEVEL = 1.982;
            }
        }
    }
```

7. Conclusion

The behavior of the above automatically created genetic network closely matches observed time-domain data. Notice that genetic programming automatically created both the topological arrangement of conditional and comparative functions as well as all necessary numerical parameters of the logic for this genetic network.

References

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.

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.

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

McAdams, Harley H. and Shapiro, Lucy. 1995. Circuit simulation of genetic networks. *Science*. 269. 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.

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. 1896 - 1902.