Multivariate Analysis and Reverse Engineering of Signal Transduction Pathways

by

Amy Norris

B.Math., University of Waterloo, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Mathematics, Institute of Applied Mathematics)

We accept this thesis as conforming
to the required standard

The University of British Columbia

April 2002

Abstract

This thesis presents mathematical and computational approaches to studying signal transduction pathways. The biology of cellular signalling is introduced, highlighting the importance of signalling pathways in regulating the processes of cell division and proliferation. A brief history of mathematical approaches to such systems is presented. Two multivariate analysis methods, principal component analysis and clustering, are introduced and applied to both gene expression and simulated protein concentration data. Several recent reverse engineering methods that have been used to study genetic networks are introduced. Finally, a reverse engineering method intended to elucidate the structure of genetic networks is adapted to the study of signal transduction systems.

Contents

Abstract ii

Contents iii

List of Tables vi

List of Figures viii

Acknowledgements xi

Dedication xii

1 An introduction to the study of signal transduction systems 1

1.1 Signal transduction 1

1.2 Epidermal growth factor signaling 7

1.3 Modeling the EGF pathway 9

1.4 The mathematical study of signal transduction 11

2 Multivariate Analysis 14

2.1 Introduction to multivariate covariance and correlation 14

2.2 Principal Component Analysis 16

2.2.1 Mathematical background for PCA 17

2.2.2 Singular Value Decomposition 19

2.2.3 Recent applications to studies of gene expression 19

2.3 Clustering 21

2.3.1 Measuring distance or similarity 21

2.3.2 Hierarchical clustering 23

2.3.3 Self-organizing maps (SOMs) . . . 24
2.3.4 Recent applications of clustering to studies of gene expression . . . 28
2.4 Applications of multivariate analysis techniques to published and simulated data . . . 30
2.4.1 Application of principal component analysis . . . 31
2.4.2 Application of SOMs . . . 41
2.5 Discussion of multivariate analysis results . . . 45

3 Reverse Engineering of Genetic Networks 48
3.1 Discrete Boolean Networks . . . 49
3.1.1 Shannon Entropy and REVEAL . . . 50
3.1.2 Recognizing dependence between genes . . . 52
3.1.3 The REVEAL method . . . 53
3.1.4 Difficulties with REVEAL Implementation . . . 55
3.2 Continuous expression and regulation . . . 56
3.3 Bayesian networks . . . 59
3.3.1 The structure of Bayesian networks . . . 61
3.3.2 Training Bayesian networks . . . 64
3.3.3 Hartemink case study . . . 66
3.3.4 Annotated edges . . . 67
3.3.5 Dynamic Bayesian networks . . . 69
3.3.6 Comments on the study of gene expression data using Bayesian networks 69
3.4 Discussion . . . 70

4 A Reverse Engineering Application 72
4.1 A system of inference of large-scale genetic networks . . . 72
4.2 Part One: Static Boolean network . . . 73
4.3 Part Two: Determination of subnetworks . . . 76
4.3.1 S-systems . . . 76
4.3.2 Genetic algorithm optimization . . . 78
4.4 Application of Part One to the EGF simulation . . . 80
4.5 Genetic algorithm toolbox for MATLAB . . . 83
4.5.1 The evaluation function . . . 86

4.5.2 Selection and reproduction 86
4.5.3 Termination 88
4.6 Modifications to the method 88
4.6.1 Adapting the genetic algorithm toolbox 91
4.6.2 Initiation of the population 93
4.6.3 Mutations 95
4.6.4 Crossovers 95
4.6.5 The Comet-Strike feature 98
4.7 Results of Part Two 98
4.7.1 Crossover tests 98
4.7.2 Application to the test system 99
4.8 Discussion 105

5 Concluding Remarks **110**

Bibliography **112**

Appendix A Finding the maximum of a quadratic form under a quadratic constraint **116**

Appendix B Modifications to the genetic algorithm toolbox **119**

List of Tables

2.1 Data sets analysed using principal components and self-organizing maps in Section 2.4. . . . 31
2.2 Eigenvalues and the proportion of the variance of the entire data set from the first five principal components of gene dataset1 . . . 32
2.3 From Study 2: Eigenvalues and the proportion of the variance of the entire data set from the first five out of 676 principal components for gene dataset2 33
2.4 Eigenvalues and percentage of total variance for the first five principal components from the correlation matrix of EGF dataset3. . . . 40
2.5 A numbered list of the proteins and protein complexes involved in the early events of the EGF signal transduction pathway. . . . 41

3.1 Compiled binary data for the three genes at time points t and $t+1$. . . . 51

4.1 Maximum and minimum relative intensities for each protein in the EGF simulation. . . . 81
4.2 The binary matrix, R, corresponding to a threshold of 5. The rows correspond to the proteins that have been deleted to produce the given effect on the protein in each column. The numbers correspond to the proteins listed in Table 2.5 . . . 82
4.3 Manual grouping of proteins into equivalence sets based on matrix R with 5 threshold. . . . 84
4.4 Kinetic constants for the test network, Equation (4.12) . . . 91
4.5 Contents of the termbank for a network of 4 proteins. For example, row 3 of the table corresponds to the term X_3X_4. The termbank has been sorted for insertion of a term into the differential equation of protein1 (the term is randomly chosen from the first 7 terms of the termbank). Notice that the term of all zeros is eliminated from the bank. . . . 94
4.6 A list of all the possible mutations and the action taken to perform the mutation in order to yield a valid solution. The Current Parameter column lists the type and the sign of parameter selected for mutation. New Parameter designates the sign of the value that is to replace the current parameter. For the mutations of g_{ijk}, the sign of the α_{ij} for the term that the exponent appears in is also listed. Recall that the subscripts refer to the k^{th} protein in the j^{th} term in the differential equation for protein i. . . . 96
4.7 The first four operators created for performing the crossover for the adapted genetic algorithm optimization. . . . 97
4.8 Results of tests of the crossover operators amyXover1-4 with population size 10 and 100 iterations. . . . 99

4.9 A comparison of the kinetic constants from a successful run of the genetic algorithm to the actual values for the test case. This run consisted of 1000 iterations of a population of 20. 101
4.10 Proteins belonging to two of the equivalence sets from Part One, listed in Table 4.3. 103

List of Figures

1.1 A simple picture of the components of the EGF signaling pathway. EGF molecules bind to the extracellular domains of receptors in the cell membrane. The binding causes dimerization of the receptor and subsequent initiation of a series of intracellular responses. The signal is transduced, through a series of phosphorylation reactions, to the cell nucleus. Here gene transcription is initiated by transcription factors. 3
1.2 The early events of the EGF signaling pathway that are included in the model of Kholodenko *et al* [29]. The proteins corresponding to the abbreviations used in this figure and throughout the thesis are listed in Table 2.5 8

2.1 A tree visualization of the results of the hierarchical clustering of a group of British Columbian trees based on height. The scale on the left represents the distance between the two clusters being joined by a given branch. The figures that appear under the tree names along the bottom are the heights of each tree, and the numbers in the graph represent the average tree heights of each newly formed cluster. This is an example of agglomerative hierarchical clustering.(Tree data from www.bcadventure.com) 23
2.2 An illustration of different linkage methods used to determine the most similar clusters at each step of a hierarchical clustering. The circles represent clusters and the dots inside each circle represent the items contained in each cluster. The darker line(s) indicate the shortest of the compared distances in the pairwise comparison between the red cluster and each cluster beneath. The cluster with the shortest distance to the red cluster is combined with the red cluster for the next step of the clustering. (a) *Single linkage* also known as nearest neighbor. The shortest distance between items in each pair of clusters is compared. (b) *Complete linkage* or farthest neighbor. The largest distance between items in the two clusters are compared to find the shortest of these. (c) *Average linkage*. The average of all the distances between items in the pair of clusters are used for comparison. 25

2.3 The training of a SOM. The small dots represent data points. The larger circles represent the nodes of the network arranged in a (3×2) rectangular lattice and located at the position in the data space given by their reference vectors. (a) Data point A is chosen randomly. Node C is the closest node to A. The dotted circle, N_C represents the neighborhood of node C. (b) Node C and the other two nodes that lie within N_C are moved toward A as dictated by the learning function. The next data point, B, is randomly chosen and the new closest node C and its neighborhood N_C are shown. (c) Node C and the two nodes lying within N_C are moved toward B. (d) After many such iterations, the nodes settle at clusters of data points. 27
2.4 From Study 2: Behavior of the first five principal components over time from the covariance matrix for the *cdc15* data. 34
2.5 Plot of the coefficients of the first two components from the covariance matrix of gene dataset2. 35
2.6 Plot of the coefficients of the first two components from the correlation matrix of gene dataset2. 36
2.7 Plot of the coefficients of the first and third components from the correlation matrix of gene dataset2. 37
2.8 Time courses for the first four components from the correlation matrix of EGF dataset4 (top) and EGF dataset3 (bottom). 39
2.9 The concentrations of each protein in the network from EGF dataset4 . . . 40
2.10 Plots of the coefficients of the first two components from the correlation matrix of EGF dataset4 (top) and EGF dataset 3 (bottom). 42
2.11 Centroid time courses for the 4 cluster SOM of EGF dataset5. 43
2.12 Centroid time courses for the 6 cluster SOM of EGF dataset5. 44
2.13 Centroid time courses for the 4 cluster SOM of EGF dataset4. 44
2.14 Clustering of the proteins of the EGF pathway. The results are from a 6 cluster SOM of EGF dataset5. The circles represent the nodes of the network and are arranged in the lattice shape used for the creation of the SOM. The numbers correspond to the proteins listed in Table 2.5 45
2.15 Proteins in the pathway are hatched according to their cluster membership. The results are for a 6 node clustering of EGF dataset5. 46

3.1 The evolution of a three component discrete network according to the rules given below. 50
3.2 Several possible dose response functions with different parameter values. (a) $\alpha = 2$, $\beta = 0.5$, (b) $\alpha = 0.4$, $\beta = 0.5$, (c) $\alpha = 2$, $\beta = -2$. 58
3.3 An example of the graphical structure of a Bayesian network 63
3.4 The two hypotheses being compared using Bayesian scoring. The model on the left represents an outdated hypothesis and that on the right represents the currently accepted system. The genes, whose values are measurable, are shown in boxes, while the gene products are included in the model as latent variables. 67
3.5 Three possible annotated Bayesian models. The question raised in the text is whether (a) would receive a worse score than (b), given that (c) is the true model. 68

4.1 Part One of the method of Maki *et al* for a system of 4 proteins. Proceeding clockwise from the top-left: Each column of the data matrix D is divided by the normal steady-state value for that protein to yield the relative intensity matrix E. E is quantized to binary values with a threshold of 2. The closure of R yields R^*. The equivalence relations are determined: proteins B and D show the same behavior (in R^*) to all other proteins in the network and are therefore grouped together in [2]. Replacing columns B and D by [2], and relabeling A as [1] and C as [3] (single protein equivalence sets) gives us $R^{*'}$. Finally, the topological sort gives us the skeleton matrix S containing only direct connections between equivalence sets. This determines the shape of the graph, shown in the center. 77
4.2 The proteins in each equivalence set and the graph of the equivalence sets. . 83
4.3 Proteins in the EGF pathway are coloured according to their equivalence sets. 84
4.4 Grouping of the EGF pathway proteins according to the equivalence sets listed in Table 4.3 85
4.5 Demonstration of mutation and crossover operators using LEGO. (a) In mutation, a single block from one tower is randomly selected (black arrow) and replaced with a different block in the child. (b) In arithmetic crossover, a detachment position along the tower is randomly selected (black arrow). Both parent towers are broken at that position, and the top halves of the tower are interchanged to yield the two children. 87
4.6 The organization of the genome of an individual. All α parameters appear first, followed by all g parameters. α_{ij} is the rate constant for the j^{th} term in the differential equation of the i^{th} protein. g_{ijk} is the power of the k^{th} protein in the j^{th} term in the differential equation of the i^{th} protein. 90
4.7 A plot showing the relationship between the number of individuals in a population and the fitness value of the optimal solution. The results are for a 50 iteration optimization of the test system (Equation(4.12)). 108
4.8 A plot showing the relationship between the number of individuals in a population and the run time of the algorithm. The results are for a 50 iteration optimization of the test system (Equation(4.12)). 109

Acknowledgements

I would like to thank Leah Keshet for her support and guidance, both in leading me to this very interesting topic and in helping me to get to the point where hopefully I can express my ideas clearly enough to interest others.

I would like to thank my parents and Cath for always being supportive and for listening to me babble on about proteins and DNA, among other things. I love you very much.

I would also like to thank Adrian Secord for many discussions about the material presented here and for helping me improve the speed of the computations. Without his undying support this thesis would not have been possible.

Amy Norris

The University of British Columbia
April 2002

To my Gram, who thought that if anyone would know why there's a moose on the quarter, that it would be me. Well, I didn't know that, but I guess this proves that I know a little about something. Thanks for your confidence in me Gram.

Chapter 1

An introduction to the study of signal transduction systems

Our bodies are composed of myriads of molecules that interact in a very complex, sometimes seemingly random fashion and yet are capable of carrying out the development of a new being from a single cell. The puzzle of how all these pieces work together to produce a functioning organism is one of the scientific challenges of this century and spans disciplines from psychology to physics, from ecology to biochemistry. Mathematics may play a role in interpreting and organizing this huge research effort.

Each cell division and differentiation, each immune response, each synaptic transmission throughout development and through the life span of an individual is instigated by a biological signal communicated between cells. The receipt and intracellular propagation of such a signal and its interpretation by the cell and subsequent response, is referred to as signal transduction and forms the biological motivation for this thesis.

1.1 Signal transduction

The molecular components involved in cellular signaling form *signal transduction pathways*. A signal transduction pathway affecting a cell is composed of the following events:

- A signaling molecule arrives outside the cell
- A receptor on the extracellular surface of the cell membrane interacts with the signaling molecule

- The receptor interacts with intracellular pathway components, starting a cascade of protein interactions that propagates the signal inside the cell
- The signal arrives at its destined location or molecular target and elicits a functional response in the cell.

A simplified example of these steps is shown in Figure 1.1.

Important biotechnological advances in recent years have allowed increasingly detailed studies of a variety of signalling pathways. These advances include production of recombinant DNA, the Polymerase Chain Reaction (PCR) [3], gel electrophoresis [54], microarrays [10], and the serial analysis of gene expression (SAGE) technique [53]. Development of such techniques is ongoing, and large-scale assays of peptides and protein-DNA binding activity are becoming more feasible [2].

Signal origin and recognition by the cell

A signaling molecule may be a protein, small peptide, amino acid, nucleotide, steroid, retinoid, fatty acid derivative or a dissolved gas [3]. Where does a signal come from? There are different types of signaling systems and the signal origin differs among them. For paracrine signaling, the signal originates from a nearby cell and, thus, the signal causes only localized effects. In endocrine signaling, hormones are secreted into the bloodstream and thus may be received by a cell some distance from the origin of the signal. In synaptic signaling, a signaling molecule is released into a synaptic cleft from one neuron and received by another neuron. And finally, a cell may send a signal to itself, which is known as autocrine signaling [3].

There are many types of signaling molecules and also many different receptors that may be present on a given cell at a given time. The set of receptors and the density and location of each receptor on the cell surface depend on cell type and on the current state and environment of the cell. The same stimulus will often cause different responses in different cells.

Two important types of cell surface receptors involved in cellular signal transduction are G-Protein coupled receptors and receptor tyrosine kinases [27]. Rhodopsin, the light receptor in the eye, is an example of a G-Protein coupled receptor. The receptor that will most concern this study, the epidermal growth factor (EGF) receptor, is an example of a

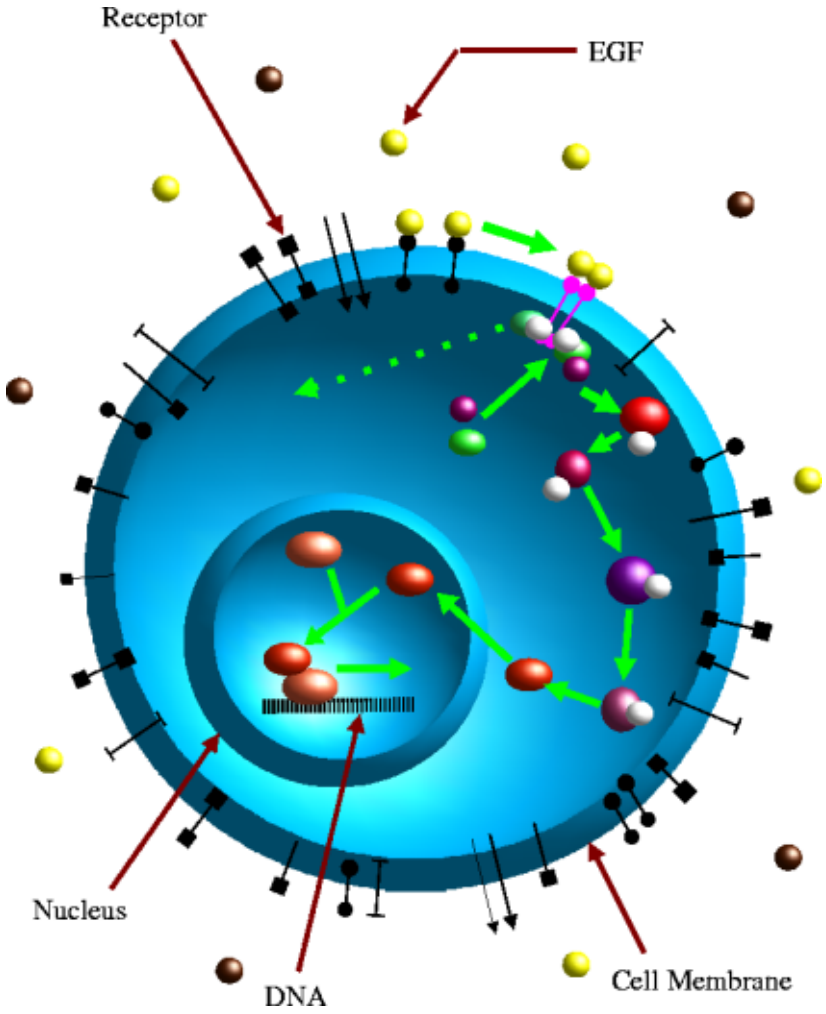

Figure 1.1: A simple picture of the components of the EGF signaling pathway. EGF molecules bind to the extracellular domains of receptors in the cell membrane. The binding causes dimerization of the receptor and subsequent initiation of a series of intracellular responses. The signal is transduced, through a series of phosphorylation reactions, to the cell nucleus. Here gene transcription is initiated by transcription factors.

receptor tyrosine kinase and will be discussed in further detail below. All receptor tyrosine kinases have certain structural features in common: an extracellular domain for binding polypeptide ligands, a single transmembrane region (in contrast to the seven transmembrane structure of a G-protein coupled receptor), a juxtamembrane domain possibly involved in regulating receptor function, and a cytoplasmic domain with tyrosine kinase catalytic regions [27].

All the structural details of a receptor are essential to the specific response and propagation of a set of signals. The sequences of the extracellular domain and of the intracellular catalytic regions determine what molecules can bind to the receptor. Thus, determining the structure and amino acid sequence of a protein is important in elucidating that protein's possible partners in a signal transduction pathway.

Propagation of the signal in the cell

The main method of signal transduction occurs through structural changes of pathway components. A given protein will affect the conformation of one or several other proteins, activating or inhibiting those proteins and thus propagating the signal down the pathway [27]. The trigger for signal propagation often occurs with the binding of the signaling molecule to the receptor, which causes a conformational change in the receptor. Subsequently, cytosolic regions of the receptor are activated, making them active targets for intracellular and membrane associated proteins.

Within the cell, signal propagation depends heavily on the action of protein kinases and protein phosphatases. The discovery of the role of kinases in the propagation of cellular signals was made during experiments studying the reaction of skeletal muscle to epinephrine and involved cyclic adenosine monophosphate (cAMP), an important second messenger for G-Protein coupled signaling [3]. Protein kinases and phosphatases catalyze the transfer of a phosphate group of adenosine triphosphate (ATP) to and from target proteins respectively. The transfer of the phosphate group occurs only at specific amino acids of the target protein. Tyrosine protein kinases catalyze the phosphorylation of tyrosine residues, while serine/threonine protein kinases catalyze the phosphorylation of serine or threonine residues. Some protein kinases (such as MAPK, which will be discussed below) can act as both tyrosine and serine/threonine kinases [3]. Protein kinases and phosphatases are useful signaling molecules because they affect rapid and reversible changes [27].

Most of the intracellular portion of the signaling pathway is a cascade of protein phosphorylations and dephosphorylations. Each step leads to activation or inhibition of further, downstream events or feeds back on upstream events. The traditional view of signal transduction has been as a linear sequence of phosphorylation events proceeding from the cell surface to the ultimate intracellular target. However, it has become increasingly clear that the propagation of signals in the cell is not a simple chain of events, but a complex and combinatorial process involving switches, integrating centres, feedback loops and crosstalk between pathways [29].

Responses initiated by signaling

The responses to signaling can include activation of enzyme activity, changes in cytoskeleton organization, changes in ion permeability, activation of DNA and/or RNA synthesis and many other aspects of cell function [3]. Through such changes, signaling pathways can control cellular function such as growth, maturation, proliferation, and differentiation.

These vital functions suggest the importance of studying signal transduction pathways. It is not a coincidence that many of the components of such pathways were first discovered as oncogenes: abnormal versions of a protein that are implicated in transforming a normal cell into a tumor cell. Cancer is a disease of uncontrolled, abnormal cell proliferation. The regulation of the processes of cell division and apoptosis occurs through signaling pathways. Thus it is in the dynamics and among the components of these pathways that some researchers are looking for causes and cures for this disease.

Regulation of the cell cycle

Let us trace backwards along a set of signaling events from the observed response, such as unchecked cell proliferation, to the entry of the signal into the cell and to the origin of the signal. One set of proteins, called cyclins, act as key regulators of the transitions between phases of the cell cycle by activating cyclin-dependent kinases (CDKs). CDKs phosphorylate proteins needed for each phase transition. The cyclins themselves are regulated by protein complexes involved in protein degradation, such as adenomatous polyposis coli (APC), and the CDKs are regulated by 2 families of CDK-inhibitors [50]. All these proteins are thus important targets of signaling pathways aimed at controlling cell proliferation [33].

Regulation of transcription

There are several levels at which a cell can control the set of proteins expressed at a given time. The most important is transcriptional control, that is, the control of when and how often a gene is transcribed [3]. Other controls exist at the level of RNA processing (such as alternate splicing), RNA transport, RNA translation, RNA degradation, and at the post-translational level such as protein inactivation and compartmentalization [3].

The transcriptional control of gene expression was first investigated for the *lac* operon by Jacob and Monod in the 1950s [23]. It was found that gene expression depended on a region of DNA upstream of the gene and a set of gene regulatory proteins that are capable of binding to that sequence. In bacteria, this sequence of DNA, called the operator, may be subject to negative or positive control through repressor and activator proteins, respectively [3]. In negative control, a repressor protein binds to the operator which prevents RNA polymerase from accessing the promoter, a process needed to initiate transcription. Positive control occurs when an activator protein binds to a DNA site near the promoter to allow RNA polymerase to start transcription. Sets of these repressors and activators exert combinatorial control over the transcription of a bacterial gene.

In eukaryotes, the process of regulating transcription is still more complicated. A set of proteins known as transcription factors must bind to the DNA before RNA polymerase can begin transcription. As well, activators and repressors in eukaryotes are capable of influencing transcription from thousands of nucleotide base pairs away from the start site of transcription. These regulatory proteins also usually act as complexes, the formation of which sometimes depends on the correct proteins arriving in the proximity of a specific DNA sequence [3]. Given this level of complexity and the number of possible combinations of all these molecules, it is not surprising that the mechanism of transcriptional control for most eukaryotic genes is not well understood.

There has been some progress in deciphering the mechanism of control of the cyclins and associated proteins outlined above. Transcription factors from the Fos and Jun families form complexes known as AP-1 which regulate, among other things, the transcription of other transcription factors such as myc [50], which turns on the transcription of the genes that encode the regulators of the cell cycle transitions mentioned above. So, the next question is, what controls the transcription of these genes?

At this point, there are two major pathway types implicated in mitogenic regulation: one involves Ras proteins (Ras, Rho, cdc42, etc) and proteins aiding their signaling to each other; the second involves the mitogen activated protein kinase (MAPK) cascade [50]. These two families are important players in transducing the signal as it travels from the cell membrane to the target molecules. Thus, it is important to determine the interaction of these proteins with other potential pathway members to fully map out the network of intracellular interactions.

Other points of interest in pathways regulating cell proliferation are the signal that initiates the cellular response and the receptor with which it interacts at the cell surface. The protein hormones known as growth factors, which include EGF, platelet-derived growth factor (PDGF), fibroblast growth factor, and insulin, control growth, division, and maturation of cells [3].

In the search for the mechanism of the development of a disease such as cancer, one needs to consider the roles played by all the parts of such a network, from the signaling molecules through the receptors, enzymes and intracellular components, cell-contact components, down to the regulatory machinery for transcription. It is indeed a complex problem.

1.2 Epidermal growth factor signaling

In order to experiment with some of the different computational techniques in this thesis, a biological case study was needed, along with the ability to gather the specific data required for different techniques. This thesis began as an effort to analyse real data about kinase activation patterns. Our partner, Kinetek, was unable to carry out challenging experiments to supply the needed data. Therefore, we decided to produce data using a computer simulation. In 1999, Kholodenko *et al* published a quantitative study of the dynamics of the early events of the EGF signaling pathway, including estimates of rate constants for all reactions under consideration [29]. Subsequently, Chaudhry implemented the model in a simulation [7] using the Gepasi software. With the components of the early pathway events relatively well defined, reasonable rate constants published, and the ability to create data sets easily, this system is an ideal choice for a case study.

This said, it is acknowledged that the use of simulated data for testing the methods

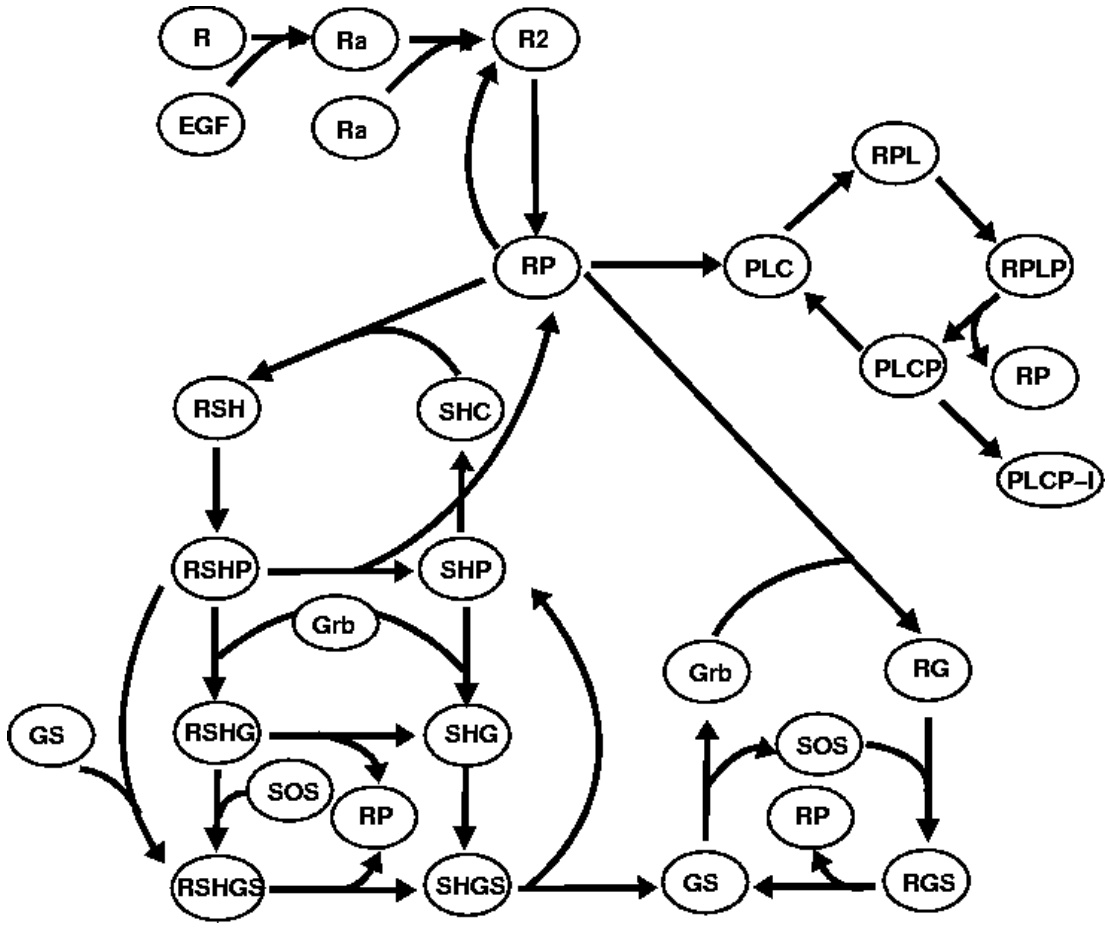

Figure 1.2: The early events of the EGF signaling pathway that are included in the model of Kholodenko *et al* [29]. The proteins corresponding to the abbreviations used in this figure and throughout the thesis are listed in Table 2.5

herein is not the ideal choice. The hope for these methods is that they may prove useful in analysing real experimental results, despite the noise and imperfections inherent in biological assays. These considerations are taken into account when evaluating the different computational approaches described here.

What are the biological features of the EGF pathway? EGF is a polypeptide consisting of 53 amino acids that is known to stimulate the proliferation of many cells *in vitro* and epithelial cells *in vivo* [27]. In addition to proliferation, the EGF pathway is implicated in the control of cell differentiation, growth, and survival.

The EGF signaling pathway begins with the binding of EGF to a single EGF receptor (EGFR). The EGFR is, as mentioned above, a member of the family of receptor

tyrosine kinases and binds several signaling molecules related to EGF [29]. The binding of EGF causes the EGF-EGFR complex to dimerize, which precipitates the autophosphorylation of intracellular tyrosine residues of the receptor [27]. This phosphorylation attracts cytoplasmic proteins containing specific sequences of amino acids called binding domains. Important binding domains include the Src Homology 2 (SH2) domain and phosphotyrosine binding domains [29].

Three of the proteins that bind to the activated receptor are growth factor-binding protein 2 (Grb2), the Src homology and collagen domain protein (Shc), and phospholipase C-γ (PLCγ) [29]. These initial binding events precipitate the transduction of the signal along several paths to initiate different responses from the cell. An important feature of the pathway from the perspective of cell division is the activation of membrane-bound Ras by a complex of Grb2 and son of sevenless (SOS) protein formed following the binding of Grb2 to the activated EGFR. This activation occurs in the form of the catalysis of the GTP-ase activity of Ras (converting GDP to GTP). This action causes the recruitment of the Raf protein and the subsequent initiation of the MAPK cascade [27].

1.3 Modeling the EGF pathway

The simulation of the early events of the EGF pathway includes the following events:

- EGF binding, EGFR-EGF dimerization and phosphorylation
- recruitment of Grb2, PLCγ, and Shc
- Grb2-SOS complex formation and dissociation
- Shc phosphorylation and recruitment of Grb2 and SOS separately to the Shc-EGFR complex and dissociation of the Shc-Grb2-SOS complex from the receptor
- phosphorylation at two tyrosine residues of PLCγ and its translocation to cytoskeletal or membrane structures

These events are shown in Figure 1.2.

Note that all reactions in the pathway are reversible, more or less favorably. The final three points above describe three coupled reaction cycles. The model considers only a limited number of the potential cytoplasmic signaling targets, and does not take into

account the possibility of competition among molecules for phosphorylation sites or adaptor or scaffolding proteins that may be involved [41]. (This is a simplification acknowledged by the authors. However, the fact that many such details are not yet confirmed for comparison, is also a valid consideration for this initial model.)

I simulated these events with the software Gepasi [36], to create artificial data for the "reverse engineering" process. Gepasi provides a convenient interface for numerically solving systems of differential equations describing a set of chemical reactions. The simulation is created by entering chemical equations into the program and specifying the kinetics and corresponding kinetic constants for each equation. For example, the first step of the pathway is the binding of EGF to EGFR, which is entered as

$$EGF + R \rightleftharpoons R_a$$

where R represents the receptor and R_a represents the activated EGF-receptor complex. Similarly, phosphorylated and nonphosphorylated forms of proteins are represented as separate quantities.

All kinetics used for the simulation are generalized mass action equations, with the exception of the phosphorylation events, which are specified using Michaelis-Menten kinetics. The rate equation for EGF corresponding to the chemical equation above is:

$$\frac{d[EGF]}{dt} = -k_1[R][EGF] + k_{-1}[R_a]$$

where k_1 and k_{-1} are the forwards and backwards rate constants for the reaction, respectively, and $[A]$ represents the concentration of protein A.

In total the simplified EGF model consists of 22 differential equations that describe the rate of change in time of the concentrations of the 22 protein conformations due to the 25 reactions between them. It should be noted that the only kinase included in the simulation is the receptor tyrosine kinase, with the action of other kinases and phosphatases being implicitly included in rate constants between phosphorylation states of proteins. In future, it would be desirable to include other kinases and phosphatases explicitly, since parts of the pathway further downstream, specifically the MAPK cascade, are composed only of these proteins.

With the Gepasi simulation, it is possible to generate concentration data for each of the protein conformations at any desired time points. In reality, obtaining this sort of information at a suitable resolution for quantitative studies is a daunting experimental challenge. In order to maintain some connection to real experimental results, I have chosen time scales according to the time scales used in assaying the real biological system, as published by Kholodenko [29]. Given this resolution, it should be noted that the time scales for some reactions in a signaling system such as the EGF pathway can be too fast to be adequately measured. That is, a series of reactions may occur between time steps, which would make the determination of the order of events very difficult using the computational methods presented below.

1.4 The mathematical study of signal transduction

There is an ever increasing amount of biological data being produced from research labs around the world. The problems of how to store, access, annotate and interpret this data are each major research problems in their own right. Ironically, for the purpose of simulations and reverse engineering of biological systems, there is usually not enough *useful* data. This outlines the importance of collaboration between biologists, mathematicians, and statisticians: the latter two to help analyse and interpret the data and to suggest new experiments that will allow better results to be obtained by the biologist and subsequently allow better models to be created.

In addressing the problem of modeling biochemical reactions, one can consider two different approaches: forward and reverse. Forward modeling implies the construction of models from collected biological information about the structure of the system, which can then be used to study the dynamics of the system and to predict future behavior. The reverse problem involves working from a set of data about a system, but without detailed knowledge about the structure of the system being modeled. The goal with the reverse problem is to find out as much information as possible about the structure and the dynamics of the system in question from observations of the system. It is this reverse methodology that is addressed in this thesis.

A second key difference between modeling methodologies is that between a logical/binary model and a kinetic model with differential equations. Further, whether time

and molecule interactions are modeled as discrete or continuous. These three considerations are important in judging how representative of the underlying biological system and how computationally expensive a given model will be.

Important early mathematical research in biological regulation was done by Glass, Kauffman, and Savageau in the early 1970's. Glass and Kauffman [15] investigated the dynamics of discrete systems of two- and four-"component" binary networks to compare with biological results of the previous decade indicating the existence of biological switches and feedback loops. Other important studies modeling metabolic pathways include those of Palsson and Lightfoot (1984) on metabolic networks [40] and of Shea and Ackers (1985) on the λ bacteriophage [48].

As for the reverse engineering of biochemical networks, many of the important works in this newer field will be reviewed in detail in subsequent chapters. However, the work in 1995 by Reinitz and Sharp [45] deserves a mention here as one of the early introductions of the reverse methodology (being applied at the time to electrical systems) to the world of biology. This publication, based on work by the same authors in 1991 with Mjolsness [38], introduces the "connectionist" model of gene regulation: in this model, a genetic network is represented by a matrix whose values represent the "connection strength" between genes, i.e. the amount that one gene influences the expression of another gene. This matrix is then incorporated into a set of differential equations that model the change in concentration of each protein resulting from the change in gene expression. The parameters are fit to experimental data. In other words, this paper starts with a general framework that permits all connections between genes and the connections are then defined such that the best fit to the data is obtained.

In addition to both the forward and reverse modeling of biochemical data, many attempts have been made to find key features in the data that may lead to further knowledge of the system. For instance, clustering and other multivariate analysis techniques have been used to reduce the dimensionality of microarray data and to pick out genes encoding proteins with similar functionality ([18], [4], [12]) . These methods can be coupled with reverse engineering in an attempt to reduce the number of different types of interactions that need to be defined. The question of whether biological systems do in fact use some sort of modularity is still an open problem.

The application of multivariate analysis and reverse modeling to the study of signal

transduction pathways is interesting for several reasons. First of all, despite the technological advances described earlier, these systems are still not well understood. Not only are the specific interactions between molecules not all determined, but the molecular components themselves are not all known. Obviously then, chemical kinetic data is not available for the majority of the reactions involved in signal transduction pathways. These facts preclude the use of complete models and simulations for most pathways.

This thesis is organized as follows: in Chapter 2 an overview of multivariate analysis and its application to the study of genetic regulation are covered; in Chapter 3 several approaches to reverse engineering are discussed; Chapter 4 describes the implementation of a two-step reverse engineering method to study the early events of the EGF pathway; Chapter 5 consists of a discussion and concluding remarks.

Chapter 2

Multivariate Analysis

Multivariate analysis is the qualitative and quantitative study of populations that depend on many variables. Given that most natural phenomena depend on more than one variable, the development of such techniques is an ongoing process that touches many fields. Multivariate analysis is specifically concerned with mathematical approaches to studying data sets consisting of a set of observations on a set of variables.

Before looking at several multivariate analysis techniques below, the organization of the data set and a few introductory statistical concepts are described.

2.1 Introduction to multivariate covariance and correlation

Throughout this chapter I shall denote the i^{th} variable observed on the j^{th} item or trial as x_{ij}. For example, the variables in a study might be height, weight, and age and the items or trials might be the different individuals in the study. So, x_{13} would be the height of the third person in the study.

Organize a set of n trials measuring p variables in the $(p \times n)$ matrix, X. Since one is most likely interested in statistics for the variables over the set of individuals (say average height), the data is thought of as a set of p n-dimensional vectors. That is, $x_i^t = [x_{i1}, \ldots, x_{in}]$ and so $X^t = [x_1, \ldots, x_p]$, where t denotes the transpose and x_j is a vector containing values of one of the variables for all n trials.

Given this organization, one can define:

- The mean, taken to be the mean of each variable over all the trials, is $\bar{X}^t = [\bar{x_1}, \bar{x_2}, \ldots, \bar{x_p}]$
- Sample covariance (measure of linear association between pairs of variables i and k)

given by the matrix $\mathrm{Cov}(X) = S$ with elements

$$s_{ik} = \frac{1}{n-1}\sum_{j=1}^{n}(x_{ij} - \bar{x_i})(x_{kj} - \bar{x_k}),\ i = 1, \ldots, p, k = 1, \ldots, p \tag{2.1}$$

A negative covariance implies that most trials with large values for variable i have small values for variable k or vice versa. A positive covariance implies that either large or small values are obtained for both variables, and a zero covariance implies that there is no particular association between the variables over the course of the trials. Given the covariance matrix S with elements s_{ij}, the generalized sample variance is given by $\det(S)$.

- A related concept is Pearson's correlation coefficients. Here the covariance is normalized by the standard deviations of the two variables, thus down-playing the impact of the range and units of the measurements [25]. The correlation takes on values between negative and positive one inclusive. These values are given by:

$$r_{ik} = \frac{s_{ik}}{\sqrt{s_{ii}}\sqrt{s_{kk}}} \tag{2.2}$$

- Given a system of q linear combinations of p random variables

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_q \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1p} \\ a_{21} & a_{22} & \ldots & a_{2p} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ a_{q1} & a_{q2} & \ldots & a_{qp} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_p \end{bmatrix} = AX \tag{2.3}$$

where a_{ij} are scalar constants, the variance of the i^{th} linear combination is

$$\mathrm{Var}(y_i) = \mathrm{Var}(a_i^t x) = a_i^t \mathrm{Var}(x_i) a_i. \tag{2.4}$$

It is important to note that the quantities above deal with linear relationships in the data. As was mentioned in Chapter 1 gene and protein interactions in signal transduction and genetic regulation can be nonlinear. Thus some relationships between variables may be

misconstrued using these linear statistical measures.

2.2 Principal Component Analysis

The goal of principal component analysis (PCA) is to reduce dimensionality and aid in the interpretation of a large data set. This goal is accomplished by finding a smaller set of basis vectors that still describe the data well (i.e. that capture major sources of variance over the trials in the study). For this reason, PCA is also often applied to a large data set before further processing using regression or clustering techniques.

There are several ways in which one may try to understand the role of the principal components. They are linear combinations of the original variables, $[x_1, \ldots, x_p]$. Thus they can be thought of as a new set of axes that have been rotated from the original axes to fit the data better (in particular, that one can find the important relationships between variables by examining a smaller number of dimensions).

One can also think of the principal components as the decomposition of the original signal into a set of distinct patterns over the set of trials or time points that can be recombined to recreate the original data. Thus one can examine the properties of the most prevalent patterns in the data, both over the set of experiments or the time course and in comparison to each other.

Before proceeding with the application of PCA, the problem must be formulated. One must distinguish between the "trials" and the "variables" for the study. Determining the principal components will allow us to examine the influence of a given variable on the most influential components and also to visualize each variable's position in the reduced component space relative to other variables.

If one uses PCA to study large-scale gene expression data, it is important to first determine the aim of the study. Are we interested in comparing different arrays or different genes? Another important consideration is the dimension of the data matrix: in gene expression studies, one will usually have a much larger number of genes than arrays. This means that a choice to use the genes as variables means a far greater computational cost, as can be seen from the dimensions of the covariance and correlation matrices listed in Section 2.1. Despite the costs, I believe that it is most useful to set the genes as the variables and the trials as the arrays.

2.2.1 Mathematical background for PCA

This section serves as a brief, intuitive introduction to principal component analysis. As stated in Section 2.1, the data set is organized into a $(p \times n)$ matrix, where p is the number of variables (genes), and n the number of trials (arrays) over which the value of each variable is measured. In looking for the principal components of the data, one is looking for p uncorrelated, linear combinations of the vectors $x_1, \ldots, x_p$ with the maximum variance [26]. That is, given the data matrix

$$X^t = [x_1, \ldots, x_p], \tag{2.5}$$

one wishes to find the p-dimensional vectors a_i^t that result in the p principal components

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{21}x_2 + \cdots + a_{p1}x_p = a_1^t X \\ y_2 &= a_{12}x_1 + a_{22}x_2 + \cdots + a_{p2}x_p = a_2^t X \\ &\vdots \quad \vdots \quad \vdots \qquad \vdots \quad \vdots \\ y_p &= a_{1p}x_1 + a_{2p}x_2 + \cdots + a_{pp}x_p = a_p^t X \end{aligned} \tag{2.6}$$

whose variances are maximized.

Let the covariance matrix of the variables be the $(p \times p)$ matrix $\mathrm{Cov}(X) = S$. From Equation (2.4), the variance of the i^{th} principal component is given by $Var(y_i) = a_i^t S a_i$. Since the variance can always be increased by increasing a_i, the condition $a_i' a_i = 1$ is imposed. So, to find the principal components one must solve the problem:

$$\textbf{Maximize } a_i^t S a_i \textbf{ subject to } a_i^t a_i = 1 \tag{2.7}$$

The details of the maximization are included in Appendix A. The results are that the coefficients of the principal components are given by the eigenvectors of the covariance matrix, that is

$$\begin{aligned} y_1 &= e_{11}x_1 + e_{21}x_2 + \cdots + e_{p1}x_p = e_1^t X \\ y_2 &= e_{12}x_1 + e_{22}x_2 + \cdots + e_{p2}x_p = e_2^t X \\ &\vdots \quad \vdots \quad \vdots \qquad \vdots \quad \vdots \\ y_p &= e_{1p}x_1 + e_{2p}x_2 + \cdots + e_{pp}x_p = e_p^t X \end{aligned} \tag{2.8}$$

where e_i is the eigenvector corresponding to the i^{th} largest eigenvalue, λ_i, of the covariance matrix, S. The spectral decomposition of S is:

$$S = \sum_{i=1}^{p} \lambda_i e_i e_i^t \Rightarrow e_i^t S e_i = \lambda_i \tag{2.9}$$

From Equation (2.4) then, the variance of the i^{th} principal component is

$$Var(y_i) = Var(e_i^t S e_i) = \lambda_i \tag{2.10}$$

It follows [25] that the total variance is:

$$\sum_{i=1}^{p} Var(X_i) = \lambda_1 + \lambda_2 + \cdots + \lambda_p = \sum_{i=1}^{p} Var(Y_i). \tag{2.11}$$

Thus one can compute the proportion of the total variance of the data that is captured by the i^{th} principal component:

$$\text{proportion of variance} = \frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i} \tag{2.12}$$

At this point, we have a method of computing the principal components and of assigning each component a rank in terms of explaining the variance in the data. Examining the coefficients of the principal components, that is, the eigenvectors of the covariance matrix, the correlation between the k^{th} variable and the i^{th} principal component is computed as:

$$\rho_{y_i,x_k} = \frac{e_{ki}\sqrt{\lambda_i}}{\sqrt{\sigma_{kk}}}. \tag{2.13}$$

This provides us with a manner of quantifying the importance of the k^{th} variable to the i^{th} principal component.

The results presented in this section can also be obtained using the matrix of correlation coefficients rather than the covariance. This is equivalent to performing the analysis with the covariance matrix after having normalized the variables by removing the mean and dividing by the standard deviation of each variable. Note, however, that the principal components and the corresponding variances will differ depending on whether the covariance or correlation matrix is used. It is recommended that the correlation matrix be used

if the variables are measured on very different scales [25]. In Section 2.4 I compare the use of the covariance and the correlation matrices in studying the EGF pathway data.

2.2.2 Singular Value Decomposition

The Singular Value Decomposition (SVD) theorem for matrices provides a powerful method for finding the eigenvalues and eigenvectors of a matrix and is useful for dealing with certain types of systems of equations [44]. MATLAB has functions for performing the SVD. The results of the decomposition are as follows: Any $(m \times n)$ matrix A, where $m \geq n$, can be written as

$$\begin{array}{ccccccc} A & = & U & \cdot & W & \cdot & V^t \\ (m \times n) & & (m \times n) & & (n \times n) & & (n \times n) \end{array} \tag{2.14}$$

where

- U is column orthogonal and whose columns are given by $U_i = (\frac{1}{w_{ii}})AV_i$
- W is a diagonal matrix with $w_{ii} \geq 0$ called the singular values.
- V is an orthogonal matrix whose columns, V_i, are the eigenvectors of A^tA.

The decomposition can also be done on a matrix with $m < n$, but in this case $w_{ii} = 0$ for $i = m + 1, \ldots, n$ and the corresponding columns of U are zero.

2.2.3 Recent applications to studies of gene expression

Two studies published in 2000, one by Holter *et al* and the second by Alter, Brown and Botstein, perform a principal component analysis of sets of gene expression data. Both papers apply the technique to previously published yeast (*Saccharomyces cerevisiae*) gene expression data [49] and Holter *et al* also use data from human fibroblasts (cells found in connective tissues) [21].

The Alter paper is complicated and suffers from unfortunate typographical errors that make the interpretation of the method difficult. The authors coin the terms "eigengenes" and "eigenarrays" to represent the variables of the new "reduced space". The measurements in this new space are organized in a diagonalized matrix whose diagonal elements are the "eigenexpression" of the i^{th} "eigengene" in the i^{th} "eigenarray", all other values being zero.

Some of my confusion with both the Alter and the Holter paper arises from their use of the Singular Value Decomposition theorem to find the eigenvalues. The advantage of using singular value decomposition is that the principal components of both the variables and the trials can be obtained easily and without having to calculate the covariance matrices. The confusion is that in the reported papers the singular values of the *arrays* are reported rather that those of the genes, while the subsequent analysis uses the principal components of the *genes*. This underemphasises the most useful aspect of principal component analysis, which is the reduction of the size of the data set. A reduction from 15 to 5 variables is not nearly as impressive as a reduction from 676 to 5.

Despite these problems, there are some interesting ideas presented. One is that the elimination of modes or components can be used to eliminate patterns in the data that may stem from noise or experimental artifacts. The key to this idea is that one need not eliminate variables or trials to be able to concentrate on the important features of the data. Rather, one can dissect the entire data set into modes, pick out those that are of interest, and reject others that are not. This represents an alternative to the common practice in large-scale gene expression analysis of choosing only a subset of genes whose level of expression exceeds some (fixed) multiple of normal expression level.

Another idea suggested by Alter that seems promising is to compare the expression patterns of principal components from two sets of microarray experiments (say two time courses or two sets of repeated experiments). In one set of experiments a key gene, such as a regulator, is overexpressed, while the other set has normal or low activity. The patterns can then be compared for differences, and the genes that are important to (have a high correlation with) the components that differ can be singled out. Note that a similar multivariate analysis technique, canonical correlation analysis, might also be suited to such an analysis (for an introduction to this method see [25]).

In the Holter paper, a key finding is that the principal component analysis suggests some temporal connections between previously determined clusters of the genes. A plot of the coefficients of the first two principal components against each other (for each gene) shows that clustered genes appear grouped together around the perimeter of an ellipse. The fact that most of the genes appear on an ellipse is a property of principal component analysis. (From Equation[2.8], a multivariate normal distribution yields a set of level curves in the form of ellipses of constant density with axes given by the principal components [25]). The

fact that the genes are grouped together is reassuring. Further, the fact that the order that the groups appear around the circle mirrors the order of the genes' temporal progression in the cell cycle is very interesting. It would be good to examine whether this behavior is repeated for other well understood systems. If so, then PCA might be used to order clusters in less well understood systems.

2.3 Clustering

Clustering is a process of grouping variables based on their similarity over trials or grouping trials based on their similarity over variables. For instance, in a study of gene expression, genes that show a similar pattern of expression over all the arrays are grouped together in a cluster. On the other hand, one could perform a clustering that groups arrays according to the similarity of the pattern of expression of all the genes in the array.

There are several important questions that one should address with respect to applying a clustering technique to gene expression or protein concentration data:

- What sort of patterns can be found, i.e. how are genes or proteins within a cluster related and how do they differ from elements outside that cluster?
- Are these repeatable, robust methods? Are they intended to and do they give a concrete statistical measure of the success or failure of a given grouping?
- What clustering method and what measure of similarity are best for understanding relationships between genes or proteins in a network?

I will address some of these issues in this section and present a general overview of several important clustering methods.

2.3.1 Measuring distance or similarity

An important issue when trying to find similarities between variables or trials in the data, is answering the question: what exactly does "similar" mean? Results of any comparison would be defined by the answer to this question. For example, perhaps one wants to divide a selection of tree species into different categories. There are many possible divisions: for instance, the grouping could be made on the basis of average height, whether the trees are coniferous or deciduous, geographic location, average lifespan, the common uses of the

wood, or some combination of these factors. Thus the choice of a measure of similarity must be based upon knowledge of the type of the grouping that one wishes to achieve.

Similarity of two items can be quantified by defining a "measure of distance" (or distance metric) between the two items. There are several different distance metrics that are common in practice and they are described below. A distance measure between two points x and y should satisfy the following criteria [25]:

- $d(x, y) = d(y, x)$
- $d(x, y) > 0$ if $x \neq y$
- $d(x, y) = 0$ if $x = y$
- $d(x, y) \leq d(x, z) + d(z, y)$

The Euclidean distance is the most common measure of distance. The Euclidean distance between two points x and y is given by

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2} \tag{2.15}$$

A statistical drawback to the Euclidean measure is that all the differences between components are equally weighted. If some variables naturally have a wider range, they will tend to dominate the metric. This is a concern for gene expression values, as a small change in the expression of some genes may be significant. An option in this case is to standardize the variables by dividing measurements by the standard deviation if the Euclidean metric is to be used.

A common measure of similarity is the dot product or correlation coefficient described in Section 2.1. Other similarity measures include the Minkowski Metric and comparison by operations of continuous-valued logic [31]. For comparing strings of characters, rather than numerical data, Hamming or Levenshtein distances can be used (these are important, for instance, in comparison of gene or protein sequences). Mutual Information is a similarity measure based on Shannon entropy (see Section 3.1.1) that has been used for clustering gene expression data [11].

Alternatively, one can compare pairs of items based on the presence or absence of certain characteristics, with similar items having more characteristics in common than

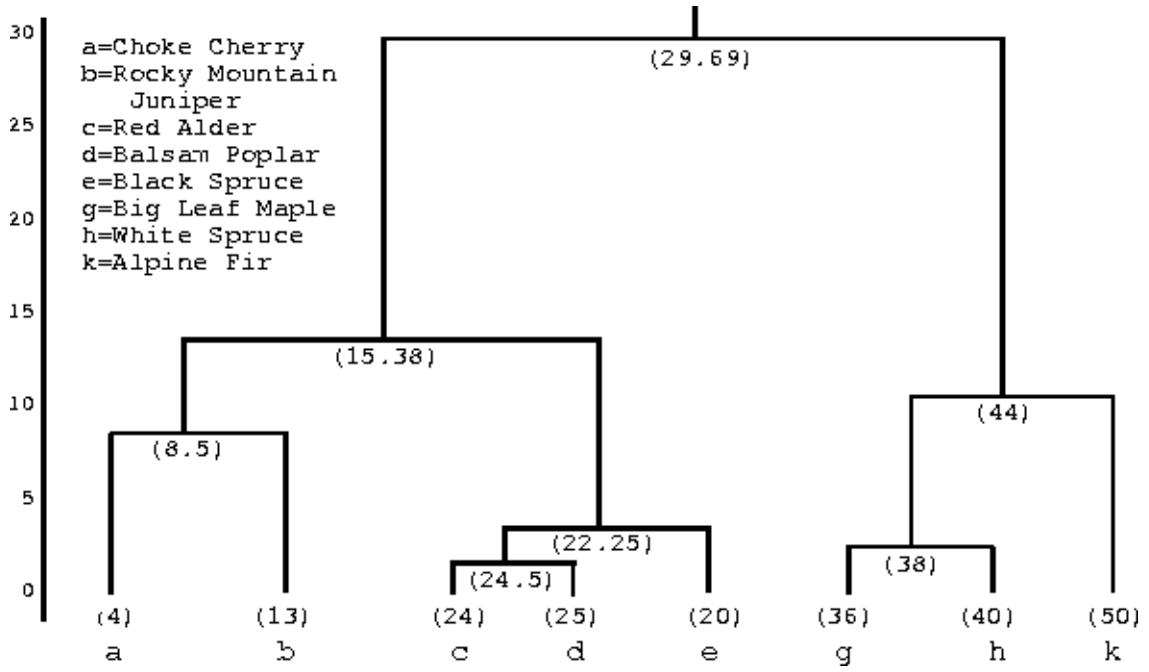

Figure 2.1: A tree visualization of the results of the hierarchical clustering of a group of British Columbian trees based on height. The scale on the left represents the distance between the two clusters being joined by a given branch. The figures that appear under the tree names along the bottom are the heights of each tree, and the numbers in the graph represent the average tree heights of each newly formed cluster. This is an example of agglomerative hierarchical clustering.(Tree data from www.bcadventure.com)

dissimilar items. For example, one could define a set of binary characteristics such as whether a given gene's expression is above or below a threshold for a given array. The similarity score could then be computed using a weighted sum of the matches between pairs over the characteristics.

All in all, for a successful clustering application, the choice of the distance measure for a particular problem should be based on features of the data being compared and the specific qualities that constitute similarities between variables for that application.

2.3.2 Hierarchical clustering

The most common visualization of a hierarchical clustering is a tree such that the similarities of the items or variables being clustered are reflected by branch lengths and relative positions in the tree. For instance, consider Figure 2.1, which shows the hypothetical output of the clustering of British Columbian tree species based on height.

Hierarchical clustering can be agglomerative or divisive. In the former, each item to be clustered begins as its own cluster. The two most similar clusters are then joined so that there is one less cluster than previously. This process is repeated until there remains only one cluster containing all the items. This can be seen in Figure 2.1, where Red Alder and Balsam Poplar join to form a cluster in the first step of agglomeration because they have the most similar tree heights. Divisive hierarchical clustering proceeds in the same fashion but in the reverse direction: all items start in one cluster that divides until each item rests in its own cluster.

The determination of which pair of clusters are closest can be made using linkage methods. Three linkage methods are shown in Figure 2.2. A distance metric must be chosen to measure the distance between items in each pair.

As for the other clustering methods discussed here, hierarchical clustering is sensitive to noisy data and outliers because error and variation are not taken into account [25]. Another drawback, specific to hierarchical clustering, is that a poor clustering choice at a given step during the process cannot be fixed and will affect all subsequent results.

2.3.3 Self-organizing maps (SOMs)

A self organizing map is an unsupervised, competitive neural network used to cluster large data sets into a specified number of groups. The classification "unsupervised" differentiates this method from supervised and reinforced learning methods. A supervised method is more of a classification than a clustering, as the criteria for inclusion in a cluster is predetermined. In unsupervised methods, the classes of items are not known before the clustering is performed, but the data is suspected to contain some natural division. Thus an unsupervised clustering reveals both the form and the contents of each grouping.

There are many different types of artificial neural networks, and I will not go into details about them here. In competitive networks, the neural cells receive input data on which they compete: based on some competition criteria, a winner amongst the cells is chosen for a given input point. The winner attains full activity and suppresses other cells in the network to varying extents. The next input is then introduced, the fight begins anew and a new winner is chosen [31].

Let us consider the details of the SOM itself. A SOM is a mapping from the input data space onto a two-dimensional array of nodes. These nodes represent the clusters to

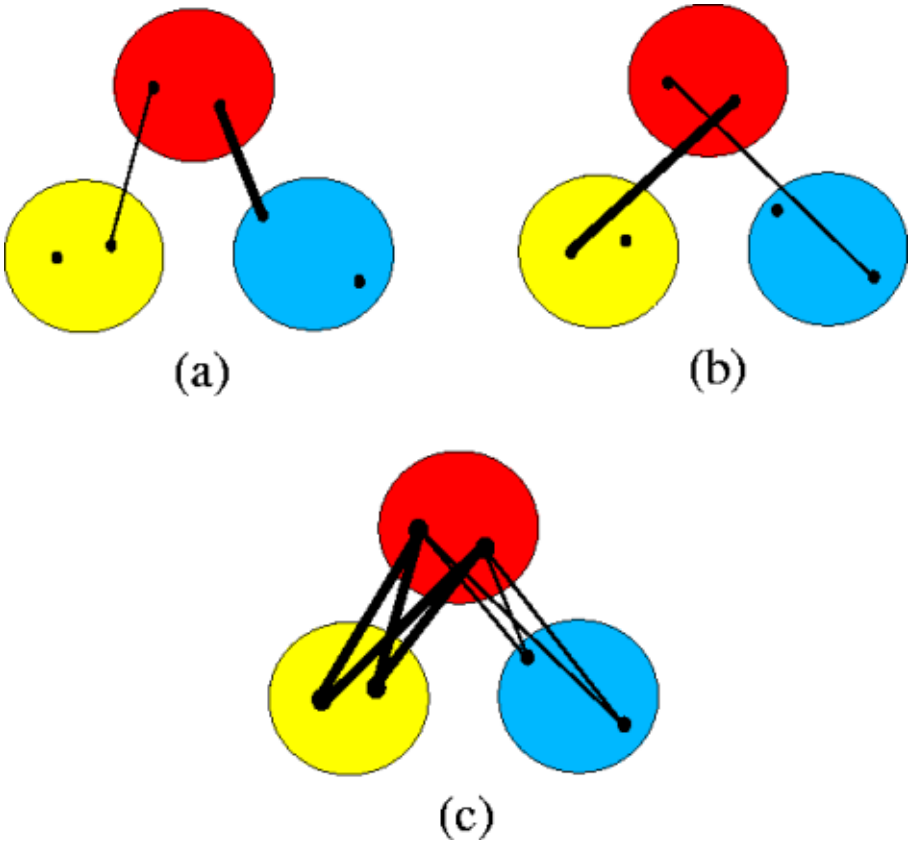

Figure 2.2: An illustration of different linkage methods used to determine the most similar clusters at each step of a hierarchical clustering. The circles represent clusters and the dots inside each circle represent the items contained in each cluster. The darker line(s) indicate the shortest of the compared distances in the pairwise comparison between the red cluster and each cluster beneath. The cluster with the shortest distance to the red cluster is combined with the red cluster for the next step of the clustering. (a) *Single linkage* also known as nearest neighbor. The shortest distance between items in each pair of clusters is compared. (b) *Complete linkage* or farthest neighbor. The largest distance between items in the two clusters are compared to find the shortest of these. (c) *Average linkage*. The average of all the distances between items in the pair of clusters are used for comparison.

be determined. To construct the map, the number of nodes (clusters) is chosen and these nodes are arranged in a lattice. The shape of the lattice may be hexagonal, rectangular, or even irregular. There are also some instances of three-dimensional lattices [30], though the two-dimensional lattice is much more common.

Recall the formulation of a $(p \times n)$ data matrix. Choose to cluster the variables, so each of the p data inputs is an n-dimensional vector. The SOM algorithm proceeds as follows:

1. Initialize the network by assigning each node its own n-dimensional reference vector.
2. Choose an input point. This selection may be done randomly or using a randomly ordered list that is repeated as necessary.
3. Find the closest network node to the input datum. The closeness is calculated using the chosen distance measure.
4. Change the reference vectors of the nodes based on each node's proximity to the closest node.
5. Repeat steps 2 through 4 until the desired number of iterations is reached.

Changing the reference vectors in step 3 constitutes the learning step of the network. As Kohonen suggests, one may think of the SOM as an "elastic net" of nodes that are stretched over the input data points in an orderly way with the goal of approximating their distribution in the input space. That is, the changing of the reference vector represents the stretching of the nodes toward the current input point. This is demonstrated in Figure 2.3. The learning function has the form

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)], \qquad i = 1, \ldots, p \tag{2.16}$$

where t is an integer representing the current iteration number and $h_{ci}(t)$ is a "smoothing kernel" defined over the nodes. The smoothing kernel determines how much each node moves at each step, t. Thus $h_{ci}(t)$ must tend to zero as $t \to \infty$ so that the nodes eventually settle to a given point. One example of a smoothing kernel is as follows: define a neighborhood, N_C around the node, C, that is closest to the current data point. Then $h_{ci}(t)$ is defined so that the nodes falling within the neighborhood are moved a distance given by a learning

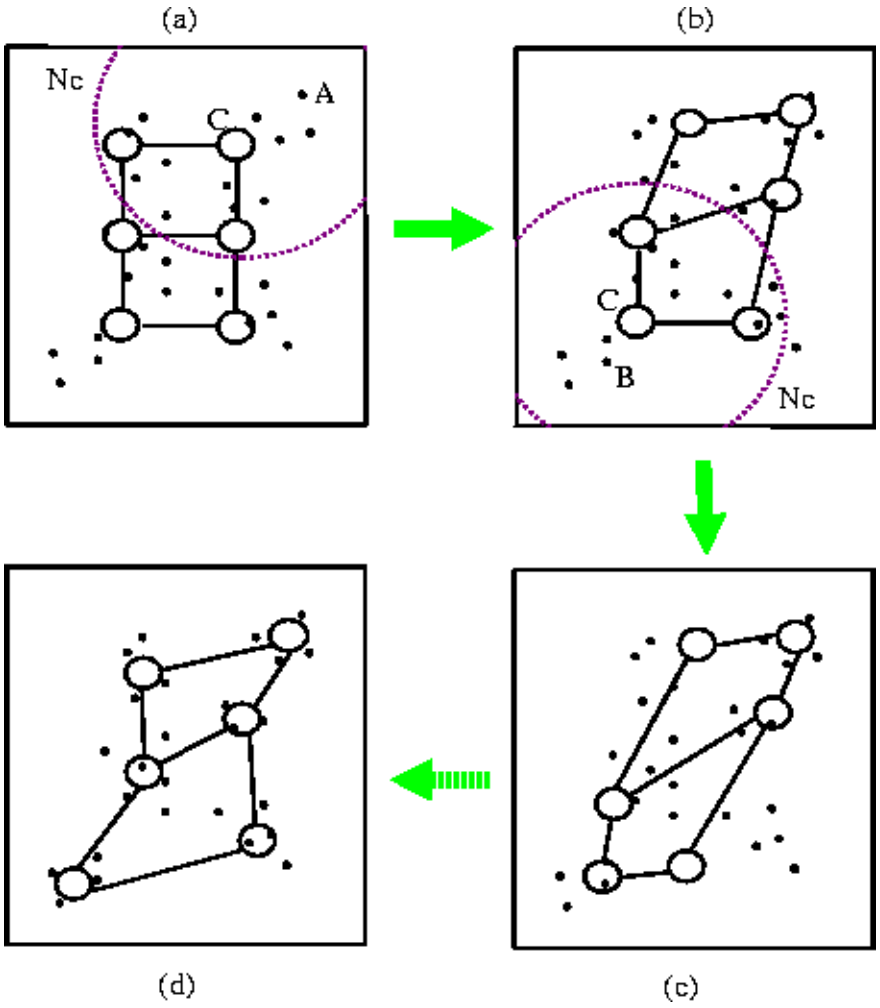

Figure 2.3: The training of a SOM. The small dots represent data points. The larger circles represent the nodes of the network arranged in a (3×2) rectangular lattice and located at the position in the data space given by their reference vectors. (a) Data point A is chosen randomly. Node C is the closest node to A. The dotted circle, N_C represents the neighborhood of node C. (b) Node C and the other two nodes that lie within N_C are moved toward A as dictated by the learning function. The next data point, B, is randomly chosen and the new closest node C and its neighborhood N_C are shown. (c) Node C and the two nodes lying within N_C are moved toward B. (d) After many such iterations, the nodes settle at clusters of data points.

rate, while those that lie outside of N_C remain stationary. For instance, one could define

$$h_{ci}(t) = \begin{cases} \alpha(t) & \text{if } i \in N_C \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

where $\alpha(t)$ is a learning rate factor that takes values between zero and one and decreases in time [31]. A modification of this learning method is to have the neighborhood size decrease with t as well, so that toward the end of learning, the adjustments are made primarily to the node of the cluster to which the current datum belongs [51].

From the discussion above it is apparent that, although the form of the clusters is not specified, there are many decisions to be made in constructing a SOM. These decisions include the learning rate factor, the smoothing kernel, the number of nodes, the lattice shape, the number of iterations, the neighborhood size, and the distance metric. In addition to these factors, any statistical processing of the data before the creation of the SOM begins will affect the results. This is true for all of the multivariate analysis techniques presented in this chapter.

The careful selection of parameters such as the learning factor and the form of the smoothing kernel is more important for larger networks (over 100 nodes) [31]. Both the number of clusters and the number of iterations chosen for a SOM are rather arbitrary. Kohonen suggests the following "rule of thumb" for the number of iterations:

$$\text{number of steps} = 500 \times \text{number of network units} \tag{2.18}$$

The choice of lattice shape appears to be based mainly on ease of interpretation and best fit to the range of the data points. The number of nodes in the network seems to be the most difficult choice to make and one which has a strong impact on the results. Some preliminary research into the number of groups one expects to find in the data and some trial-and-error are needed to determine a suitable network size.

2.3.4 Recent applications of clustering to studies of gene expression

Many clustering studies of gene expression have been carried out in the last five years by groups at Stanford University led by Pat Brown and David Botstein. The majority of these studies have been aimed at identifying patterns in the gene expression of various

cell types and cellular states (for example, tumor vs. normal cells). Examples include the investigation of transcriptional programs of sporulation [8] and responses to stimulants [21]. The website for the Pat Brown laboratory (http://cmgm.stanford.edu/pbrown/) is a good source for information related to large-scale gene expression.

The study by Eisen (1998) [12] is an early example of using hierarchical clustering to address the recent flood of gene expression data. The aim of this clustering is basically to organize and present a vast quantity of expression data in a format that is both agreeable and useful to a biologist who needs to interpret the data. The Eisen study applies hierarchical clustering with pairwise average-linkage to two sets of data: a single time course and a set of unrelated data from several experiments on the budding yeast, *S. cerevisiae*. This paper is an early example addressing the flood of gene expression data. The most important result of the paper is the conclusion that genes of similar function cluster together. Also, genes with high sequence homology are found to cluster near each other. Altogether, the results indicate that exploring similarity in patterns of gene expression with clustering methods is a reasonable preliminary method to survey functionality in large gene expression data sets such as those obtained with DNA microarrays. This is an important result that has paved the way for a plethora of clustering studies of large-scale gene expression data.

Another point of note about this study is its method of presenting the clustered data. An image of a gene array in its native form is not highly informative without a spreadsheet of data values and, even then, the size of the dataset can be overwhelming. The visual presentation in this study is enhanced by colouring the reordered data points according to the expression reading: genes with no expression for a time point or single experiment are black; positive expression is coloured in red; negative expression is coloured in green. Thus patterns in the data can be more easily identified by inspection of the images.

Pablo Tamayo *et al* (1999) use Self Organizing Maps(SOMs) to study patterns of gene expression in three separate case studies [51]. The first study is of the yeast cell cycle. This data was chosen since it has been well studied using other methods and, thus, is an ideal system on which to test the ability of SOMs to identify patterns. The yeast data follows gene-expression at 10-minute intervals over two cell cycles. The second study uses the myeloid leukemia cell line HL-60 and expression monitoring Affymetrix arrays to investigate hematopoietic differentiation using gene expression data for over 6000 genes. The data is a time series of samples collected at 0, 0.5, 4, and 24 hours following the

beginning of macrophage differentiation. The final study is similar to the second study but with increased complexity due to the use of multiple cell lines, namely HL-60, U937 (a myeloid cell line), Jurkat (a T cell line), and NB4 (an acute promyelocytic leukemia cell line).

The SOMs were carried out using GeneCluster: a publicly available computer package created by the Whitehead/MIT Center for Genome Research [1]. Genes whose expression does not change sufficiently over the cycles are removed. The data from the remaining genes is normalized to a mean of zero and variance of 1 (within each cycle for the yeast data and within time points for each cell line). The filtered data is then used to create a SOM. The geometry, number of iterations and various other options are set and then the SOM trains and produces a graph of the centroid and a list of the genes in each cluster.

The results of the paper are very encouraging. The yeast cell cycle (6×5) SOM agrees well with the biological conclusions obtained over a much longer time using visual inspection. The results of the SOMs on the human hematopoietic differentiation data are also positive. An analysis of one cluster of a (4×3) HL-60 SOM reveals several genes that would be expected in this cluster in accordance with the graph for the cluster which indicates a gradual induction of the genes over time. However the inclusion of several unexpected genes in this cluster suggests possible roles for these genes in macrophage differentiation [51]. The confirmation of expected patterns and the hints of other unexpected patterns is a very promising sign for the use of SOMs in the development of hypotheses and generation of suggestions for further experimentation.

2.4 Applications of multivariate analysis techniques to published and simulated data

In order to gain a greater understanding of these techniques, I first apply the methods to publicly available gene expression data from the publications noted above. Following the success of these applications, an attempt is made to apply multivariate analysis techniques to the EGF simulation data discussed in Chapter 1. The goal of this second effort is mainly to see whether the encouraging results of the gene expression studies can be extended to the study of protein concentration data. I am interested in what kind of information about a signal transduction pathway could be learned from the application of multivariate

Dataset	Type	Size	Source
gene dataset1	gene expression	5766×14	Spellman elutriation synchronized cell cycle
gene dataset2	gene expression	676×15	Spellman cdc15 data
EGF dataset3	protein concentration	23×9	EGF simulation
EGF dataset4	protein concentration	23×61	EGF simulation
EGF dataset5	protein concentration	23×51	EGF simulation

Table 2.1: Data sets analysed using principal components and self-organizing maps in Section 2.4.

techniques to protein concentration data. The data sets used in the following sections are listed in Table 2.1.

2.4.1 Application of principal component analysis

A series of gene expression studies on the *S. cerevisiae* cell cycle are available from the websites associated with the Pat Brown Lab at Stanford. The data used for the application of PCA discussed here is from 14 arrays of elutriation (size-based) synchronized cell cycle (gene dataset1) and 15 arrays where the cell cycle is synchronized by the arrest of a *cdc15* temperate-sensitive mutant (gene dataset2) [49]. Both data sets are obtained using the same 6179 genes. The sets of arrays are both time courses: gene dataset1 spans from 0 to 390 minutes measured every 30 minutes; gene dataset2 is measured every 20 minutes from 10 to 290 minutes.

Due to the confusion mentioned in Section 2.2.3, I have decided against using the singular value decomposition as is done in the publications mentioned above. Instead, I find the eigenvectors and eigenvalues directly from the covariance or correlation matrix.

Study 1

Using the elutriation synchronized cell cycle data (same data set as Alter, Brown and Botstein), I eliminate the genes from the data set for which some values are missing. In practice, it would be best to develop a method for approximating the missing values or statistically accommodating such an occurrence in the analysis. However, given that the data used for the analysis was reported to have no missing values [4], I chose the above option. The remaining data is organized into a (14×5766) matrix (gene dataset1). The reader may notice that this arrangement is contrary to the earlier statement that the more

covariance		correlation	
eigenvalues	% variance	eigenvalues	% variance
0.6351	39.8765	4.3296	30.9261
0.3852	24.2163	3.6146	25.8188
0.1371	8.6078	1.1458	8.1842
0.0853	5.3555	0.8707	6.2193
0.0733	4.6019	0.7224	5.1601

Table 2.2: Eigenvalues and the proportion of the variance of the entire data set from the first five principal components of gene dataset1

useful setup involves having the genes as the variables, which determines the number of rows in the matrix. However, given that there are 5766 genes, the available computing space on a machine of the Institute of Applied Mathematics lab is less than the approximately 2 Gigabytes [6] necessary to compute the eigenvalues of the (5766×5766) covariance matrix in MATLAB. Thus I do the analysis using the arrays as variables.

I calculate both the covariance and the correlation coefficient matrices, with the aim of discovering whether a large difference in principal components is seen between the two. The eigenvalues of both matrices are listed in Table 2.2. The first 5 eigenvalues of the covariance matrix account for 82.7% of the variance in the data, while the first five eigenvalues of the correlation matrix account for 81.0% of the variance. Thus, there is not a large difference between the two approaches in terms of the proportion of information captured by the largest principal components. The covariance principal components do attribute a larger portion of the variance to the first principal component, however. The fact that a relatively small proportion of the total variance is captured in the first principal components of both approaches is perhaps indicative of the complexity of cell cycle regulation: that there are many different correlations between the genes whose expression is being monitored.

Study 2

The second study yields more interesting results, since gene dataset2 is smaller, allowing us to take the genes as the variables of the system. For this study, I use the *cdc15* arrays and use only the 797 genes identified by Spellman as those involved in cell cycle regulation [49].

Genes with missing expression values are again removed, resulting in a (676×15) data matrix. The covariance and the correlation coefficient matrices are each used to calculate

covariance		correlation	
eigenvalues	% variance	eigenvalues	% variance
100.0816	36.7111	220.1517	32.5668
65.6441	24.0790	156.9503	23.2175
43.5826	15.9866	89.0825	13.1779
17.9742	6.5931	57.7077	8.5366
14.126	5.1819	48.8233	7.2224

Table 2.3: From Study 2: Eigenvalues and the proportion of the variance of the entire data set from the first five out of 676 principal components for gene dataset2

the 676 principal components. For this example, one begins to appreciate the dimensionality reduction ability of PCA. At this point a perusal of the largest and smallest of the 676 principal components alone can be fruitful for studying the system. Note that the eigenvalues from this study differ from those of Holter *et al*, in part because they have reported the eigenvalues using the time points as variables, i.e. they report 14 eigenvalues, rather than 676. Another reason for the difference is that no preprocessing of the data is done here, while Holter *et al* have normalized the data by removing the mean of both the genes and the arrays and setting the standard deviations to 1.

The first five eigenvalues are listed in Table 2.3. The first five principal components using the covariance matrix capture 88.6% of the variance in the data. Figure 2.4 shows the behavior of the first five principal components over the time course of the arrays. Notice the clear cyclical behavior, especially of the first two components. Using Equation[2.13] and consulting the publication listing the genes in the study, I find that the gene that is most correlated with the first principal component is *cdc54*. *cdc54* is essential for the initiation of DNA replication and is one of the key targets of the signal transduction pathways that regulate the cell cycle mentioned in Chapter 1.

Figure 2.5 shows the plot of the coefficients of the first two principal components for the covariance matrix (the coefficients are simply the eigenvectors of the covariance matrix). The elliptical shape is the expected result of such a plot if the distribution of the data is approximately multivariate normal [25]. Particularly, all genes that are the same distance from the origin in this two dimensional space with the axes given by the eigenvalues fall on the perimeter of an ellipse. However, notice that there are a significant number of points scattered far outside the core. This indicates that differences in the range of the genes being

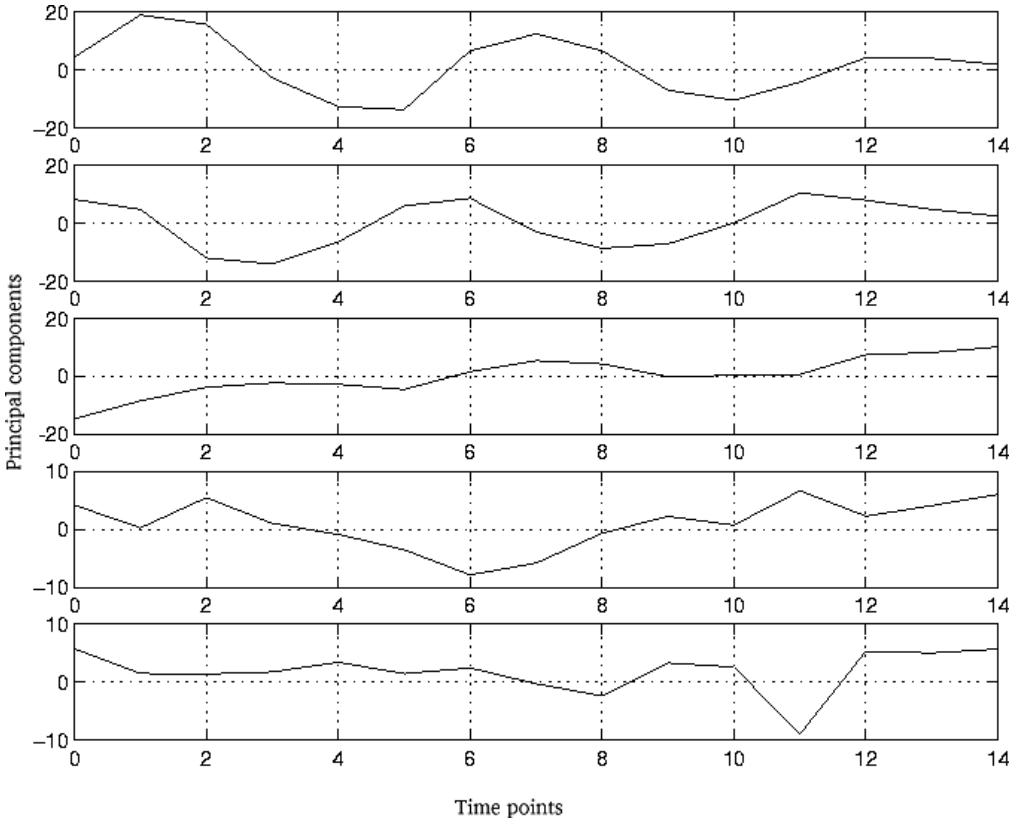

Figure 2.4: From Study 2: Behavior of the first five principal components over time from the covariance matrix for the *cdc15* data.

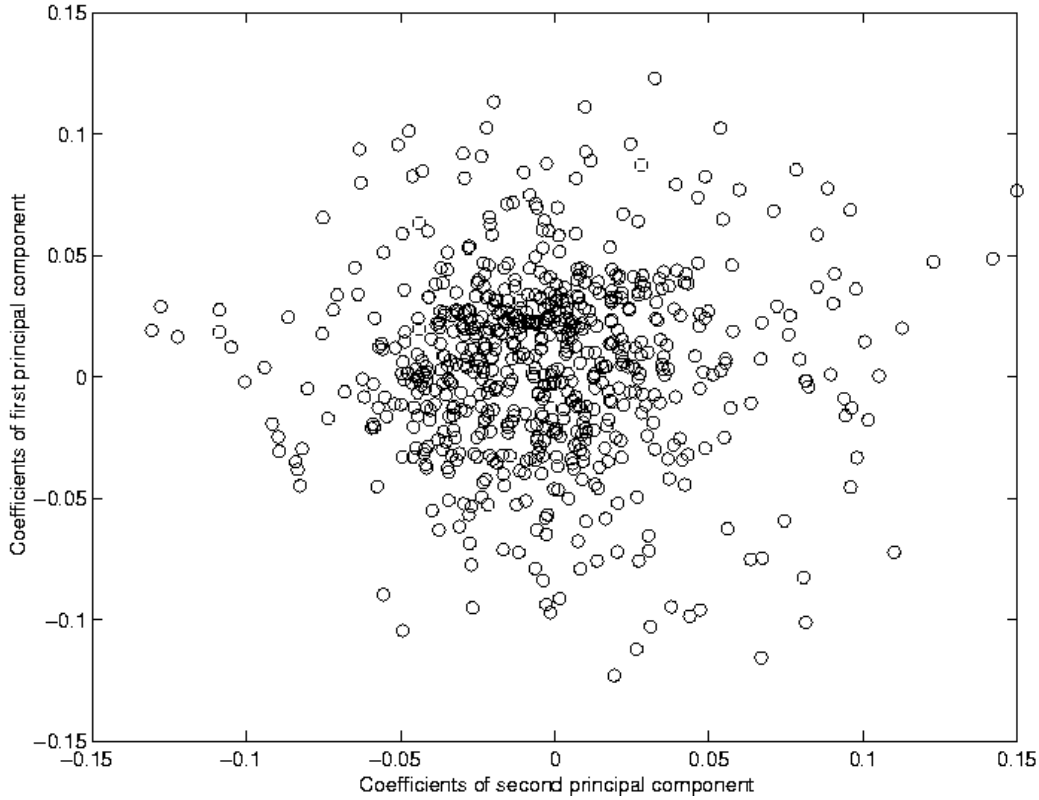

Figure 2.5: Plot of the coefficients of the first two components from the covariance matrix of gene dataset2.

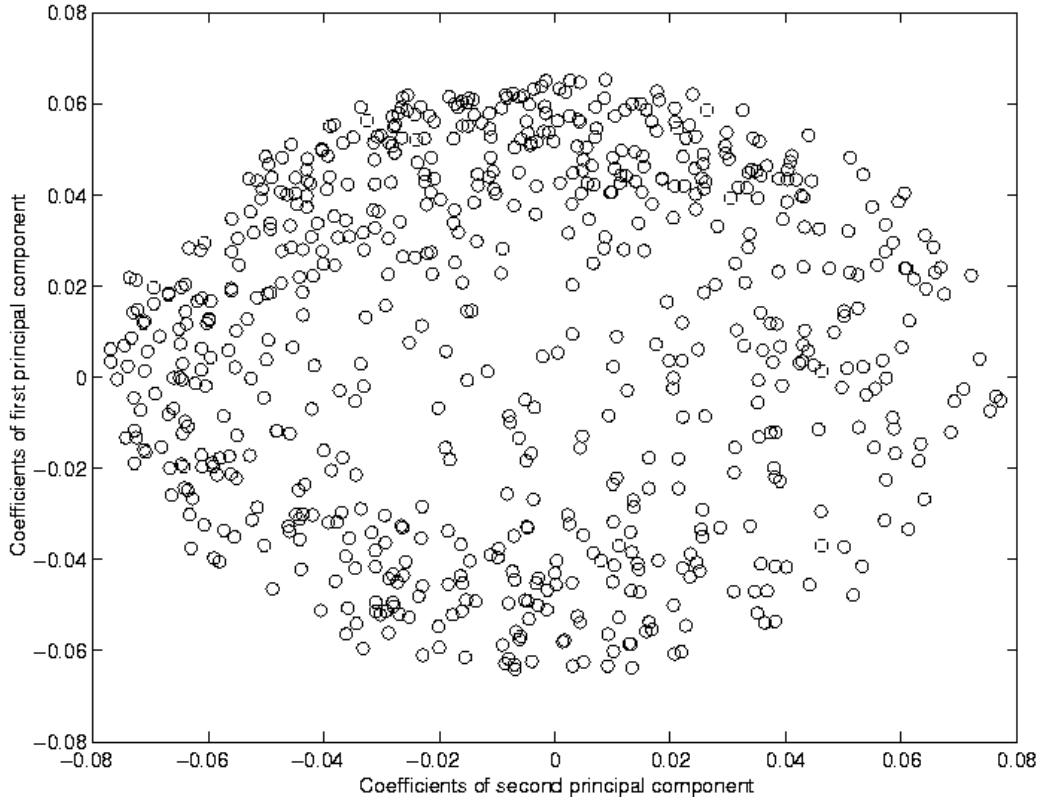

Figure 2.6: Plot of the coefficients of the first two components from the correlation matrix of gene dataset2.

measured affects the results and that correlation is likely a better choice. Figure 2.6 shows the same plot using the correlation matrix and the results are drastically improved. Notice that most of the points lie near the perimeter of a single ellipse.

Contrast these last images with the plot of the first and third components of the correlation matrix in Figure 2.7. Here the shape of the ellipse is more elongated so there seems to be a much larger range of values along the direction of the first principal component than the third.

Study 3

I now proceed to an analysis of the data generated from the EGF simulation. Using the EGF simulation I create two data sets: one with 61 time points (EGF dataset3) and one with 9 time points (EGF dataset4) over the same 120 second period. EGF Dataset4 emulates the dataset from Kholodenko [29] in order to compare the results from applications using the

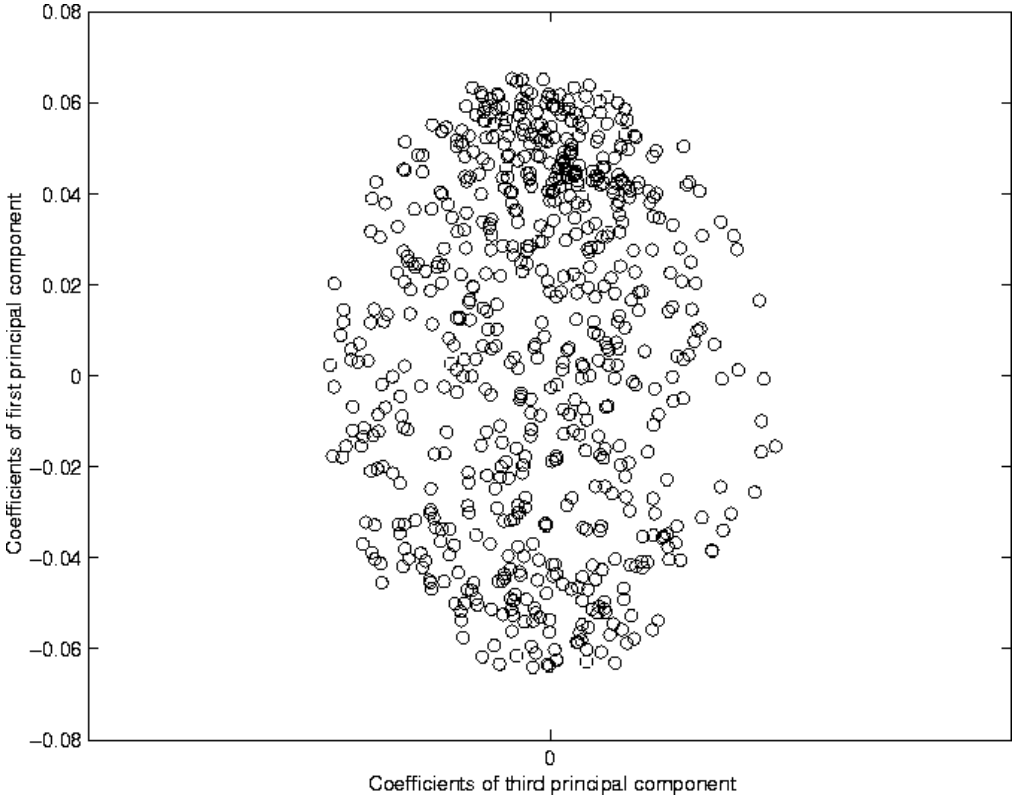

Figure 2.7: Plot of the coefficients of the first and third components from the correlation matrix of gene dataset2.

simulated data to those that could be obtained with actual experimental data. The number of data points for the larger EGF dataset3 was chosen arbitrarily.

The principal component analysis is performed on both the 9 and 61 time point data sets using the correlation coefficient matrix with the protein states as the variables. Thus 23 principal components are obtained. Figure 2.8 shows the behavior of the first four principal components over the time course for each data set. There is a greater amount of detail revealed in EGF dataset3 as compared to EGF dataset4. Although the behavior of the first component is similar in both data sets, the subsequent components show quite different results. This is an important observation for experimentalists trying to decide on the number of time points to include in a dataset.

The first component in both graphs of Figure 2.8 show the slow climb and leveling off of concentration seen in the experimental results [29]. None of the first four components of EGF dataset3 show the peak then decrease in concentration found in the experimental results and in components 2 and 4 of EGF dataset4. The fourth component of EGF dataset3, however, has a very similar pattern to the rate of EGFR dephosphorylation seen in the computational results of Kholodenko.

One can compare the relative ease of interpreting the overall patterns in the data using the principal components as opposed to the plot of the concentration of all the proteins over time (Figure 2.9) even for such a small number of proteins. Notice that there are several types of behavior visible in this graph:

- initially high concentration decreases rapidly, then stabilizes
- slow rise and leveling off of concentration
- slight rise in concentration followed by a slow decrease.

These patterns are comparable to the patterns seen in the time courses of the larger principal components in Figure 2.8.

The percentage of the variance explained by each principal component is somewhat difficult to interpret for the 9 point time course. Most of the eigenvalues were very small or negative, with only the first 5 or 6 being of reasonable size for interpretation. For the 61 point time course, however, the first five eigenvalues capture over 99% of the total information in the data set. The eigenvalues for the first five principal components are listed in Table 2.4.

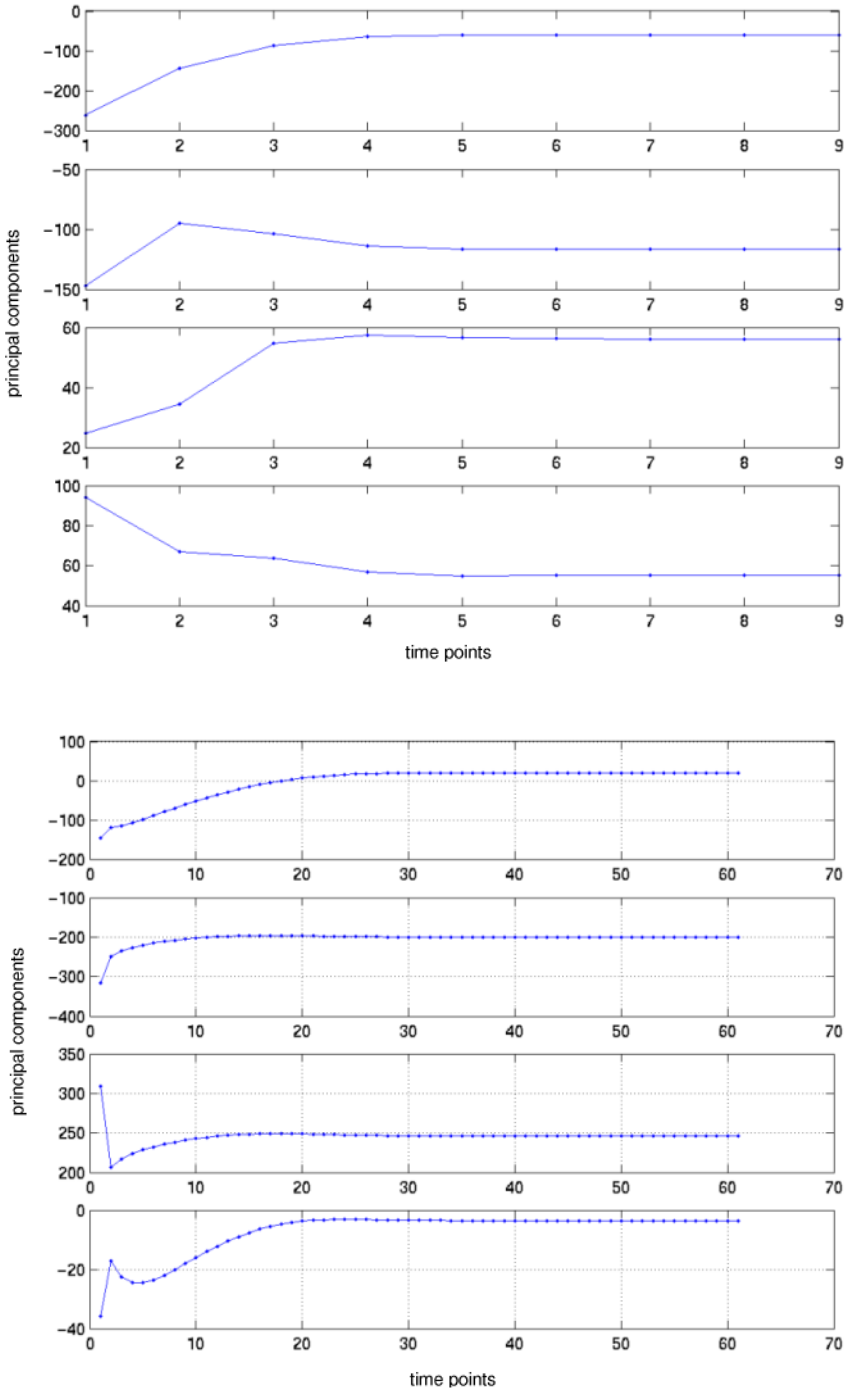

Figure 2.8: Time courses for the first four components from the correlation matrix of EGF dataset4 (top) and EGF dataset3 (bottom).

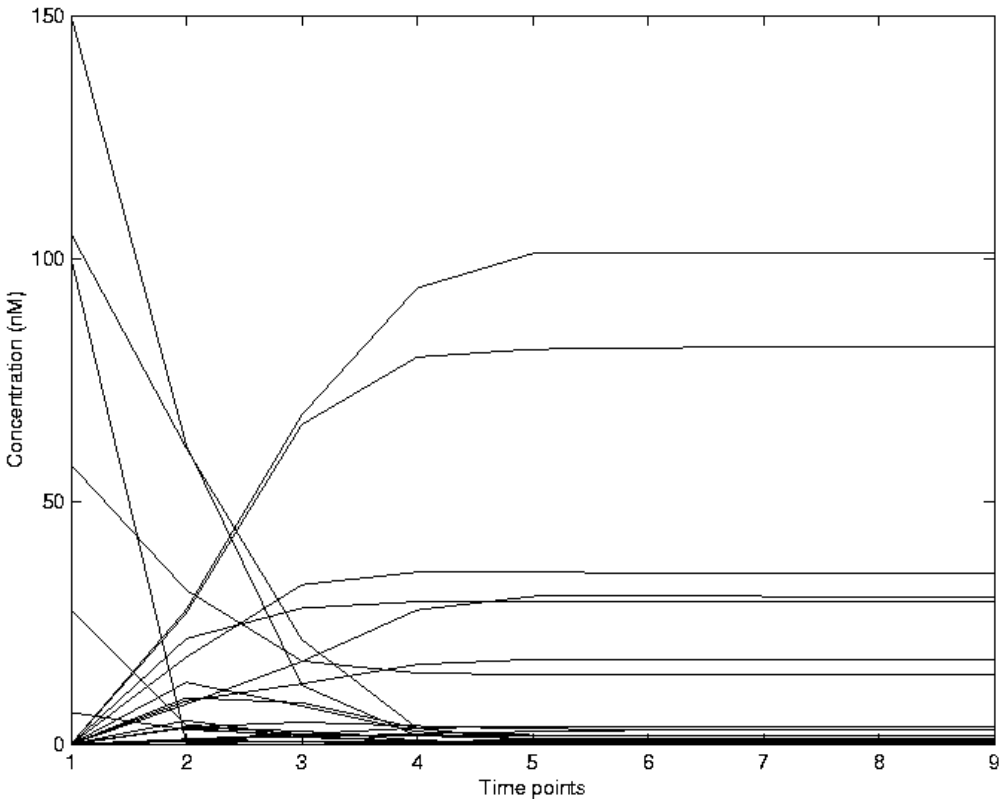

Figure 2.9: The concentrations of each protein in the network from EGF dataset4

EGF	
eigenvalues	% variance
15.9357	69.2856
4.4274	19.2494
1.8468	8.0297
0.6364	2.7671
0.1048	0.4559

Table 2.4: Eigenvalues and percentage of total variance for the first five principal components from the correlation matrix of EGF dataset3.

Number	Protein Abbr.	Actual Protein (Complex)
1	EGF	Epidermal growth factor
2	R	Epidermal growth factor receptor
3	Ra	Activated receptor-EGF complex
4	R2	Dimerized receptor-EGF complex
5	RP	Phosphorylated, dimerized receptor-EGF complex
6	PLC	Phospholipase C-γ
7	RPL	Receptor-PLC complex
8	RPLP	Phosphorylated receptor-PLC complex
9	PLCP	Phosphorylated PLC
10	PLCP-I	Phosphorylated PLC-inositol
11	GRB	Growth factor-binding protein 2
12	RG	Receptor-GRB complex
13	SOS	Son of sevenless protein
14	RGS	Receptor-GRB-SOS complex
15	GS	GRB-SOS complex
16	SHC	Src homology and collagen domain protein
17	RSH	Receptor-SHC complex
18	RSHP	Phosphorylated receptor-SHC complex
19	SHP	Phosphorylated SHC
20	RSHG	Receptor-SHC-GRB complex
21	SHG	SHC-GRB complex
22	RSHGS	Receptor-SHC-GRB-SOS complex
23	SHGS	SHC-GRB-SOS complex

Table 2.5: A numbered list of the proteins and protein complexes involved in the early events of the EGF signal transduction pathway.

The plots of the coefficients are particularly interesting. As shown in Figure 2.10, the proteins all fall around the perimeter of an ellipse. The most pleasing feature, however, is the appearance of the proteins in distinct groups around the perimeter. This is the first indication that the protein data also appears to contain natural grouping, a fact that I shall confirm using clustering. Notice also that EGF dataset3 gives more precise information on the location of each protein in the two dimensional space. Thus points that were superimposed in the results of EGF dataset4 are now distinguishable. Note that the numbers associated with points of the plot designate proteins of the pathway, as listed in Table 2.5.

2.4.2 Application of SOMs

Next I create SOMs for the simulated EGF data. All SOMs are created using GeneCluster [1]. This convenience comes at the expense of flexibility in choosing certain aspects of

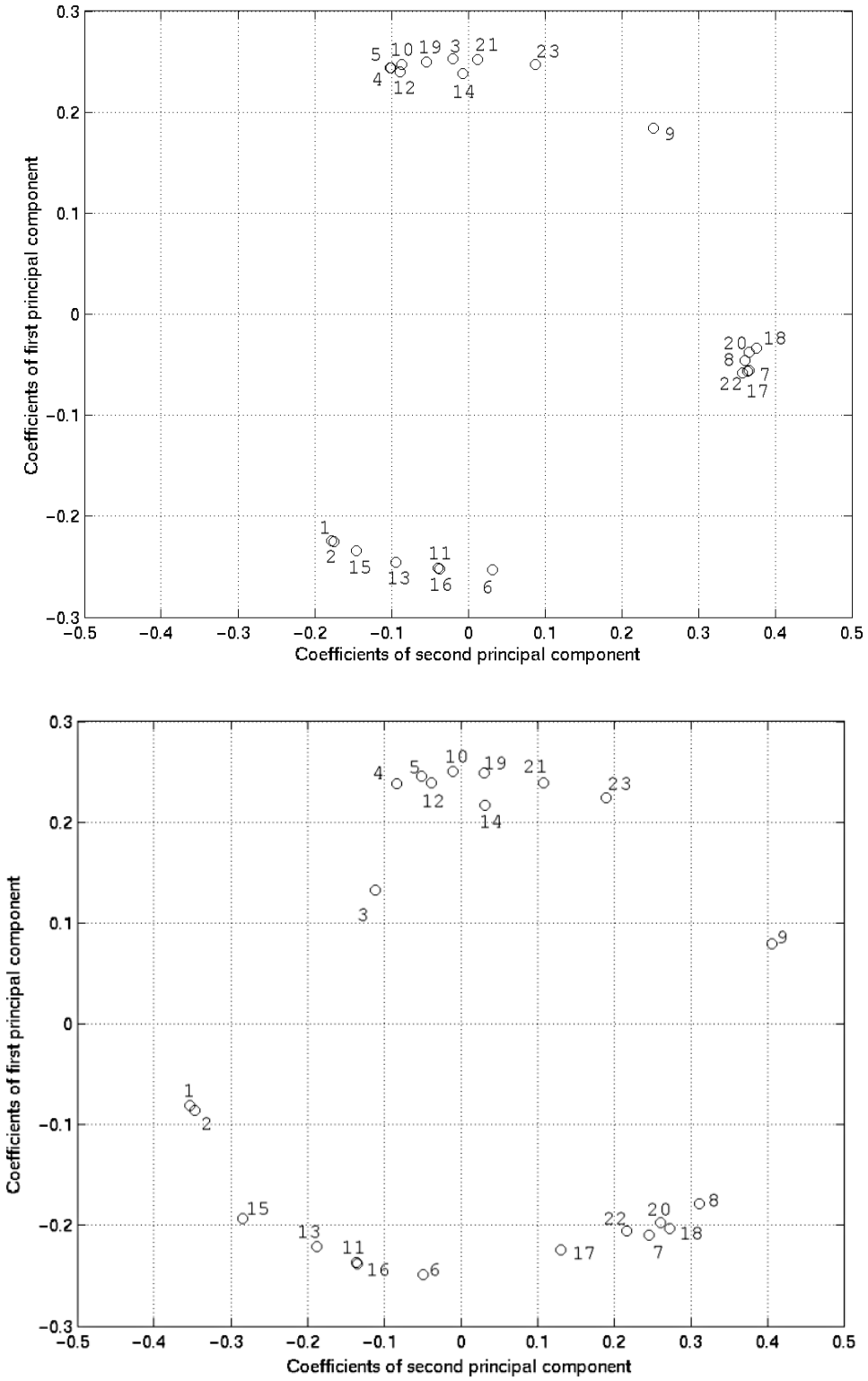

Figure 2.10: Plots of the coefficients of the first two components from the correlation matrix of EGF dataset4 (top) and EGF dataset 3 (bottom).

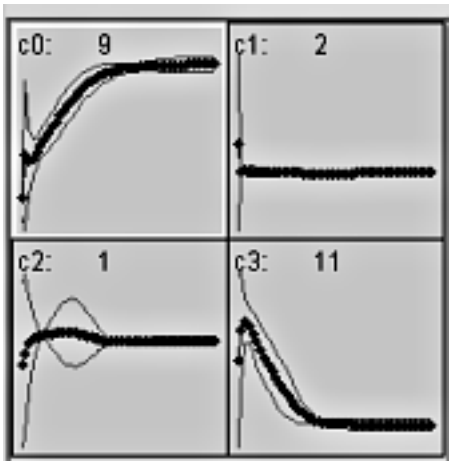

Figure 2.11: Centroid time courses for the 4 cluster SOM of EGF dataset5.

the process such as the lattice shape. All lattices are rectangular, but the number of nodes and the number of rows and columns in the lattice are chosen by the user. Note that all data sets used for clustering are first normalized using GeneCluster to a zero mean and standard deviation of 1.

The first clustering of the EGF data is done using a data set with 51 time points (the default time step for Gepasi, the data simulation software [36]), designated as EGF dataset5. The results of the PCA shown in Figure 2.10 lead us to attempt a clustering with 4 lattice nodes, since the data appears to contain 4 natural groupings. The results of this clustering are shown in Figure 2.11. The time courses of the two most populated clusters agree with the patterns observed in Kholodenko [29] and seen in Figure 2.9. I also create a SOM with 6 nodes, and the results are shown in Figure 2.12. The 6 cluster SOM captures more details, especially in the early moments of the reaction.

I create SOMs for EGF dataset4, which has fewer time points. These SOMs are not entirely successful, as some clusters are left blank. That is, in a clustering using 4 nodes, only 3 nodes actually settle on data points. This is presumably a result of a node following one or two data points until those points merge with another cluster. This is likely a problem caused by both the small number of proteins and the reduced number of time points. Nonetheless, the clusters that do contain proteins show similar results to the previous clustering with EGF dataset5: the two main patterns are a gradual increase and an early peak followed by a leveling off. These patterns are actually clearer in the SOM of

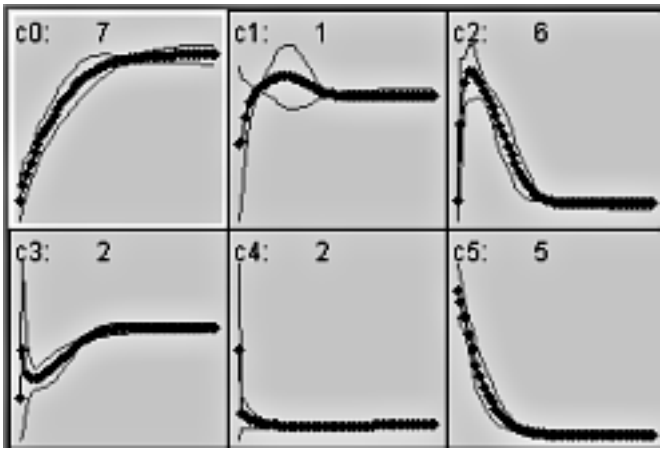

Figure 2.12: Centroid time courses for the 6 cluster SOM of EGF dataset5.

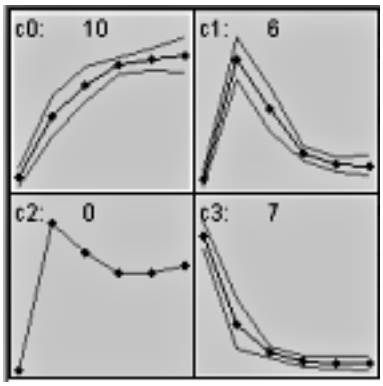

Figure 2.13: Centroid time courses for the 4 cluster SOM of EGF dataset4.

EGF dataset4.

I conducted preliminary studies investigating the effect of lattice shape. The arrangement of the nodes does appear to affect the clustering in some cases, however no overall "best" shape can be determined based on the results thus far.

To investigate the effect of changing the initial EGF concentration on the clustering results, simulations were created with initial EGF concentrations. Some shuffling between clusters is observed. In particular, PLC-P, SHG, SHGS, and PLC move to different clusters at different initial concentrations of EGF. None of these shufflings reflects a major change in behavior over time (a transient to sustained concentration increase, for example). The reasons for the shuffling between clusters are not known.

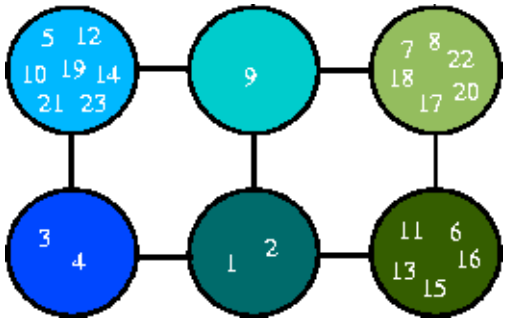

Figure 2.14: Clustering of the proteins of the EGF pathway. The results are from a 6 cluster SOM of EGF dataset5. The circles represent the nodes of the network and are arranged in the lattice shape used for the creation of the SOM. The numbers correspond to the proteins listed in Table 2.5

2.5 Discussion of multivariate analysis results

In the study by Holter *et al*, the grouping of genes in the principal component analysis of gene expression data is similar to the clustering of the genes. Is this also the case for the analysis of the simulated EGF pathway data?

Figure 2.14 shows the grouping of proteins from a 6 cluster SOM of EGF dataset5. If one compares these results to the results of the principal component analysis shown in Figure 2.10, one finds that the groups of proteins in the plot of the component coefficients are the same as the clusters of the SOM.

There are two other questions that I would like to address:

1. Is the placement of the groupings in Figure 2.10 significant in some way?

2. What is the connection between proteins in a cluster?

To investigate these questions, I create Figure 2.15 so that the position in the pathway of the cluster members can be visualized. There are no obvious similarities in function between members of a given cluster. That is, proteins within a cluster share similar patterns of activation, but don't appear to share any specific role in the pathway, nor have the proteins been grouped as phosphorylated/nonphosphorylated, say. It does appear that most protein complexes involving the EGF receptor are clustered together, with the exception of RG and RGS, but this could definitely be specific to this application.

PCA and clustering should be applied to data from several different pathways to

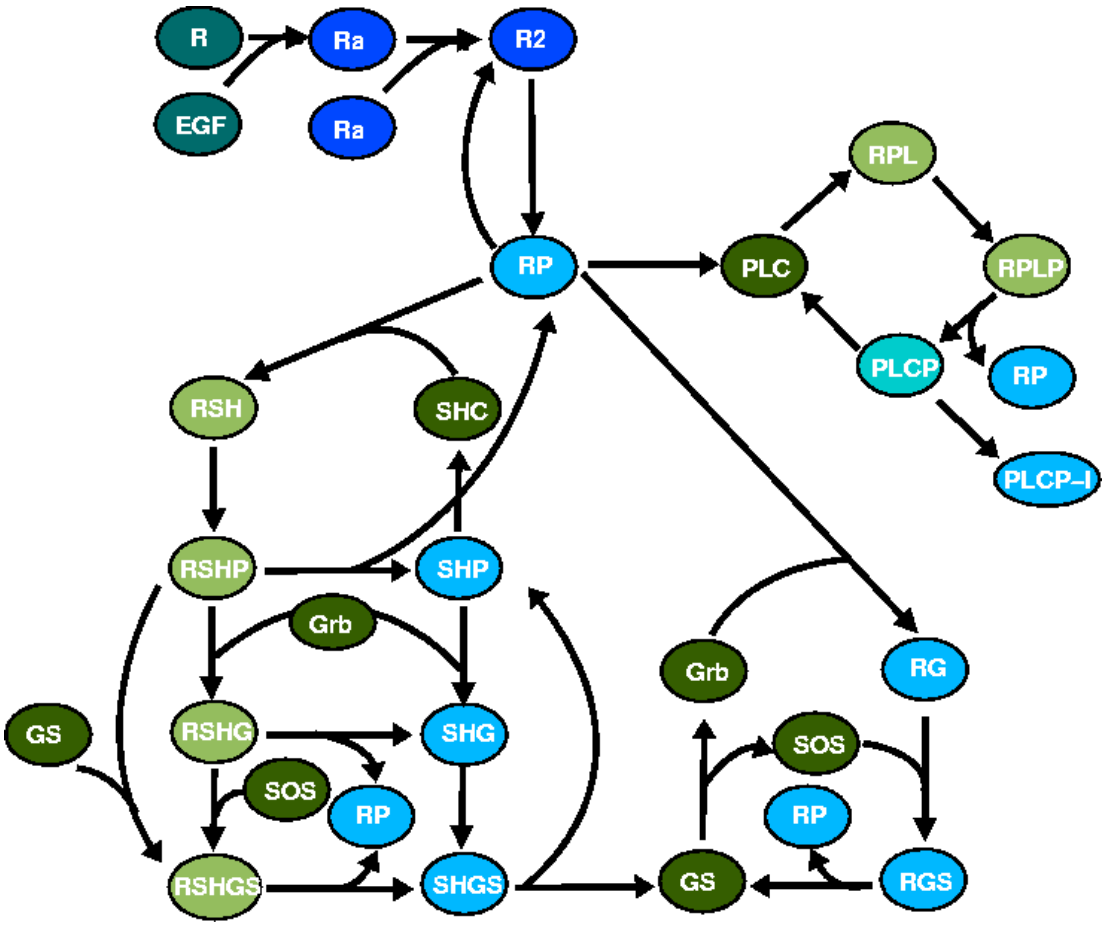

Figure 2.15: Proteins in the pathway are hatched according to their cluster membership. The results are for a 6 node clustering of EGF dataset5.

further investigate the relationship of proteins within a cluster. I believe that the identification of relationships within a cluster is complicated in this case by the fact that the members of the pathway are not individual proteins, but are complexes of proteins. Thus comparisons of amino acid sequence homology, for example, can not be performed. If the clustering could be done on data for individual proteins, then perhaps further similarities within clusters could be identified.

The results of this study do show that there appear to be natural groupings in the data and that multivariate analysis allows an investigator to study the time courses and relationships of these groupings.

One suggestion for future studies of signal transduction pathways using multivariate techniques, is that the number of components should be increased. The methods presented in this chapter are intended for studying large data sets, such as the gene expression datasets obtained from microarrays. Thus, I recommend including more pathway components in future studies of this sort.

Chapter 3

Reverse Engineering of Genetic Networks

This chapter covers several reverse engineering efforts for investigating networks of genes. Although our research is aimed primarily at signal transduction pathways composed of reactions between proteins, it is instructive to study genetic network techniques, in the hope that they will also be applicable to protein pathways. We first consider some of the differences between protein and gene networks.

A graphical representation of a network consists of a set of nodes and a set of edges joining pairs of nodes. Let the nodes represent proteins in the case of a signal transduction pathway and genes in the case of a genetic network: what do the edges in each of the graphs represent? In the case of a gene network, there does not seem to be a specific meaning attributed to these connections. Generally, genes don't affect each other in a direct way, that is, they don't interact physically. An edge between two genes in a network really represents the idea that a change in the activity of gene A will cause some change in the activity of gene B. This change is the result of the transcription and translation of gene A and the subsequent activities of its gene product (possibly in a signaling pathway).

An edge between two protein nodes represents a more direct connection, specifically that two proteins are involved in a chemical reaction. The form of this chemical reaction (a complex forming, a phosphorylation, a degradation) is not necessarily known, but actual physical effects are implied. In Chapter 4 it becomes evident that the fact that the edges of the graph have a specific nature affects the adaptation of these gene network methods to study protein interactions.

Another important difference between gene and protein network determination is the available data. Because the effects of gene interactions are not directly transmitted, the time scale is much slower. Thus, current technology is capable of capturing adequate snapshots of the steps of the interactions. On the other hand, protein reactions occur within seconds and so many steps in a pathway may be carried out between measurements. This should also be taken into account when attempting to draw inferences about a series of connections between proteins.

Despite these differences, it is advantageous to study the methods presented here. Both gene and protein networks are large, complex biological systems involved in regulating cell function. Also, most reverse engineering attempts of the kind described in this chapter have only been applied to the more readily available gene expression data and, thus, are our only guidelines. This chapter introduces three reverse engineering methods: one for discrete Boolean networks, one incorporating continuous gene regulation, and one using Bayesian networks.

3.1 Discrete Boolean Networks

A discrete Boolean network is a model of a system at specific time points using binary values to indicate the state of each component of the network. In the case of a genetic network, the state at time t of the network is given by a value of 1, to indicate that the gene is "on" or a value of 0, to indicate that the gene is "off", for each gene in the network. Here, "on" implies that the gene is expressed above a specified threshold. The network evolves to a different state at the next time point based on logical rules that dictate the interactions between genes. For example, consider the network of 3 genes A, B, and C, whose rules of interaction are:

$$\begin{aligned} A &= B \\ B &= \neg(A \wedge C) \\ C &= A \wedge B \wedge \neg C \end{aligned} \tag{3.1}$$

where $\neg$ denotes the logical "not" and $\wedge$ denotes logical "and" . Given these rules and a starting state, the network of genes evolves as shown in Figure 3.1.

The goal of reverse engineering the network is to determine these logical rules, and thus to be able to predict the evolution of the network over time. To find these rules, first

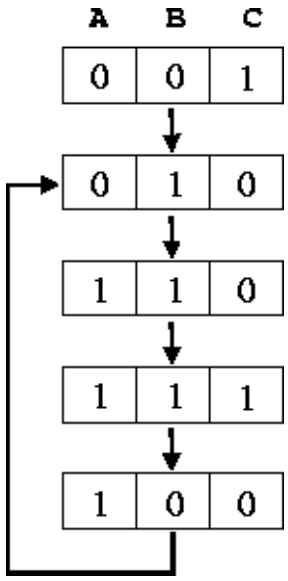

Figure 3.1: The evolution of a three component discrete network according to the rules given below.

measure the gene expression of all the genes in the network many times for each state. This could correspond, for example, to running different experiments and assaying the gene expression of every gene over a series of time points. One then quantizes the data to binary data based on some threshold of expression value. At this point one can attempt to make inferences about the logical rules that could produce state $(t+1)$ from state t.

The rest of this section describes one algorithm that finds these logical rules that define a discrete Boolean genetic network. This Reverse Engineering Algorithm (REVEAL), was developed by Liang, Fuhrman, and Somogyi and presented at the Pacific Symposium on Biocomputing (1999) [34].

3.1.1 Shannon Entropy and REVEAL

A table of binary data is compiled for the activity of each gene (0 =“off”, 1 =“on”) at successive time points t and $t+1$, as shown in Table 3.1. The goal is to discover the underlying rules of the network that created this data. One might ask the question: given that gene A has a value of 0 at time t, what is the probability that gene B has a value of 0 at time $t+1$? In some sense, one is asking what level of uncertainty remains in determining the value of B once the value of A is known. This provides a measure of the influence of

	Time t			Time $t+1$		
Assay	A	B	C	A	B	C
1	0	0	1	0	1	0
2	1	0	1	0	0	0
3	1	1	1	1	0	0
4	0	0	0	0	1	0
5	1	0	0	0	1	0
6	0	1	1	1	1	0
7	1	1	0	1	1	1
8	0	1	0	1	1	0
9	1	1	1	1	0	0
10	0	0	1	0	1	0

Table 3.1: Compiled binary data for the three genes at time points t and $t+1$.

gene A on gene B.

The role of uncertainty involved in a choice can be quantified by a measure called Shannon entropy, introduced in the landmark work by C.E. Shannon for the study of discrete signaling systems, such as telegraphy [47]. Suppose that the random variable X takes the value x with probability $p(x)$. The Shannon entropy of X is given by

$$H(X) = -\sum_{x} p(x) \log_2 p(x) \tag{3.2}$$

One can calculate the probabilities $p(x)$ from Table 3.1. Given the probability p(0) of gene A being off and the probability, p(1), of the gene being on ($p(1) = 1 - p(0)$ since the probabilities must sum to one), it is possible to calculate $H(A)$ to find the level of uncertainty in choosing the state of A. Note from Equation (3.2) that H takes its maximum value, equal to one, when all states are equally probable ($p(0) = p(1) = 0.5$).

Returning to the earlier network example: for time t we can calculate the sample probabilities of a gene begin "on" or "off" from Table 3.1. For gene A, for example, the gene is "on" (has an activity level of 1) 5 times out of a total 10 measurements. Thus $p(1) = 0.5$ for gene A at time t. Using these probabilities, we can calculate the entropies

from Equation 3.2:

$$H(A) = H(B) = -0.5\log_2(0.5) - 0.5\log_2(0.5) = 1$$
$$H(C) = -0.4\log_2(0.6) - 0.6\log_2(0.6) \approx 0.97 \tag{3.3}$$

There is a high degree of uncertainty as to what the state of each gene will be in this example. If, on the other hand, $p(0) = 0.9$ and $p(1) = 0.1$ for some gene X, then $H(X) \approx 0.47$, which reflects the fact that one would expect X to be in the off state with high probability. (Thus there is not a lot of choice involved in determining the state of X.) This feature characterizes Shannon entropy.

We are interested in determining relationships between sets of genes in a system where, potentially, many genes play a role in regulating a single gene. Thus, one would also like to quantify the entropies of combinations of elements of a network. The joint Shannon entropy of two random variables, X and Y, is

$$H(X,Y) = -\sum_x \sum_y p(x,y)\log_2 p(x,y) \tag{3.4}$$

This definition can be extended to find the combined entropy of n random variables:

$$H(X_1, X_2, \ldots, X_n) = -\sum_{x_1}\sum_{x_2}\cdots\sum_{x_n} p(x_1, x_2, \ldots, x_n)\log_2 p(x_1, x_2, \ldots, x_n) \tag{3.5}$$

A definition of conditional entropy follows from the previous results: $H(X|Y)$ measures the uncertainty left in the determination of X once information about the value of Y is known. The conditional and joint entropies are related by:

$$H(X,Y) = H(Y|X) + H(X) = H(X|Y) + H(Y) \tag{3.6}$$

3.1.2 Recognizing dependence between genes

The motivation of REVEAL is to discover what combination of input elements of the genetic network determines the output expression of each gene. The key to accomplishing this goal for a given element, Y, using Shannon entropies is to find the set of elements $X_1, \ldots, X_k$ such that

$$H(Y, X_1, \ldots, X_k) = H(X_1, \ldots, X_k) \tag{3.7}$$

To describe the above statement in the language of the previous section, the uncertainty in choosing the states of all of $Y, X_1, \ldots, X_k$ is the same as the uncertainty in the choice of $X_1, \ldots, X_k$. So, no new information is needed, no uncertainty added, in determining Y in addition to this set of elements. Thus, the state of Y must be completely determined by this set of elements.

Once this set of genes is found, it remains only to find the logical rule that specifies the state of Y given a combination of input states for the set. This rule is determined by examining the table of values (drawn from the data used to determine the probabilities $p(x)$ initially) of the input values from $X_1, \ldots, X_k$ and the output of Y.

3.1.3 The REVEAL method

Now that the dependence of a gene on a given set of genes can be recognized, a systematic method for finding such a match needs to be determined. REVEAL proceeds as follows:

Test 1: Only one other gene determines the state of gene Y: Search for matches using single input rules. Such a match involves either the relationship

t	$t+1$
X	Y
1	1
0	0

or

t	$t+1$
X	Y
1	0
0	1

that is, equality or inequality. Thus, for a given gene, Y, test each gene X_i to see whether $H(Y, X_i) = H(X_i)$. If such an X_i is found, then we have identified the gene that completely determines the output of Y. At this point, construct a table of values with input (t) values of X_i and output ($t+1$) values of Y from Table 3.1 to determine the rule that governs the output for Y. If no such X_i is found, then Y is not completely determined by any one single gene in the network.

Test 2: Two genes determine the state of gene Y: All combinations X_i, X_j are tested as inputs determining the output of each gene that has not yet been assigned a one-input rule. So, for each remaining gene Y, test $H(Y, X_i, X_j) = H(X_i, X_j)$ for all combinations (i, j) where $i \neq j$.

Test k: k genes determine the state of gene Y: This process is continued until either the rule for each gene is determined or a specified maximum number of allowed inputs is reached (the reason for this upper limit is discussed below). Thus, at step k, all k-tuples of genes are tested to determine whether $H(Y, X_1, \ldots, X_k) = H(X_1, \ldots, X_k)$.

Consider the example network: A_t denotes the input value of gene A and A_{t+1} denotes the output value of gene A. Calculate the entropy of each genes at time t and the joint entropies at time $t+1$ for each gene with respect to all input genes:

$$\begin{array}{llll} H(A_t) = 1 & H(A_{t+1}, B_t) = 1 & H(B_{t+1}, A_t) = 1.49 & H(C_{t+1}, A_t) = 1.36 \\ H(B_t) = 1 & H(A_{t+1}, C_t) = 1.97 & H(B_{t+1}, B_t) = 1.85 & H(C_{t+1}, B_t) = 1.36 \\ H(C_t) = 0.97 & H(A_{t+1}, A_t) = 1.97 & H(B_{t+1}, C_t) = 1.57 & H(C_{t+1}, C_t) = 1.30 \end{array}$$

The only match for $H(Y, X) = H(X)$ is $H(A_{t+1}, B_t) = H(B_t)$. Thus the output of B depends on A. To determine the rule describing this dependence, a table of values is extracted from Table 3.1:

B_t	A_{t+1}
0	0
0	0
1	1
0	0
0	0
1	1
1	1
1	1
1	1
0	0

So the rule is $A_{t+1} = B_t$.

Proceeding in the same way, one can determine all the 2-input-gene rules. A match is obtained for $H(B_{t+1}, A_t, C_t) = H(A_t, C_t)$. Looking at the table of values:

A_t	C_t	B_{t+1}
0	1	1
1	1	0
1	1	0
0	0	1
1	0	1
0	1	1
1	0	1
0	0	1
1	1	0
0	1	1

the rule is $B_{t+1} = (A \wedge C)'$. The 3-input-gene rule for C is determined in a similar way.

3.1.4 Difficulties with REVEAL Implementation

There are several difficulties involved with the actual implementation of the REVEAL method. On one hand, there is the problem that as k (the number of genes involved in input rule) increases, the computational expense involved in determining the Shannon entropies for testing each k-tuple increases quite significantly. Even though the number of rules remaining to be determined decreases as successful matches are found, the number of combinations to be tested in each iteration increases much more quickly. For a network of n genes there are $\binom{n}{k}$ combinations to test in the k^{th} iteration for each remaining gene.

Also, it seems that even once a match has been made between a set of input genes and a given output gene, determining the rule which governs the relationship from a table of values would become increasingly difficult as k increases. The creators of REVEAL have also found that the accuracy of the matching decreases as k increases and as the number of trials (input-output pairs) decreases.

Another difficulty in the application of REVEAL to a real data set obtained, for example, from a set of microarrays, is in converting the expression values to binary values. The authors of REVEAL mention the possibility of extending the method from Boolean to multiple-state values. This would likely allow more of the fine details of a real biological network to be included, although the problem of determining thresholds for the quantization of the data remains.

However, despite these difficulties, the method has successfully identified connections in artificial networks containing rules up to $k = 3$. The authors suggest that parallelization and the use of wiring and rule constraints to limit the search space will improve the efficiency and thus the applicability of REVEAL [34]. It has been suggested that methods such as clustering may be useful in the identification of wiring constraints by reducing the number of components in the network. Also, it would certainly be a great advantage if biological information about networks in general as well as a specific network being investigated, could be integrated into REVEAL to decrease the size of the search space of possible inputs for each step.

3.2 Continuous expression and regulation

Some features of gene and protein regulation and interaction are poorly represented using a Boolean network. For example, genes may have different regulatory influences at different levels of expression, and may regulate other genes to varying degrees. As well, the need to specify a fixed number of regulatory inputs with a Boolean network limits the ability to fully describe the interactions taking place.

The models offering continuous expression and regulation fall into two categories: discrete and continuous time. I will discuss an example of a discrete method formulated by Weaver and Workman. Although the treatment of the problem using a discrete time system means a loss of realism in the model, this simplification has the advantage of making the problem computationally tractable and also requires fewer data points for the purposes of reverse engineering the regulatory networks.

In the model, a vector, u, represents the expression state of the network at time t. The amount of gene i expressed at time t is given by $u_i(t)$. In addition, a weight matrix, W, is defined to represent the regulatory interaction between the elements in the network (e.g. genes). Each row of the matrix lists all the regulatory inputs for one gene, where

- $w_{ij} > 0 \Rightarrow$ stimulation of transcription of gene i by gene j
- $w_{ij} < 0 \Rightarrow$ repression of transcription of gene i by gene j
- $w_{ij} = 0 \Rightarrow$ gene j has no effect on the transcription of gene i

Using these values, the regulatory influence of all other genes on gene i can be

quantified by summing the influence of each gene multiplied by the activity level of that gene at time t:

$$r_i(t) = \sum_j w_{ij} u_j(t) \tag{3.8}$$

In other words, the regulatory control of one gene is a linear superposition of the influences of all other genes in the network, weighted by relative "connection strengths" w_{ij}.

The next step is to calculate the response of each gene to its regulatory input. This is done via a nonlinear dose-response (or "squashing") function:

$$x_i(t+1) = \frac{1}{1 + e^{-(\alpha_i r_i(t) + \beta_i)}} \tag{3.9}$$

Included in this function are two gene-specific parameters, α_i, a positive real number and β_i, which can be any real number. These parameters define the shape of the dose-response curve for each gene, as pictured in Figure 3.2.

The output of this function is a number between 0 and 1 for each gene. To convert this number into a value comparable to realistic data output from a gene expression assay, the number is multiplied by a "maximal expression level", which corresponds to the expected or observed maximal expression for each gene.

Reverse engineering is performed with this model using a neural network approach: gene expression data for all the genes in the network are divided into a training set and a test set: the trials of the training set will be used to fit the parameters of the weight matrix and the test set (of at least one trial) is used to test the accuracy of the results produced with these fitted parameters. The gene expression data is ideally obtained from actual experiments using microarrays. The expression values of each gene over the set of arrays is normalized relative to the given gene's maximal expression value. These normalized values are compared to the x_i values during training.

Once the network is trained and tested, the weight matrix defines the connections between all the genes in the network. At this point, the weight matrix can be used to predict the effect of perturbations to the system, such as a mutation of a gene of the network.

Several assumptions are made in order to simplify the process of fitting the weight matrix parameters. In order to determine one row of the matrix at a time, it is assumed that the regulation of each gene is an independent event. This is not a valid assumption for all biochemical systems. The advantage of this simplification is that the fitting problem

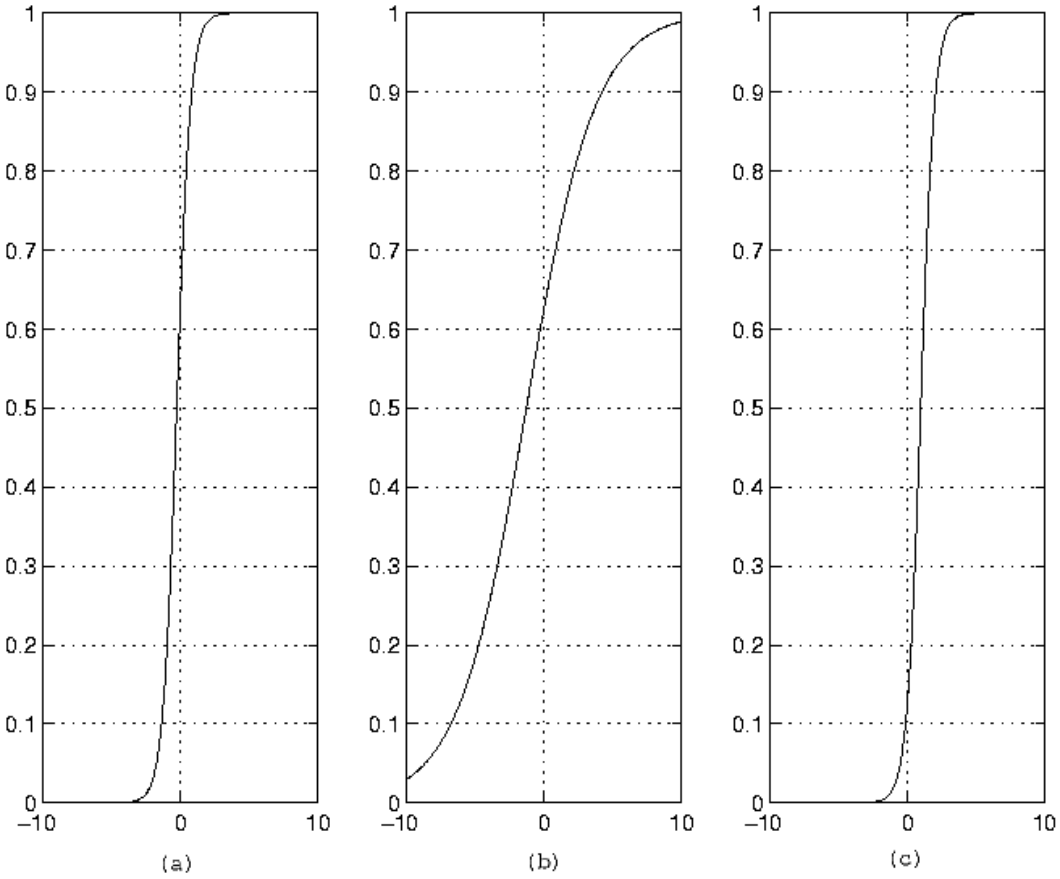

Figure 3.2: Several possible dose response functions with different parameter values. (a) $\alpha = 2$, $\beta = 0.5$, (b) $\alpha = 0.4$, $\beta = 0.5$, (c) $\alpha = 2$, $\beta = -2$.

is divided into smaller cases, and thus a smaller data set can be used to fit a smaller set of parameters. Given a larger data set, it would be interesting to compare the results of performing the training on the entire weight matrix to those of training each row of the matrix under the independence assumption.

Another problematic assumption is that of a linear regulatory relationship between genes. Specifically, it is not obvious that there should be such a direct relationship between a gene's expression level and the active amount of it's resultant gene product produced. It is the activated protein that would be involved in the regulation of other genes.

A problem that arises directly from the method is that of the maximal expression level. The designation of such a value would likely require a literature search to determine what the common expression values for a given gene are. Since the aim of this work is to also consider systems for which complete data is not yet available, this could be a difficulty. Even assuming that a reasonable estimate for this value could be made for each gene from the maximum value observed over the set of assays being used as data, fitting for those genes that are being expressed at maximal levels or that are nearly repressed will be subject to errors [55]. This can be seen from the squashing function, Equation (3.9), whose tails map a fair range of $r_i(t)$ values at the extremities to a small range of $x_i(t)$ values. This makes highly expressed or repressed genes more sensitive to noise and means that it is more difficult to obtain good values when "desquashing" to obtain real expression values for interpretation of the model.

An admirable feature of this model, however, is the ability to include the effects of environmental factors on expression using environmental variables added to the input vector and weight matrix. That is, by adding an extra column to the weight matrix, the influence of an environmental factor on the transcription of all genes in the network can be modeled. These influences are important since, as mentioned in Chapter 1, signal transduction is sensitive to the current state of the cell and its surroundings.

3.3 Bayesian networks

The Bayesian network approach to the problem of recreating biological connections between molecules or genes moves away from the rule-based system seen in Section 3.1 and into the domain of the probability and decision theory of expert systems. These new elements allow

the incorporation of uncertainty in the evaluation of networks. Uncertainty in observations, incomplete data, non-deterministic relationships, and other sources of uncertainty are commonplace in the study of gene and protein networks. Thus this approach is ideal for the study of these systems.

Expert systems are intended to replace the human "expert" decision maker with a computational counterpart. Early expert systems were purely rule-based. Initial attempts to use probability theory in expert systems [16] hit a roadblock of limited computational power and the idea (and its unmanageable calculations) was abandoned for over 10 years. In 1986, Pearl introduced Bayesian networks to expert systems [42] and in 1989 Andreassen *et al* [5] created MUNIN, a real-world expert system which could perform disease diagnoses. MUNIN was the first example of Bayesian networks in an expert system. There are two early applications of Bayesian networks to gene expression data. Friedman *et al* (2000) [14], the first to use Bayesian networks to study gene expression data, attempt to determine connections between genes by identifying salient features in networks that score well against the data. Hartemink *et al* do not begin from merely a set of genes of interest, but from full models representing competing hypotheses for a real biological system. These approaches encounter some similar problems and some problems unique to their chosen method. The focus in this discussion of Bayesian networks is on explaining the nature of such a network, its advantages for studying biological systems, and the case study performed by Hartemink.

There are many problems inherent in the study of gene expression data, some of which I have touched upon in earlier sections. For instance, the noisy and often incomplete nature of the reported data is sometimes not quantified using statistical measures. The amount of data is often limited in size and nonuniform, having been obtained from a series of distinct experiments. As well, in many of the analyses that have been discussed thus far, the complex nature of the relationships between elements of a biological network has not been properly addressed.

The use of Bayesian networks is advantageous in dealing with several of these issues. As opposed to REVEAL, for example, the number of elements controlling the expression of a given gene is not fixed. This allows more flexibility in modeling the interactions between elements of a network. Because Bayesian networks use probabilities in quantifying the fit of a model, they are less sensitive to noisy and incomplete data [17]. As well, Bayesian networks allow for the inclusion of unobservable factors in the model, as discussed below.

Another advantage of this approach is that the networks are easily interpretable, clearly specifying the dependencies between elements of the network. Also, there are by now many examples in other fields of training Bayesian networks from actual experimental data, and thus many obstacles have already been tackled.

3.3.1 The structure of Bayesian networks

Bayesian networks offer flexibility in describing a network. This is a valuable asset when considering their use for representing cellular signaling systems, which are widely diverse. A Bayesian network accommodates this diversity in the definition of its variables, which represent events. For example, an event could be a certain expression value of a gene. Variables may be either discrete or continuous and can have any number of mutually exclusive states. Thus one net could be used to model both proteins and gene expression, say, which are measured on quite different time scales. Variables can be *information variables*, which are observable quantities, or *latent variables*, which are unobservable quantities. This permits the inclusion of factors of suspected importance in the model, despite the fact that these elements may not be directly measurable.

Borrowing the notation of Friedman *et al*, the n nodes of a graph, G, are defined to be the set of random variables $X_1, \ldots, X_n$, where a given random variable X_i takes on a value x_i belonging to the state set of X_i. As noted above, this state set may be continuous or discrete and of any size (finite or infinite). For all cases considered below, random variables take on discrete values from a finite range.

A Bayesian network is a representation of a joint probability distribution using a directed acyclic graph (DAG). A directed graph consists of a set of nodes, as defined in the preceding paragraph, as well as a set of directed edges (arrows) connecting the nodes. The acyclic property of a DAG requires that no path following the directed edges of the graph can return to the starting node of the path. Given that a DAG specifies a directed relationship between nodes, a given node X has:

- a set of parents, $pa(X)$, that point directly to X.
- a set of children that are pointed to directly from X.
- a set of descendants that are reachable by traveling on a path starting at X and following the directed edges of the graph.

So, for example, in Figure 3.3, $pa(X_4) = \{X_2, X_5\}$. In this same network X_2 is the only child of X_1, since it is the only node pointed to directly from X_1. The set of descendants of X_1 is $\{X_2, X_3, X_4\}$, however.

A DAG that provides the topology of a Bayesian network defines a set of conditional independence statements between nodes of the graph based on the following property:

Each variable is conditionally independent of all nondescendent variables given its parents (3.10)

That is, once the parents of node X are known, information about nodes not descended from X can provide no more information about the probability of X begin in a given state.

The mathematical representation of conditional probability is $P(A|B) = x$, which means that if B is known and everything else known is irrelevant to A, then $P(A) = x$. Now the concept of conditional independence above can be defined.

Definition 1 *The variables A and C are independent given the variable B if*

$$P(A|B) = P(A|B, C) \tag{3.11}$$

This means that C provides no further information on the state of A. Several other results from probability are required to form the basis of this discussion of Bayesian networks. First,

The fundamental rule for probability calculus

$$P(A, B|C) = P(A|B, C)P(B|C) \tag{3.12}$$

One can derive from these rules and definitions Bayes' Rule upon which the Bayesian scoring metric is based:

Bayes' rule

$$P(B|A, C) = \frac{P(A|B, C)P(B, C)}{P(A, C)} \tag{3.13}$$

Since a Bayesian network is a DAG whose nodes satisfy condition (3.10), the joint probability distribution of a Bayesian network can be written in the product form (also

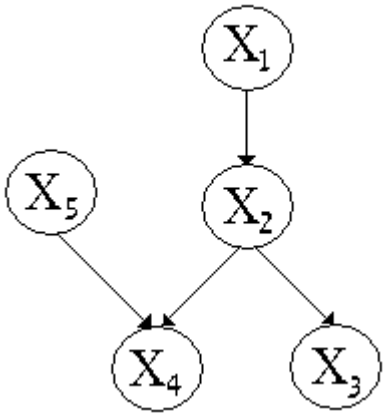

Figure 3.3: An example of the graphical structure of a Bayesian network

called the chain rule):

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | pa(X_i) \tag{3.14}$$

Consider an example of a Bayesian network with the structure depicted in Fig (3.3): the joint probability in this case, derived from (3.14), is given by

$$P(X_1, \ldots, X_5) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_2, X_5)P(X_5) \tag{3.15}$$

This joint probability and the graph that it represents are what one is trying to find for a given set of genes represented by the nodes (variables) of the graph.

Consider the three different connections that are present in the example graph:

Serial Connection There is a serial connection between X_1, X_2, and X_3. Information about the state of X_1 will affect our knowledge about both X_2 and, through X_2, X_3. However, if the state of X_2 is known for certain, then information about X_1 no longer has an influence on our knowledge of X_3 and vice versa. That is, when X_2 is *instantiated*, X_1 and X_3 are conditionally independent. Thus, in a serial connection such as this, X_1 and X_3 are said to be *d-separated* given X_2.

Diverging Connection In a diverging connection, two children are d-separated given a common parent node. There is a diverging connection between nodes X_2, X_3, and X_4. That is, knowledge is transmitted through X_2 between X_4 and X_3 unless X_2 is

instantiated.

Converging Connection Nodes X_5, X_4, and X_2 form a converging connection. In the case of a converging connection, there is no transferal of information between parents through a node *until* that node is instantiated. In this type of connection, the parents of the hub are conditionally dependent.

It is now possible to outline the properties of the example Bayesian Network of Fig (3.3). Let $I(A; B|C)$ denote that A is conditionally independent of B given C. Then, keeping property (3.10) in mind:

- The joint probability of the network is given by Equation (3.14).
- The independencies of the network are

  $I(X_1; X_5)$ $\quad$ $I(X_2; X_5|X_1)$

  $I(X_3; X_5, X_4, X_1|X_2)$ $\quad$ $I(X_4; X_1, X_3|X_2, X_5)$

  $I(X_5; X_1, X_2, X_3)$

With these two points, one can describe the interactions between nodes in the network. For the application considered in this study then, these describe the connections between genes in a genetic network model.

A final point about independence statements in Bayesian networks: if two graphs have the same set of independencies, then they are said to be equivalent. This point is important in that equivalent networks are indistinguishable using the scoring method described below. Thus the search for a model is actually the search for an equivalence class of models. The correct graph within the class would still need to be determined subsequently.

3.3.2 Training Bayesian networks

The goal of training a Bayesian network to experimental data is to find a model whose structure and set of probabilities are best able to reproduce the data. It must also be kept in mind that the collected data represents just a sample of the actual distribution of data that must be modeled by the chosen network. Mathematically, the problem is to search for the model, M, with the highest probability of being correct given the data, D. Thus $P(M|D)$ must be maximized. $P(M|D)$ is evaluated using Bayes' rule. The *Bayesian score*

is obtained by taking the logarithm of both sides of Equation (3.13) to get:

$$\begin{aligned} Score(M) &= \log(P(M|D)) \\ &= \log\Big(\frac{P(D|M)P(M)}{P(D)}\Big) \\ &= \log(P(D|M)) + \log(P(M)) - \log(P(D)) \end{aligned} \tag{3.16}$$

The third term of the righthand side of Equation (3.16) is the log of the prior distribution of the data. Since this value is independent of the model, it plays no role in the search for an optimal M. Also, one may assume at this stage that all models are equally probable, which means that P(M) are all equal. Thus, the key to the scoring metric is the term $\log(P(D|M))$ which represents the logarithm of the *likelihood* of a model given the data [24].

The task of calculating the likelihood is less than straightforward and the method chosen to perform this task depends on the state of the available information about the network. Keep in mind that in order to define the model, both the structure of the graph and the probability distribution, as in Equation (3.14) must be chosen. For example, if the structure of the graph is already known and the experimental data set is complete (that is, there are no latent variables or missing data points in the data set), then a maximization of the likelihood is relatively straightforward. However, if the graph is unknown or there are latent variables included, then graph fitting or estimation-maximization (EM) methods respectively are attempted [39].

In the most difficult case (the case of interest here), where both the structure is unknown and the data incomplete, several approaches may be attempted. In a fairly small network, EM methods combined with graph searches may be sufficient, possibly with the help of some stochastic sampling, as described below. For larger networks, the exact solution of such a problem is not a feasible option computationally. There are examples of networks where exact inference is indeed possible, but for which the calculation time of such a solution is prohibitive on today's computers. [22] Several methods of finding an approximate solution exist. These methods fall mainly into three categories:

Partial exact evaluation In this method, the exact inference is calculated, but some variables are held fixed.

Variational approximation This method applies the Law of Large Numbers to dense graphs. Probabilities that are difficult to calculate are approximated using the means of the distributions. This method is recommended by Hartemink [17] as a good choice for large biological networks. However, I question whether large networks are sufficiently dense for the approximation to be valid, since it seems that an increase in the number of nodes (for a protein network at least), would not correspond to a significant increase in the number of parents and children for each node.

Stochastic sampling Stochastic methods such as Importance sampling and Markov Chain Monte Carlo methods [13] involve random sampling from a simpler distribution to approximate the results of using the actual distribution. These methods tend to be slow for larger networks, but generally yield good results.

3.3.3 Hartemink case study

Hartemink *et al* (2001) [17] use Bayesian networks to study genes regulating galactose metabolism in *S. cerevisiae*. The application involves using the Bayesian scoring metric to compare two competing hypotheses of regulation. The two hypotheses concern the causal relationship between the gene Gal4 and its gene product and the product of the gene Gal80. The issue that differentiates the hypotheses is whether Gal80 regulates Gal4 at the transcription stage or posttranslationally. In the former case, there would be a causal link between Gal80 protein and the Gal4 expression, while in the latter case Gal80 would affect the Gal4 protein.

For the network formulation of the hypotheses, the genes are information variables since expression values are available from 52 Affymetrix GeneChip oligonucleotide arrays. It is notable that none of the gene expression assays were initially performed to target the specific problem being posed. This means that although the data is not selected or biased towards finding a specific result, the Bayesian network is capable of determining the underlying model. The state of the protein products in this example are included as latent variables in the network.

The gene expression data is converted to binary values. Although the use of variables that take on continuous values is possible with Bayesian networks, quantization is more in cases where only a small number of data sets are available.

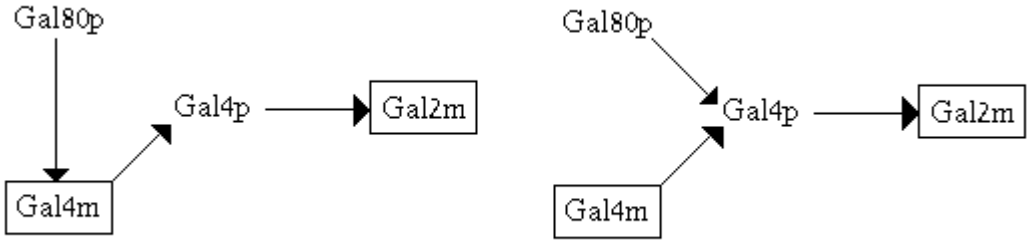

Figure 3.4: The two hypotheses being compared using Bayesian scoring. The model on the left represents an outdated hypothesis and that on the right represents the currently accepted system. The genes, whose values are measurable, are shown in boxes, while the gene products are included in the model as latent variables.

The results of the model scoring are positive: the currently accepted hypothesis, that Gal80 acts on Gal4 posttranslationally, is calculated to be more likely than the previously accepted hypothesis. Both hypotheses are shown in Figure 3.4. To support these results, all possible nonequivalent models of the connections between Gal80, Gal4, and another gene, Gal2 are compared. Those models with a link between Gal80 and Gal2 score higher than those without, which suggests that Gal80 and Gal2 are not conditionally independent given Gal4. This is in keeping with the successful hypothesis.

3.3.4 Annotated edges

With the method outlined above, it is possible to obtain information on likely dependencies between elements of the system. A direction is associated with each edge of the net, however there is no further clue as to the type of dependency the edge between the two variables represents. Thus, annotations are assigned to each edge as follows:

- $X \rightarrow Y \Rightarrow$ arbitrary dependence of Y on X.
- $X \rightarrow^{(+)} Y \Rightarrow$ the probability of Y being "on" is greater when X is "on" than when X is "off", independent of the values of other parents of Y.
- $X \rightarrow^{(-)} Y \Rightarrow$ the probability of Y being "on" is greater when X is "off" than when X is "on".

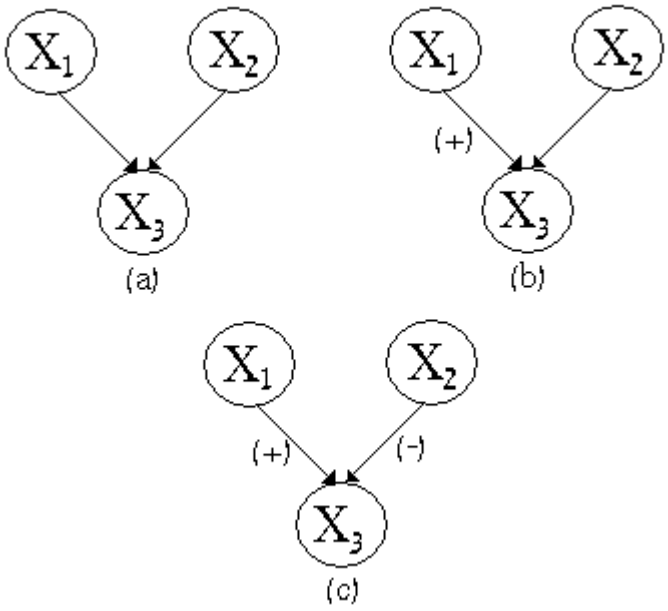

Figure 3.5: Three possible annotated Bayesian models. The question raised in the text is whether (a) would receive a worse score than (b), given that (c) is the true model.

- $X \rightarrow^{(+/-)} Y \Rightarrow$ the dependence between X and Y is not arbitrary, but the correct designation is undetermined.

The modifications to the scoring method to accommodate the annotations involve considering only those distributions that satisfy the annotation constraints in calculating the likelihood of the model.

The results of adding annotations are not completely positive. In fact, the annotation (-) of the edge between Gal80 and Gal2 resulted in very poor scores, despite the fact that this annotation correctly describes the true biological nature of the relationship. However, these results are explained by the authors by the fact that a protein, that is responsible for neutralizing the repressive role of Gal80, whose identity is not currently known, is missing from the network. This example clearly presents the danger in making detailed conclusions about the nature of the dependencies between components of a system without all the components being present.

The method of adding annotations to a graph is not explicitly defined. For a relatively large graph, this task could mean a significant addition to an already expensive

procedure. It is suggested that an unannotated graph could be used as a starting point and annotations added incrementally. It is not immediately obvious that the addition of a correct annotation will receive a higher score. This fact is reinforced by the results of the case presented above. For example, in the figure below, would (a) receive a worse score than (b) if the true system corresponds to graph (c)? That is, is it better to have an annotation added which is correct, but may result in a negative score from the lack of a balancing effect that has not yet been annotated? The answer to such a question would impact the effectiveness of adding annotations incrementally.

In short, the idea of being able to represent more detail is an admirable one, but the best way to achieve this end, if it can be achieved in a Bayesian network framework, is debatable.

3.3.5 Dynamic Bayesian networks

Biological systems are not static systems, but are changing constantly in time. Modeling methods for studying these systems that include this temporal component are needed. Dynamic Bayesian network (DBN) is a general term for a class of models that include Hidden Markov Models (HMM) and linear dynamical systems. The idea of a DBN is to follow a given graphical structure through different time points.

To get a picture of the concept of a DBN, imagine folding a piece of paper several times and cutting the folded paper into the shape of a gingerbread man to get a chain of paper figures. Unfolding the chain one figure at a time is analogous to "unrolling" a DBN model. Each figure is a "slice" or graphical representation of the system at a given point in time, but the structure of the graph in each slice is identical. It is the manner in which the figures of the chain are connected to each other (holding hands, say) and the template structure of all of the figures that define the differences between various models.

3.3.6 Comments on the study of gene expression data using Bayesian networks

As noted above, there are many features of biological systems and the experimental data that is collected from these systems, that recommends the use of Bayesian networks in their study. In the case study by Hartemink *et al* of the regulation of galactose metabolism, the data used to fit the model was not collected with the purpose of differentiating the

competing hypotheses considered. However, it is possible that data collected with this purpose in mind would provide more conclusive results, because the data set could be targeted to provide a better representation of the actual distribution of data. If this is the case, then an agreeable system could be established of running experiments, using the Bayesian networks to elucidate portions of the system and subsequently to perform more assays suggested by the results of the analysis to gain more conclusive results. Hopefully, this would facilitate faster convergence to a realistic network model of the real biological system.

There are still several problems to overcome. New methods of computing the likelihood are being researched. The problem of the loss of information due to the quantization of the gene expression values is a recurring theme of this report. The systematic generation of a comprehensive and significant set of models to score is a necessity, and some progress in this regard has been made [56]. As well, one needs to address the evaluation of models containing cycles, since such models are common to biological systems.

3.4 Discussion

The methods presented in this chapter have different strengths and weaknesses. REVEAL is relatively easy to understand and to implement. Unfortunately, at this stage in its development it uses a binary quantization of the data, which results in a less detailed description of the nature of the interaction between genes. Also, the choice of a threshold for the quantization is not a simple task and if done correctly requires expert knowledge of the system under study. This is definitely a drawback considering the nature of the problem and the size of genetic networks.

The weighted matrix method improves on REVEAL in that it does not require quantization of the data and does not place a limit on the number of genes that determine a given gene's expression. Further, it can be trained on a relatively small data set [55]. However the method is not as clear in determining the connections between specific network components, since the network of interactions is ultimately represented by a large matrix of real numbers. This makes the interpretation of results much more difficult.

The Bayesian method seems very promising. Its flexibility and probabilistic nature are definite advantages over the other two methods. The main problem with the Bayesian

method is that, in my opinion, it would be difficult to apply to a network of the size encountered in signal transduction pathways or genetic networks. However, if parts of the graph structure or probability distribution are known in advance, perhaps the difficulty of the task of training the networks can be reduced.

Chapter 4

A Reverse Engineering Application

This chapter outlines the application of a reverse engineering method to the data created from the EGF simulation described in Section 1. The method is based on an approach used by Maki *et al* (2001) [35] for the elucidation of genetic networks. The particular appeal of this method is its "divide and conquer" methodology, which appears to be a useful approach for studying large systems such as signal transduction pathways.

The work of Maki *et al* is discussed in Section 4.1. The next two sections each present one of the steps of the reverse engineering. Sections 4-7 cover the application of the method to the EGF simulation data. The chapter ends with a discussion of the results.

4.1 A system of inference of large-scale genetic networks

The application discussed in this chapter is derived from work by Maki *et al* that has been presented in several conferences in the past few years ([35], [52]). Its intended application is the inference of the structure of genetic networks from DNA microarray data. The aim of this application is to extend the principals of the approach to the series of chemical reactions that form a signal transduction network.

Before the method is applied, a specific set of data must be collected: a deletion[1] of each component of the network is made and the resulting steady-state and time-series data collected for the rest of the network components. At this point, the following analysis is performed:

[1]The deletion of a gene is the removal of the part of the chromosome corresponding to that gene. Thus the action of the system without that genetic component (and its gene product) can be studied.

Part One: Static Boolean Network

The steady-state data is quantized to binary data to reflect the interactions between components in the network. This data is used to place the network components into strongly connected groups. Each group is treated as a node in a directed graph whose structure is then determined.

Part Two: Determination of Subnetworks

The components within a group have been so grouped because the fast Boolean approach of Part One is not sufficient to identify connections between these components. The second stage involves determining the network structure *within* each group. A slower but more powerful method is used to determine the structure of the graph within each group using a best fit of a differential equation model to the time series data. The parameter fit is accomplished by minimizing the squared errors between the experimental and calculated values of components using a genetic algorithm optimization.

Neither of the above steps would be sufficient on its own, but each has specific advantages for solving parts of the overall problem. The Boolean method does not have the level of detail needed to determine connections between all the components of the network, but it is fast enough to deal with a large network. The optimization of parameters of a differential equation model is capable of deciphering feedback loops and other connections that can't be determined by the static network approach, but it is too detailed a method to apply to the entire network for a problem of this size. Thus, the combination of these methods allows the positive qualities of each method to shine.

4.2 Part One: Static Boolean network

Before beginning the reverse engineering of the EGF pathway, the steady-state and time series data need to be simulated. Since the goal of this data set is to determine which proteins are affected by the removal of a specific protein from the system, I simulate the data set by fixing the specific protein's concentration to a very small value (1×10^{-5}). Experimentally, the deletion of a protein would likely be accomplished by deleting the corresponding gene. The deletion of a protein complex might be accomplished by the specific inhibition of the complex. The plausibility of generating this data experimentally is debatable, especiialy since the deletion of some proteins will kill a cell.

Consider a network of p proteins whose network structure is to be found. The goal is to represent the network by a directed graph. The nodes of the graph are the p proteins of the system. The edges between nodes represent chemical reactions between the proteins. For instance, if there is a directed edge between nodes A and B, then protein A is a reactant in a reaction that produces protein B. With each edge there are one or two kinetic constants associated that determine the rate of the reaction. For instance, if the reaction between A and B is $A \rightleftharpoons B$, then there are first order rate constants associated with the production of B and of A (the forward and reverse reactions respectively). In this first step, the aim is specifically to determine the structure of connections between specific groups of proteins and not the determination of the kinetic constants.

The following steps in Part One of the method are illustrated in Figure: 4.1.

Step 1: The steady-state data is organized into a matrix D such that d_{ij} is the steady-state concentration of the j^{th} protein due to the deletion of the i^{th} protein.

Step 2: The relative concentrations of the deletion runs to the normal concentration levels are calculated for each protein. This is done by dividing each column, i, of D by the normal steady-state level for protein i. These values are compiled in a $(p \times p)$ relative intensity matrix, E.

Step 3: The next step is the quantization of the data in E to create a binary matrix, R. A value of 1 in the matrix means that the expression of a protein changed more than a specified threshold in response to the deletion of another protein. For a chosen threshold value θ the matrix R is defined by

$$r_{ij} = \begin{cases} 1 \text{ if } (e_{ij} > \theta \text{ or } e_{ij} < 1/\theta) \text{ and } i \neq j \\ 0 \text{ otherwise} \end{cases} \tag{4.1}$$

At this point, there are two things to accomplish:

1. Identify strongly connected groups
2. Remove indirect connections

Step 4: The first task above is to find the groups of proteins that are indistinguishable from each other based on the binary matrix. First, take the transitive closure, R^*, of

the binary matrix, R. R^* is defined by

$$R^* = \bigcup_{n=0}^{\infty} R^n \tag{4.2}$$

where

$$R^0 = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases} \quad \text{and} \quad R^{n+1} = R^n \circ R \tag{4.3}$$

Here

$$(R^n \circ R)_{ij} = \min\left(1, \sum_k r^n_{ik} \cdot r_{kj}\right) \tag{4.4}$$

So, R^* is a matrix that has filled out all the possible pathways in the graph. That is, if the deletion of Protein A affects Protein B, then the deletion of Protein A affects all the proteins affected by Protein B as well. So, R^* has values of 1 in row A for not only Protein B, but all the proteins affected by Protein B. In effect, R^* captures all reaction cascades in the network. Notice in Figure 4.1 that R^*_{13} has been changed to a value of 1 since $R_{12} = 1$ and $R_{23} = 1$.

Now find the strongly connected groups (or equivalent sets) in the graph as follows: construct a matrix of "equivalence relation", ER, such that an equivalence relation exists between components A and B if A affects B and B affects A [35]. That is

$$er_{ij} = \begin{cases} 1 \text{ if } r^*_{ij} = 1 \text{ and } r^*_{ji} = 1 \\ 0 \text{ otherwise} \end{cases} \tag{4.5}$$

An equivalence set of the i^{th} protein, denoted $[i]$, is then defined by the set $\{j | er_{ij} = 1\}$, imposing the restriction that each protein can belong only to one equivalence set. From this definition of the equivalence sets, notice that any cycles in the graph will appear within the same equivalence set. These are the features of the graph that the Static Boolean step is unable to distinguish.

At this stage, matrix R^* is reduced to represent the graph between the equivalence sets, rather than the individual proteins. In Figure 4.1, notice that proteins B and D are in the same equivalence set. Therefore, the column and row of the matrix corresponding to Protein D are eliminated, since they are identical to those of Protein

B with respect to all proteins outside of the equivalence set. In other words, all redundancies of the binary matrix are eliminated.

Step 5: The second task listed above is to eliminate values in the binary matrix that represent indirect connections in the graph. This is the final step shown in Figure 4.1. Notice that [1] affects [2] and [3] but that [2] affects [3] as well. This means that [1] affects [3] indirectly through [2]. Therefore, $R_{13}^{*'}$ is set to zero. Repeating this reasoning for all the proteins in the network results in the skeleton matrix, S, that contains only first degree connections.

To accomplish this result, perform a topological sort of the graph. A topological sort orders the nodes of a directed acyclic graph according to the degree of each node [44]. So, if node X has no parents, it has the lowest degree and is first in the ordering. Of all the descendants of X, the node(s) with the next lowest degree (those that are directly connected to X) appear next in the ordering. Thus all the direct connections in the graph can be determined.

This ordered graph represented by S is the output of Part One. At this point we have identified the connectivity in the graph of equivalence sets.

4.3 Part Two: Determination of subnetworks

In this step, the goal is to identify the connections between network components that have been grouped into equivalence sets. This section describes the method used by Maki *et al.* Section 4.6 describes departures from this method that are made in order to apply the method specifically to systems of chemical reactions among proteins.

4.3.1 S-systems

Maki *et al* represent the network of proteins within an equivalence set using a canonical nonlinear form of differential equations known as an S-system [20]. An S-system has the form

$$\frac{dX_i}{dt} = \alpha_i \prod_{j=1}^{n} (X_j)^{g_{ij}} - \beta_i \prod_{j=1}^{n} (X_j)^{h_{ij}} \tag{4.6}$$

where X_i is the i^{th} reactant or state variable among a total of n variables. The exponents g_{ij} and h_{ij} are real numbers representing "interactive effectivity" in S-system language or

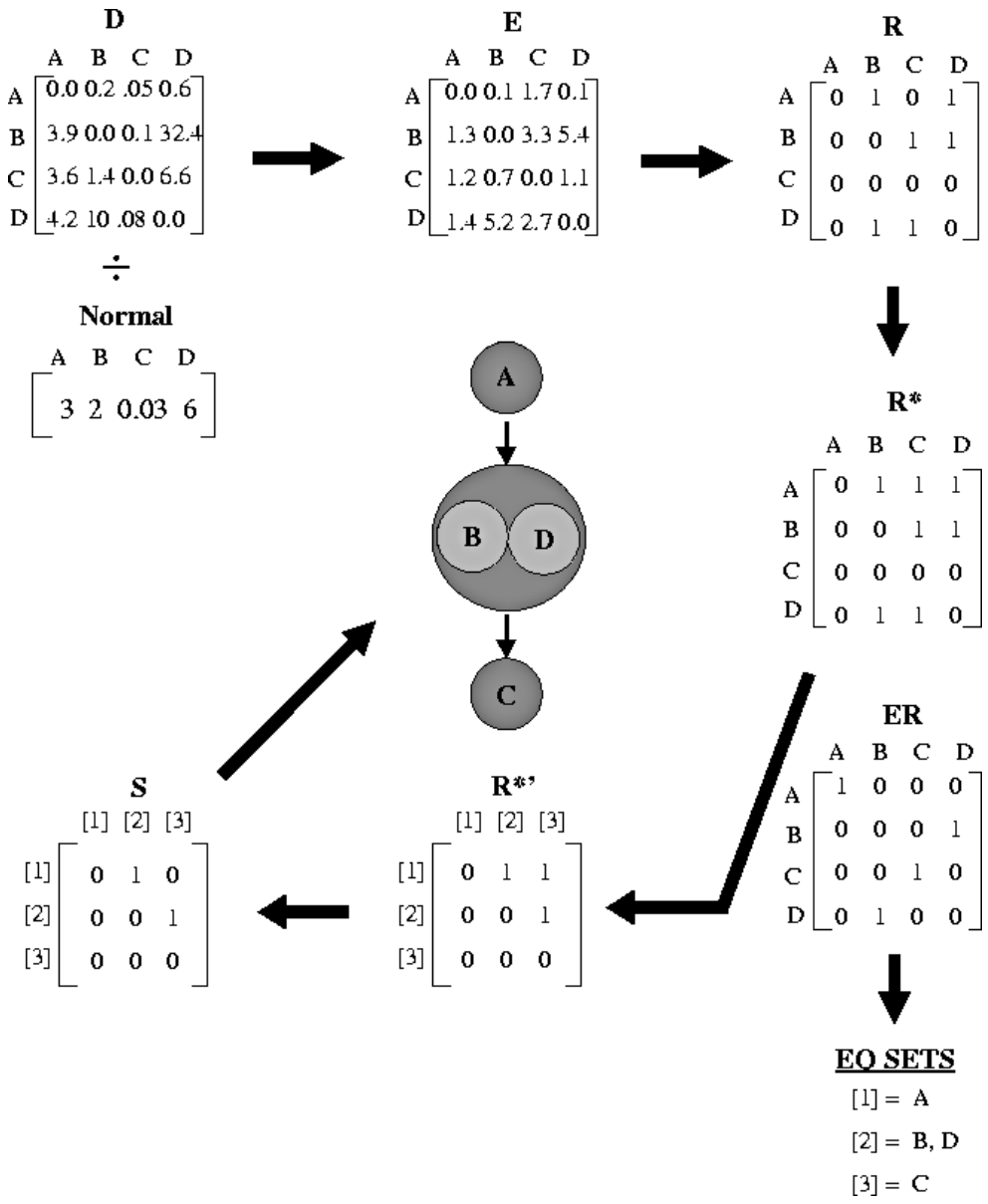

Figure 4.1: Part One of the method of Maki *et al* for a system of 4 proteins. Proceeding clockwise from the top-left: Each column of the data matrix D is divided by the normal steady-state value for that protein to yield the relative intensity matrix E. E is quantized to binary values with a threshold of 2. The closure of R yields R^*. The equivalence relations are determined: proteins B and D show the same behavior (in R^*) to all other proteins in the network and are therefore grouped together in [2]. Replacing columns B and D by [2], and relabeling A as [1] and C as [3] (single protein equivalence sets) gives us $R^{*'}$. Finally, the topological sort gives us the skeleton matrix S containing only direct connections between equivalence sets. This determines the shape of the graph, shown in the center.

"interrelated coefficients" with respect to genetic networks [35]. In a protein setting, one would compare them to the order of the reaction of X_j to produce X_i as in a mass action equation. The parameters α_i and β_i are used to denote the relative inflow and outflow of gene X_j [35]. For proteins these constants represent some sort of combined rate constant for all the reactions contributing (both positively and negatively) to the concentration of protein i.

Many differential equations can be "recast" as S-systems, including equations of elementary functions and compositions of elementary functions [46]. The advantage of recasting a system into this form is that there are efficient methods of solving S-systems using Taylor polynomials [20].

The main disadvantage of the S-system is that there are a large number of parameters: for a system of n state variables, there are $2n(n+1)$ parameters. Given that the goal is to fit these parameters to experimental data, this could be a significant problem, depending on the size of the equivalence set.

The strategy is as follows: each protein is represented by one of the variables in the S-system. The parameters are fit to the experimental data by minimizing the squared differences between data and output of the S-system. In the course of the minimization, if a given parameter becomes smaller than a designated threshold, the parameter value is set to zero. This corresponds to the loss of an edge in the graph of the protein interactions. Thus, the resulting parameter set defines the optimal network fit to the data. The method of optimization is described in the next section.

4.3.2 Genetic algorithm optimization

Genetic algorithms are a class of stochastic optimization techniques used in solving a wide range of problems. The stochastic nature of a genetic algorithm makes it a good choice for problems with many local maxima and minima, as the process is less likely to become trapped in a local extremum.

The conceptual basis of genetic algorithms is drawn from genetics and evolution. The idea is to create a population of individuals and evolve the population while applying the principles of natural selection. The parameters to be optimized constitute the genome of the individual and the objective function evaluated for those parameters defines the fitness of that individual. At each iteration (generation) of the algorithm, the fitness of all

individuals in the population is evaluated. This fitness value determines the probability of a given individual reproducing and thus propagating its genetic makeup through to the next generation. The final population is then theoretically the most fit population for survival in its mathematical environment.

The basic steps of the algorithm are as follows:

1. **Create an initial population**

   - The parameter values that represent individuals are usually randomly chosen to create an initial population. However, given that subsequent generations will evolve from this initial generation, it can be useful to start with some reasonable guesses amongst the randomly generated ones [19].
   - Representation for individuals is chosen. The structure and alphabet used to code the genome influences the genetic algorithm implementation. For a discussion of binary versus float representation and efficiency see [37].

2. **Evaluate the fitness of each individual in the population**

   - The fitness is the value of the objective function (the function to be optimized) given the set of parameters that constitute that individual.

3. **Breed the members of the current population to create the next generation**

   - The *selection* of parents from the current population and their subsequent *reproduction* are the aspects of genetic algorithms that vary most.
   - The selection is a probabilistic method based on the principles of natural selection: the higher the fitness of an individual, the more likely it is to survive and propagate.
   - The reproduction can involve many different operators, most of which also have a probabilistic element. For example, a simple reproduction function would involve a crossover rate to create new individuals with parts of their chromosome taken from each parent and a (generally smaller) mutation rate to randomly alter parts of a progeny's genetic code [9]

4. **Repeat steps 2-3 until a termination criteria is satisfied**

- Common termination criteria include reaching a maximum number of iterations or finding a solution within a given threshold. (This threshold could be the change from the previous generation's solutions or a projected optimal fitness level, say).

There are many variations within this common framework for a genetic algorithm. For the application of the method of Maki *et al* to the EGF simulation data, we use a publicly available MATLAB implementation, called the Genetic Algorithms for Optimization Toolbox (GAOT) [19]. The specifics of the algorithm used in this toolbox are described in Section 4.5.

4.4 Application of Part One to the EGF simulation

All programs used for this application have been created in MATLAB and are included in Appendix B. Using Gepasi, the steady-state concentrations for the proteins in the full EGF system (see Chapter 1) and for each deletion in that system are recorded. The relative intensity matrix, E is calculated as described in Section 4.2.

The next step is the calculation of the binary matrix, R. The range of values for each protein are shown in Table 4.1. From the table, it is evident that the range of values is very large, even when considering the values from only one protein. This fact makes it difficult to chose an appropriate concentration threshold value. After some experimentation with different threshold values, I carried out Part One using thresholds of 2 and 5. (Note that the thresholds have no associated units since the relative intensity values are dimensionless).

An example of the binary matrix corresponding to a threshold of 5 is shown in Table 4.2. However, when the equivalence sets are calculated using programs created in MATLAB (according to the procedure described in Section 4.2), the results indicate that there are only 3 equivalence sets. The proteins in each equivalence set are shown in Figure 4.2. The equivalence sets are shown in their place in the EGF pathway in Figure 4.3. The groupings do not seem to correspond to any of the cycles in the system and seem to group too many proteins together by comparison to the groups found in Table 4.3. Similarly, a threshold of 2 resulted in only 2 equivalence sets.

I believe that the origin of the problem may be in taking the closure of matrix R to get R^* (Step 4). The EGF test case has coupled cycles, and they are likely causing some

Protein	Maximum relative intensity	Minimum relative intensity ($\times 10^{-6}$)
1	1.17	0.02
2	167.22	0.57
3	1.06	0.57
4	1.13	0.00
5	1.14	0.00
6	600.00	0.10
7	1.04	0.00
8	1.10	0.00
9	1.0000	0.00
10	1.0000	0.00
11	4.1206	0.37
12	2.6355	0.00
13	19.1011	0.36
14	8.9479	0.00
15	15.9763	0.35
16	132.4561	0.20
17	1.6238	0.00
18	1.8656	0.00
19	1.7718	0.00
20	1.6446	0.00
21	1.6449	0.00
22	1.0147	0.00
23	1.0000	0.00

Table 4.1: Maximum and minimum relative intensities for each protein in the EGF simulation.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
EGF	0	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
R	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
Ra	0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
R2	0	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
RP	0	1	1	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
PLC	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
R-PL	0	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
R-PLP	0	1	1	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
PLC-P	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
PLCP-I	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRB	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	1	1	1
R-G	0	1	1	1	1	1	1	1	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1
SOS	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
R-G-S	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1
G-S	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1
SHC	0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1
R-SH	0	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	1	1	1	1
R-SHP	0	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
SHP	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1
R-SH-G	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1
SH-G	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	0	1	1
R-SH-G-S	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
SH-G-S	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0

Table 4.2: The binary matrix, R, corresponding to a threshold of 5. The rows correspond to the proteins that have been deleted to produce the given effect on the protein in each column. The numbers correspond to the proteins listed in Table 2.5

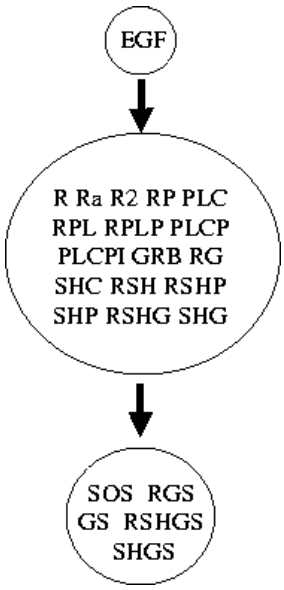

Figure 4.2: The proteins in each equivalence set and the graph of the equivalence sets.

problems in Part One that were not seen in previous applications of this method. That is, too many connections exist between proteins in the pathway, so that upon taking the closure of R too many proteins are grouped into the same equivalence set.

If the equivalence sets of the binary matrix R (Table 4.2) are found, rather than the equivalence sets of R^*, the outcome better. The equivalence sets can be found by grouping together proteins that have the same sequence of responses, with the exception of their behaviour towards other members of the set. The resulting 10 equivalence sets are shown in Table 4.3. If I colour the proteins in the pathway based on the equivalence sets found by inspection of matrix R the results make much more sense, as seen in Figure 4.4.

4.5 Genetic algorithm toolbox for MATLAB

Part Two is a genetic algorithm minimization. The source files for the Genetic Algorithm Toolbox are available publicly, so these are the starting point for the genetic algorithm implementation.

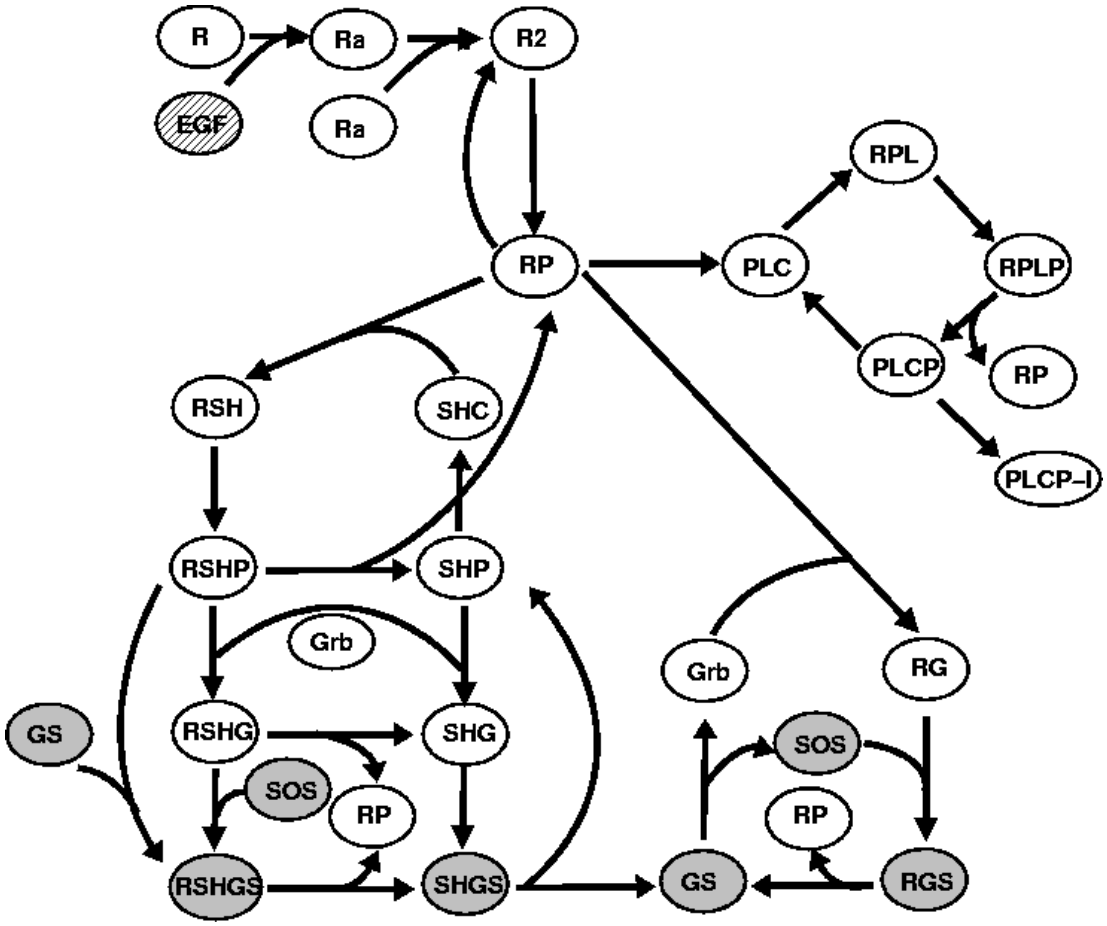

Figure 4.3: Proteins in the EGF pathway are coloured according to their equivalence sets.

	Proteins in Group
1	EGF
2	R Ra R2 RP
3	PLC RPL PLCP RPLP PLC-I
4	GRB
5	RG
6	SOS
7	RGS GS
8	SHC RSH RSHP SHP
9	RSHG SHG
10	RSHGS SHGS

Table 4.3: Manual grouping of proteins into equivalence sets based on matrix R with 5 threshold.

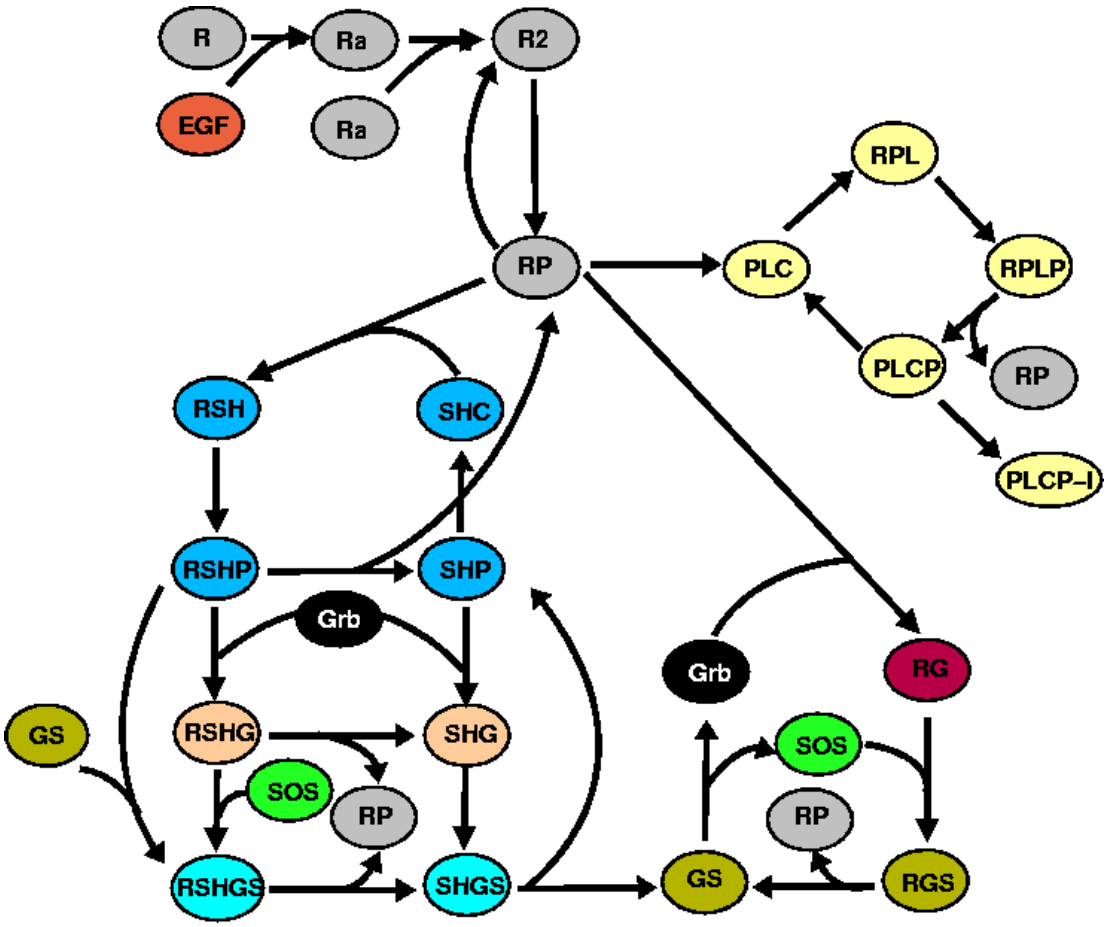

Figure 4.4: Grouping of the EGF pathway proteins according to the equivalence sets listed in Table 4.3

4.5.1 The evaluation function

To use the toolbox, an evaluation function must be provided to calculate the fitness of individuals. The objective function for this particular application is the squared difference between the experimental (simulated) protein concentrations and the protein concentrations predicted from the differential equations with the specific individual's parameter set:

$$f = \sum_{i=1}^{n} \sum_{t} (X_{calc,i,t} - X_{expt,i,t})^2 \tag{4.7}$$

where i sums over the number of proteins, n, in the network (equivalence set) and t sums over the time series.

Due to the decision not to use the S-system form of the differential equations (discussed in Section 4.6) I cannot use the same method of solving the differential equations as Maki *et al* [35]. Instead, I use the MATLAB function *ode23*, a low order Runga-Kutta numerical differential equation solver, to calculate the protein concentration values, $X_{calc,i,t}$,

The evaluation function takes the parameter set of an individual as input and outputs that same parameter set and the value of the objective function for that parameter set. The evaluation function carries out the following steps:

1. Read the file with the simulation data.
2. Solve the system of differential equations for the parameter set using *ode23*. The simulated protein concentrations at time $t = 0$ are used as the initial conditions for the numerical solution.
3. Calculate the value of the objective function using Equation (4.7).

4.5.2 Selection and reproduction

The genetic algorithm toolbox (GAOT) offers several different options for selection and reproduction. This application uses the default normalized geometric selection. In this type of selection, each individual in the population is ranked using a geometric distribution normalized according to the probability of choosing the most fit individual. An individual's ranking determines the probability that they are chosen as a parent.

Given the decision to use a real number representation rather than a binary representation for the genome of an individual, the genetic algorithm toolbox offers a plethora of

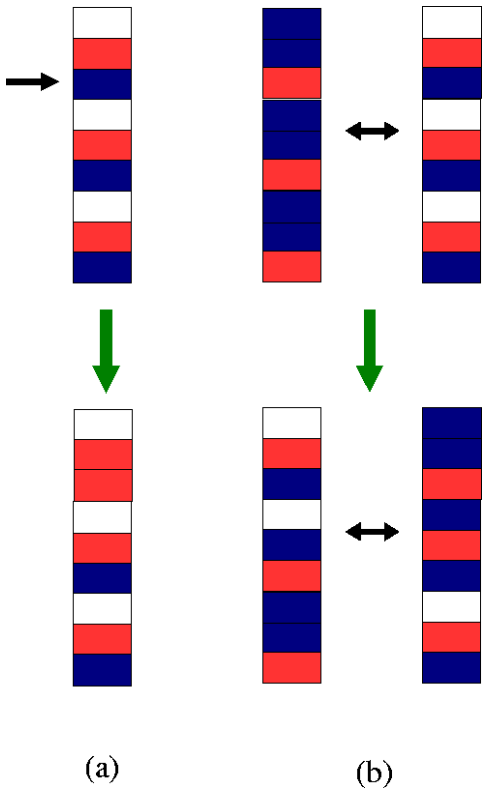

Figure 4.5: Demonstration of mutation and crossover operators using LEGO. (a) In mutation, a single block from one tower is randomly selected (black arrow) and replaced with a different block in the child. (b) In arithmetic crossover, a detachment position along the tower is randomly selected (black arrow). Both parent towers are broken at that position, and the top halves of the tower are interchanged to yield the two children.

reproduction operators: uniform mutation, non-uniform mutation, simple crossover, arithmetic crossover, and others [19]. The mutation operators have in common the feature that a parameter is randomly selected from among all the parameters of all the individuals. The difference lies in how the parameter is changed. For instance, the new value could be a number randomly chosen within the bounds for that parameter (uniform mutation) or set equal to the upper or lower bound (boundary mutation) or chosen from within the bounds with a non-uniform probability (nonuniform mutation).

The crossover operation involves an exchange of values between two individuals to produce two offspring who partition the genomes of their parents between them. To picture this process, imagine the set of parameters that make up an individual as a tower of LEGO blocks of different colours. A mutation involves randomly choosing one block from a tower

and replacing it with another block (possibly of a different colour). The different mutation operators describe the manner in which this new block is chosen and inserted in the tower. A common crossover operation would be analogous to breaking the two LEGO towers at the same point along the tower and interchanging the pieces of same length between the two towers. (Thus, each of the resulting towers contains a section from each parent tower.) For this application to protein networks, each block is a biochemical parameter value and a tower is a parameter set. Reproduction involves exchange of values between hypothetical sets of parameters.

4.5.3 Termination

The toolbox also offers several options for termination. One can stop at a specified maximum number of generations or when the fitness value is within a given epsilon of a known optimal fitness value. For this implementation, I specify a maximum number of generations and an optimal error value. The optimal error value was determined by using the evaluation function to determine the fitness of the parameter set corresponding to the actual solution. In all runs, however, the maximum number of iterations was reached before the optimal fitness value. I experimented with the number of generations to find the most successful number for the application to the EGF pathway, typically using between 1000 and 10,000 iterations.

4.6 Modifications to the method

In this application of the method of Maki *et al* to signal transduction pathways, I have made several changes. The first major change is the decision to not use the S-system formulation and instead to use a generalized mass action form:

$$\frac{dX_i}{dt} = \sum_j \left(\alpha_{ij} \prod_k (X_k)^{g_{ijk}} \right) \tag{4.8}$$

where the parameter α_{ij} is the real number rate constant (whose order is determined by the associated product) for the j^{th} term contributing to the rate of change of the concentration of the i^{th} protein. The parameter g_{ijk} is the positive integer exponent for the k^{th} protein of the j^{th} term. These powers are derived from the stoichiometry of the rate-determining

step of the reaction in which the term is involved.

The main differences between Equation (4.8) and the S-system form are:

- The summation of terms of products of proteins and their associated rate constants.
- The second decay term is absorbed into the one term by not restricting α_{ij} to positive real numbers.
- The powers are restricted to integers less than or equal to 2, since here they represent kinetic orders.

It is known that the S-system is a feasible representation for a system of generalized mass action equations that describe a set of chemical reactions [46]. However, we question the use of the protein concentration data as the values of the state variables in the S-system for fitting the parameters. Consider: if one knew the structure of a network of chemical reactions one could write out the mass action equation describing the rates of change of the concentration of each protein.

At this point, one could recast this system of equations into an S-system format. However, the recasting of the mass action equations into S-system format requires the introduction of extra variables to account for the sums in the mass action equations [46]. For example, consider the equation

$$\frac{dX_1}{dt} = k_1 X_3 - k_2 X_1 X_2.$$

As it is, the equation is not in S-system form because of the sum of terms. This equation is recast as follows: let $X_1 = Z_1 Z_4$, $X_2 = Z_2$, and $X_3 = Z_3$. Then,

$$\begin{aligned} \frac{d(Z_1 Z_2)}{dt} &= k_1 Z_3 - k_2 Z_1 Z_4 Z_2 \\ \Rightarrow Z_2 \frac{dZ_1}{dt} + Z_1 \frac{dZ_2}{dt} &= k_1 Z_3 - k_2 Z_1 Z_4 Z_2 \end{aligned} \tag{4.9}$$

and the terms on the left are arbitrarily assigned to terms on the right:

$$\begin{aligned} \frac{dZ_1}{dt} &= k_1 Z_3 Z_2^{-1} \\ \frac{dZ_2}{dt} &= k_2 Z_4 Z_2. \end{aligned} \tag{4.10}$$

Thus, there are actually more state variables in the S-system recast of the mass

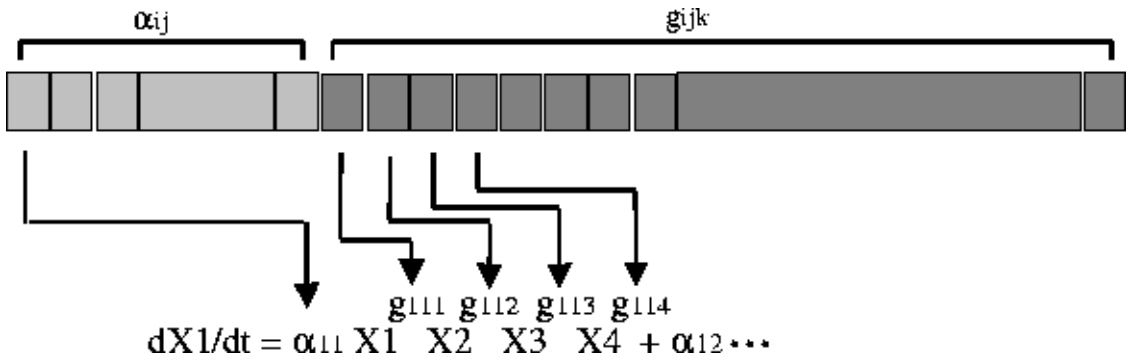

Figure 4.6: The organization of the genome of an individual. All α parameters appear first, followed by all g parameters. α_{ij} is the rate constant for the j^{th} term in the differential equation of the i^{th} protein. g_{ijk} is the power of the k^{th} protein in the j^{th} term in the differential equation of the i^{th} protein.

action equations for a given set of reactions. However, in the method set forth by Maki *et al*, there are the same number of state variables as network components. How then to interpret the fitting of the protein concentration data to the state variables? What do the parameters of the S-system represent?

I believe that despite an increase in the number of parameters, the generalized mass action equation is a more natural choice for a network of chemical reactions among proteins. This choice also leaves us with a set of biochemically relevant kinetic constants for the signal transduction pathway after the optimization, an improvement over the original method.

Given the decision to use Equation (4.8), the parameters that constitute an individual of the population are organized as shown in Figure 4.6. The ordering of the parameters in the genome is as follows:

$[\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1m}, \alpha_{21}, \alpha_{22}, \ldots, \alpha_{nm}, g_{111}, g_{112}, \ldots, g_{11n}, g_{121}, \ldots, g_{1mn}, g_{211}, g_{212}, \ldots, g_{nmn}]$,

where m is the maximum number of terms and n is the number of proteins.

For the remainder of the discussion, I shall refer to a term of the summation on the righthand side of one of the differential equations describing the rate of change of a protein's concentration simply as a term. The j^{th} term for protein i is defined by the kinetic constant, α_{ij} and n exponents g_{ijk}. The number of nonzero α_{ij}s tells us the number of terms for protein i, that is, the number of reactions that protein i is involved in (where forward and reverse reactions are separate reactions). In the application discussed here, we set an upper limit $m = n$ on the number of nonzero α. The index k of the nonzero

Parameter	Value
k_1	0.01
k_2	0.02
k_3	0.2
k_4	0.03

Table 4.4: Kinetic constants for the test network, Equation (4.12)

exponents g_{ijk} tells us which proteins are included in the term.

4.6.1 Adapting the genetic algorithm toolbox

In order to incorporate the changes described in the previous section, substantial changes to the Genetic Algorithm toolbox are required. The strategy of fitting the system of differential equations to the simulation data remains essentially the same. However, there are several problems that arise in trying to use a genetic algorithm optimization with the mass action equations. Specifically, the resulting optimal network must be biologically feasible.

To study this problem I create a test system of chemical reactions:

$$\begin{aligned} X_1 + X_2 &\rightleftharpoons X_3 \\ X_3 &\rightleftharpoons X_4 \end{aligned} \tag{4.11}$$

for which I simulate the time series data in Gepasi. Immediately, the large number of possible terms becomes obvious: for each protein, there are 2^n possible combinations of proteins that could contribute to the change in concentration when terms up to bimolecular kinetics are considered. A limit needs to be placed on the number of reactions in which each protein can participate to avoid huge computational expense as n increases. This is biologically reasonable, since most proteins in a pathway carry out a specific function with a specific set of substrates.

Another problem that becomes apparent after a few runs of the genetic algorithm, is that the basic features of mass action kinetics need to be incorporated into the optimization routine for the results to correspond to a series of chemical reactions. Consider the test

case above: the mass action equations corresponding to the network in actual fact are

$$\begin{aligned}
\frac{dX_1}{dt} &= -k_1X_1X_2 + k_2X_3 \\
\frac{dX_2}{dt} &= -k_1X_1X_2 + k_2X_3 \\
\frac{dX_3}{dt} &= k_1X_1X_2 - k_3X_3 - k_2X_3 + k_4X_4 \\
\frac{dX_4}{dt} &= k_3X_3 - k_4X4
\end{aligned} \tag{4.12}$$

and the rate constants are listed in Table 4.4.

Ideally, the algorithm should reconstruct these terms. In practice however, we have found that these are not the terms that the optimization yields. This is not entirely surprising, since the parameter space is very large and there are doubtless many differential equations that can reproduce the simulated concentration values within a reasonable error. The need to include further restrictions on the form of the solution could be an indication that another optimization method with specified constraints is needed to define the structure of the system. However, the genetic algorithm is still a very good optimization method for large search spaces and is reasonably flexible. Therefore, I have chosen to adapt the original genetic algorithm so that it can only output solutions corresponding to networks of chemical reactions.

To accomplish this task, the rules that dictate what terms appear in the equations must be identified and individuals of the population of possible solutions that don't obey these rules must be punished. The main difficulty in doing so, is that the form of the equations is determined by the chemical reactions. This present attempt, on the other hand, is to try to create differential equations that can represent a feasible set of chemical reactions without knowing even the specific reactants and products involved in each reaction.

The main principle that must be satisfied by the equations is that if a term kX_j appears in the differential equation for the rate of change of protein i, then the complementary term $-kX_j$ must appear in the differential equation for the rate of change of protein j. This ensures that the equations for both proteins register the fact that the reaction

$$X_j \rightarrow X_i$$

exists. If only one of these terms appears, then the differential equation does not actually

represent a chemical reaction and is not a valid solution.[2]

An additional requirement for a valid solution is that a negative term in the differential equation for the i^{th} protein must include that protein in the product. This comes directly from chemical kinetics.

Having identified these requirements, how can they be implemented? At each iteration of the genetic algorithm, new individuals are bred from the current population according to the selection and reproduction operators discussed in Section 4.5.2. So, in the case that a population of individuals can be generated that represent valid solutions according to the requirements above, a way must be found to ensure that the next generation of individuals are also valid. Given the stochastic nature of the reproduction operators, this is not an easy task.

The crux of the problem is this: when a crossover occurs, there is a fair probability that the crossover point will lie *within* one of the terms of the summation on the righthand side of the differential equation. Similarly, if a mutation occurs, it will alter a term. These changes must be reflected in the complementary terms appearing in the differential equations of the other affected proteins. Thus, the complementary terms must be located and changed to reflect the crossover or mutation.

Attempts to simply punish non valid solutions did not work, since the percentage of solutions that are valid out of all possible parameter combinations is extremely tiny. In other words, without further intervention, the genetic algorithm never really gets started because the randomly generated initial population doesn't contain any valid solutions from which to breed better solutions. Thus, the first task was to create a valid initial population.

4.6.2 Initiation of the population

In creating the initial population, some randomness must be retained, but the basic structure of each individual must also satisfy the mass action requirements. My solution is to create a bank of all the possible terms, given the number of proteins in the network and the integer bounds on g_{ijk}, and to distribute these terms randomly among the differential equations for each protein. An example of the contents of the termbank for the network of 4 proteins is

[2] In future, we would like to consider enzymes explicitly as components of a signaling network. Therefore, we do not wish to make assumptions about relative concentrations of network components that would lead to a Michaelis-Menten law formulation and provide an exception to the above rule.

Termbank
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1

Table 4.5: Contents of the termbank for a network of 4 proteins. For example, row 3 of the table corresponds to the term X_3X_4. The termbank has been sorted for insertion of a term into the differential equation of protein1 (the term is randomly chosen from the first 7 terms of the termbank). Notice that the term of all zeros is eliminated from the bank.

shown in Table 4.5.

The process of constructing the initial population is implemented in the program `initpop` as follows:

for each individual
  while all differential equations have less than n terms
    randomly select one of the n proteins
    randomly select a positive α within the specified bounds
    randomly select a valid term from the termbank
    add the term to the current protein's differential equation
    determine which complementary terms need to be added and add them
  arrange the parameters from the differential equations into the parameter solution for the individual in the form of Figure 4.6

To obtain a valid selection from the termbank, I restrict the random selection to only those terms that have a zero exponent for the protein to which the term is being added. This restriction is necessary because the rate constant for the term is positive. Why are only terms with a positive rate constant added? Consider the first reaction of the test case:

$$X_1 + X_2 \rightarrow X_3$$

The negative terms that appear in the differential equations for X_1 and X_2 (Equation (4.12)) do not tell us to which protein the arrow is pointing, which means that the placement of the complementary term is a mystery. However, if the positive term $k_1 X_1 X_2$ appears in the differential equation of X_3, then the negative complementary terms must appear in the differential equations of X_1 and X_2. This is the basis of the addition and deletion of complementary terms in `initpop` and also in the mutation and crossover operators discussed in the next two sections.

An additional feature in `initpop` is the ability to accept previously generated solutions as individuals in the population being created. This means that the best solution from a previous run of the genetic algorithm can be used to seed another run, and possibly to increase the fitness of the new population more quickly.

4.6.3 Mutations

Table 4.6 describes the different cases that result from a mutation and the action taken to ensure that the new solution remains valid. This action depends on the value, type, and sign of the current parameter and on the sign of the parameter value to be inserted in its place. Consider the term $0.1X_1^2X_2$. If, for example, the α value, 0.1, is chosen for mutation to a value of 0, then by row 3 of Table 4.6 the term is deleted (α_{ij} and g_{ijk} are set to zero) and the complementary terms in the differential equations of protein 1 and 2 are deleted.

In the case $g_{ijk} > 0, \alpha_{ij} > 0$, if the powers of all other proteins in the term are zero, then the term is deleted. This is done so that there is no "floating" kinetic constant without a term attached. Otherwise the kinetic constant would appear on its own as a term contributing to the differential equation of protein i.

4.6.4 Crossovers

Crossovers present a special challenge. The probability that a traditional crossover will produce two valid offspring is virtually nil. However, crossover is an essential component of the genetic algorithm. Thus, an entirely new crossover operator is needed. The aim of the crossover operation is to exchange genetic material between parents to create offspring that are potentially better solutions than the parents. The crossover operator must still contain a stochastic element so that the search space is well covered, however the fact that the children are built from parts of the more fit members of the current generation means

Current Parameter		New Parameter	Action Taken
α	$\alpha_{ij} > 0$	> 0	change α in term and in complementary terms
		< 0	no change
		$= 0$	delete current term and complementary terms
	$\alpha_{ij} = 0$	> 0	if there is room, create the new term and complementary terms
		< 0	no change
	$\alpha_{ij} < 0$	> 0	no change
		< 0	change α in term and in complementary terms
		$= 0$	delete current term and complementary terms
g	$g_{ijk} = 0, \alpha_{ij} = 0$	> 0	no change
	$g_{ijk} = 0, \alpha_{ij} \neq 0$	> 0	add new term to protein k if there is room, change current term and complementary terms
	$g_{ijk} > 0$, any α_{ij}	> 0	change current term and complementary terms
	$g_{ijk} > 0, \alpha_{ij} > 0$	$= 0$	if $g_{ijk} = 0\ \forall k$, delete term and complementary terms. Otherwise change the term and complementary terms and delete the complementary term for protein k
	$g_{ijk} > 0, \alpha_{ij} < 0$	$= 0$	same as previous, but change is made only if $k \neq i$

Table 4.6: A list of all the possible mutations and the action taken to perform the mutation in order to yield a valid solution. The Current Parameter column lists the type and the sign of parameter selected for mutation. New Parameter designates the sign of the value that is to replace the current parameter. For the mutations of g_{ijk}, the sign of the α_{ij} for the term that the exponent appears in is also listed. Recall that the subscripts refer to the k^{th} protein in the j^{th} term in the differential equation for protein i.

Operator	α_{ij} and g_{ijk} together	Replacement
amyXover1	no	yes, between children
amyXover2	yes	no
amyXover3	yes	yes, between children
amyXover4	yes	yes, no terms are deleted

Table 4.7: The first four operators created for performing the crossover for the adapted genetic algorithm optimization.

that there is a better than random chance that the next generation's individuals are closer to an optimal solution. Any new crossover operator should have these goals in mind.

Four crossover operators were initially created: `amyXover1-4`. All four crossover operators function in a similar fashion to `initpop`. The difference is that the valid terms for creating the two children are chosen from a bank of terms collected from the two selected parents. This is accomplished as follows:

- compile a list of unique terms from the two parents
- randomly distribute the terms and their complements to one child until the child is full
- fill the second child by picking from the remaining terms and creating the complementary terms

Thus the output is two valid children that have the parent's terms distributed among them.

The specific characteristics of each crossover operator are listed in Table 4.7. The two main differences between operators are their treatment of the kinetic constants, α_{ij}, and their renewal of the bank of terms. For instance, `amyXover1` creates two separate banks: one for the kinetic constants and one for the terms. Thus the α_{ij} are shuffled separately from the terms. This causes quite a disruption, and the resulting children may be quite different from their parents. In contrast, `amyXover2` distributes the kinetic constants with their terms.

Replacement refers to whether or not the bank of terms is replenished. `amyXover1` and `amyXover3` both renew the bank of terms such that both children are assigned terms chosen from a full bank of parental terms. `amyXover4` does not delete terms from the bank, thus the probability of choosing a given term is always the same. Distributing terms without

replacement is most in keeping with traditional crossover operators.

One problem with these crossover operators is that there is no ability to exchange material *between* terms. Currently, any changes to terms occur from the random mutations described in the previous section. However, mutation rates are generally kept low, since they are more likely to be destructive than helpful in bringing the current solution closer to an optimal solution.

4.6.5 The Comet-Strike feature

In addition to the basic operators discussed above, I add a "comet-strike" feature [28]. The idea behind the comet-strike is to introduce new individuals to increase the gene pool, renewing the population and thus avoiding stagnation. The comet-strike is implemented by wiping out the entire population except for the current best individual. This best solution is included in an otherwise completely new population and breeding resumes.

The comet-strike is used to combat the "inbreeding" witnessed with the new crossover operators. That is, as a population converges toward a minimum, the best terms appear in an increasing portion of the population. This means that the bank of unique parental terms to choose from can become too small to add enough terms to the differential equations of both children.

4.7 Results of Part Two

The adapted genetic algorithm described in the previous sections was applied to both the test system (Equation (4.12)) and the simulated EGF protein concentration time course data. Tests were also performed to assess the success of the four crossover operators described in Section 4.6.4.

4.7.1 Crossover tests

The four crossover operators were tested using runs of 100 iterations and a population of 10 individuals. The results are presented in Table 4.8. The first run of tests started all crossover operators with the same initial population (by starting with the same seed for MATLAB's random number generator). The second run started each crossover operator with a different initial population. In the first run, `amyXover2` and `amyXover3` yielded

Test Run	Operator	Error	Comments
Run 1	amyXover1	0.2676	Equations for 2 proteins have no terms 2 unique constants
	amyXover2	0.3900	All equations have terms, but they are the same 2 unique constants
	amyXover3	0.3900	All equations have terms, but they are the same 2 unique constants
	amyXover4	0.3920	All equations have terms, well distributed 2 unique constants
Run 2	amyXover1	0.3906	2 different terms, 3 different constants
	amyXover2	0.2239	4 different terms, 4 different constants
	amyXover3	0.3920	3 different terms, 3 different constants
	amyXover4	0.0550	2 different terms, 2 different constants

Table 4.8: Results of tests of the crossover operators amyXover1-4 with population size 10 and 100 iterations.

differential equations that all contained the same terms but with different constants. In the second run, however, `amyXover2` yielded the best results. The results are inconclusive. It appears, from longer runs using combinations of the crossover operators at different rates, that no single crossover works best. Rather, a combination of two or more of the current operators provides the best solution.

4.7.2 Application to the test system

The results from the genetic algorithm fitting of the test system of chemical reactions are promising, but indicate that there are still problems to tackle. The implementation of Part Two has gone through several incarnations. Initial runs yielded results for the system such as:

$$\begin{aligned}
\frac{dX_1}{dt} &= -0.355X_1X_3 + 0.486X_2X_4 - 0.396X_1X_4 - 0.388X_1X_2 \\
\frac{dX_2}{dt} &= 1.13X_1X_3X_4 - 0.352X_2X_3 - 1.06X_1X_2X_3X_4 - 0.292X_1X_2X_4 \\
\frac{dX_3}{dt} &= 0.964X_1X_2 - 0.915X_2X_3 - 0.293X_1X_2X_3 \\
\frac{dX4}{dt} &= 1.31 \cdot 10^{-5}X_3 - 0.0047X_2X_4 + 0.854X_3 - 1.55X_3X_4
\end{aligned} \tag{4.13}$$

This result produces a low value for the squared difference, but is very far from the model differential equation solution (Equation(4.12)). This early result underlines the need for the alterations to the optimization described in Section 4.6.1. Specifically, the structure of

the differential equations does not correspond to a possible set of chemical reactions. For example, the first term in the equation for protein1: $-0.355X_1X_3$ does not appear in the equation for protein3. Further, this term does not appear in any of the other differential equations. Essentially, protein1 loses mass without any of the other components gaining that mass, which means that mass isn't conserved. This solution cannot correspond to a real physical system.

The development of programs to create a valid initial population and reproduction operators greatly improved results. In a given generation, however, there are some very poor individuals being produced. For example, there are sometimes individuals with parameter sets consisting entirely of zeros! Another observation made was that the fitness value of individuals appeared to stall around the value 0.39 for many generations. This seems to indicate that there is a significant minimum at this value that the algorithm is consistently falling into before jumping to a lower value due to a mutation or successful crossover. Upon further investigation it was revealed that this local minimum corresponded to the empty solution. This means that, for the test system, a constant value for the concentration of all proteins over the time course gives a better fitness than many other randomly chosen solutions. This result is likely related to the discrepancy in time scale: most of the change in protein concentration occurs quickly, while the time points at which the protein concentration is measured are comparatively large.

On a more positive note, consider the differential equations describing the form of the network from one of the more successful genetic algorithm trials (using a range of $[-1, 1]$ for α_{ij} and $g_{ijk} = 0$ or 1):

$$\begin{aligned}
\frac{dX_1}{dt} &= -0.0081X_1X_2 + 0.0081X_3 \\
\frac{dX_2}{dt} &= -0.0081X_1X_2 + 0.0108X_3 \\
\frac{dX_3}{dt} &= 0.0081X_1X_2 - 0.0210X_3 - 0.0108X_3 + 0.0081X_3 \\
\frac{dX_4}{dt} &= 0.0210X_3
\end{aligned} \tag{4.14}$$

The values of the kinetic constants compare fairly well with those of the test case, as shown in Table 4.9. Notice, however, that the constant values are fairly repetitive: for example, in the differential equation for X_1, the same constant appears twice (as separate rate constants, k_1 and k_2). This is a result of the limited pool of values available for the alpha parameters

Parameter	Actual Value	Genetic Algorithm Solution
k_1	0.01	0.0081
k_2	0.02	0.0108
k_3	0.2	0.0108 (to X_1) 0.0081 (to X_2)
k_4	0.03	0.0210

Table 4.9: A comparison of the kinetic constants from a successful run of the genetic algorithm to the actual values for the test case. This run consisted of 1000 iterations of a population of 20.

in the crossover operators.

The reactions included in the above solution network are very close to the actual test case reactions. There are a few key differences, however. First of all, notice that the rate constants for the reaction $X_1 + X_2 \leftarrow X_3$ are not the same in the differential equations for X_1 and X_2. So, although the solution reflects the fact that X_3 is a reactant producing both X_1 and X_2, it does not reflect that it is one reaction that produces both proteins. Notice, however, that the sum of the constants for these two contributions of X_3 are close to the value of k_2.

Another important difference between the actual and the fitted systems are that the reaction $X_3 \leftarrow X_4$ is completely missing. Consider the differential equation for X_3: the maximum number of terms has been reached. In the actual system, there are only four terms in the differential equation, but the error noted in the previous paragraph means that there is an extra term. At this point, it seemed that perhaps if the limit, n, on the number of terms were raised then the missing reaction would be found as well. It is very likely that the loss of this reaction has also affected the rate constants of all the reactions.

This result and the results of the next section motivated a modification of the adapted genetic algorithm to allow the maximum number of terms to be chosen at each trial. This involved changes to most of the programs used to adapt the original algorithm, since the number of parameters in an individual depends on the number of terms allowed in each differential equation. A new crossover operator was also created in order to fully take advantage of the increase in possible terms and to incorporate some of the insight gained from the crossover tests of Section 4.7.1.

The new crossover, `amyXovermax`, distributes the parental terms without replacement. However, instead of filling one child until the maximum number of terms is reached

and then filling the next child with the remaining terms, `amyXovermax` randomly chooses one of the children at each iteration and adds a term to that child. This is an improvement in that it avoids the problem of uneven distribution of terms among the children that occurs with the previous crossover operators created in this study.

The results of the optimizations performed with this added functionality are mixed. Some solutions comparable to (4.14) were obtained, but poor solutions were also obtained. I believe that ultimately this added functionality will improve the results of the optimization, since it provides the system with much more flexibility.

An improvement to this new feature might be to start with a large limit on the maximum number of terms and decrease this limit over the course of the optimization. For instance, one could define a function, $F(gen)$ that sets the maximum number of terms for each variable based on the current generation, gen, where $F(0) = m_0$ and $F(maxgen) = n$. Here, m_0 and $maxgen$ are specified at the start of the optimization, and n is the number of proteins in the network. This setup would permit more mistakes in term distribution early on, and would also mean a larger parental termbank from which to generate offspring. I intend to test this theory in future work.

Application to EGF simulation results of Part One

The third equivalence set found in Part One (Section 4.4) contains the 5 proteins:

Protein Number	Protein Name
1	SOS
2	RGS
3	GS
4	RSHGS
5	SHGS

Despite my reservations about the grouping from Part One, I ran the optimization for this equivalence set. A 100 iteration run with a population size of 20 produced the following

EGF pathway Set 1		EGF pathway Set 2	
Protein Number	Protein Name	Protein Number	Protein Name
1	RP	1	RP
2	SHC	2	PLC
3	RSH	3	RPL
4	RSHP	4	RPLP
5	SHP	5	PLCP
		6	PLCPI

Table 4.10: Proteins belonging to two of the equivalence sets from Part One, listed in Table 4.3.

results:

$$\begin{aligned}
\frac{dX_1}{dt} &= 0.733X_2X_4X_5 + 0.967X_2 - 0.63X_1X_2 - 0.858X_1X_5 \\
\frac{dX_2}{dt} &= -0.733X_2X_4X_5 - 0.967X_2 - 0.63X_1X_2 \\
\frac{dX_3}{dt} &= -0.156X_3 + 0.63X_1X_2 + 0.858X_1X_5 \\
\frac{dX_4}{dt} &= -0.733X_2X_4X_5 \\
\frac{dX_5}{dt} &= 0.156X_3 - 0.733X_2X_4X_5 - 0.858X_1X_5
\end{aligned} \tag{4.15}$$

These equations correspond to the following series of chemical reactions:

$$\begin{aligned}
RGS + RSHGS + GS &\rightarrow SOS \\
RGS &\rightarrow SOS \\
SOS + RGS &\rightarrow GS \\
SHGS + SOS &\rightarrow GS \\
GS &\rightarrow SHGS
\end{aligned} \tag{4.16}$$

These equations match very badly with the actual EGF pathway. The last equation above is the closest match, although the protein SHP is missing from the left hand side. Although the algorithm should run for more iterations, this preliminary result is very discouraging.

In Section 4.4, I discussed the problems with the results of Part One. Since the equivalence sets from matrix R seem much more promising, I performed runs of Part Two using the two equivalence sets listed in Table 4.10. The 7 reactions that should be found

by the optimization of Set 1 are:

$$\begin{aligned} RP + SHC &\rightleftharpoons RSH \\ RSH &\rightleftharpoons RSHP \\ RSHP &\rightleftharpoons SHP + RP \\ SHP &\rightarrow SHC \end{aligned} \tag{4.17}$$

and the 9 reactions for Set 2 are:

$$\begin{aligned} RP + PLC &\rightleftharpoons RPL \\ RPL &\rightleftharpoons RPLP \\ RPLP &\rightleftharpoons RP + PLCP \\ PLCP &\rightarrow PLC \\ PLCP &\rightleftharpoons PLCP - I \end{aligned} \tag{4.18}$$

Again, mixed results were seen. For Set 1, errors between 0.10 and 0.67 were obtained for runs varying between 10 and 20 individuals and between 500 and 2000 iterations. The solutions obtained were quite varied. An example of a solution obtained after 500 iterations using a population of 10 individuals is:

$$\begin{aligned} RP &\rightleftharpoons RSH \\ SHC &\rightarrow RSH \\ RP + RSH &\rightarrow SHP \\ RP + SHC &\rightarrow RSH \end{aligned} \tag{4.19}$$

Some of these reactions match the solutions that we are looking for quite well. For example, the last reaction of (4.19) matches the forward reaction of (4.17) exactly. However, there is quite a bit of repetition in the terms. RSH appears in all 5 reactions, while RSHP appears in none. It appears that, as for the test system, the limit on the maximum number of terms restricts both the variety of terms and the number of reactions.

For Set 2, errors ranged between 0.93 and 0.16 for a similar range of trials. The maximum error of 0.93 was obtained for a run with bounds of $[-10, 10]$ specified for the rate constants, α_{ij}. Subsequently, the range of α_{ij} was restricted to $[-6, 6]$, since no rate

constants used in the simulation for the reactions of Set 2 exceed these bounds. An example of results obtained for Set 2 with this reduced range and 1000 iterations is

$$\begin{aligned} RPL &\rightleftharpoons RP \\ PLC &\rightleftharpoons RP \\ PLC + PLCP &\rightarrow RP \\ RPL &\rightarrow PLCPI \end{aligned} \tag{4.20}$$

As seen with Set 1, there are two few reactions resulting from the optimization. In Set 1, 4-6 reactions were typically found, while the actual number being sought is 7 reactions. In Set 2, 7-8 reactions were usually found, rather than 9. Duplicate reactions were often seen, and two or three proteins dominating all the reactions was common. This seems to be a clear indication that the crossover operators need to be improved so that they do not limit the diversity and the number of terms.

4.8 Discussion

The results presented in this chapter show that the method of Maki *et al* is applicable to signal transduction networks, but only after significant modifications. An important issue for this application of the method is obtaining the necessary data experimentally. There is a large quantity of very specific data required, i.e. time course and steady state data for the system after the "deletion" of individual network components. This will not be possible for all systems. A change in data set would have the most impact on Part One of the method. It would be interesting and useful to study the results from adapting the method to different data sets.

An important consideration for Part One is the determination of the threshold for the binary quantization of the data. Different threshold levels resulted in different equivalence sets. This is definitely a drawback to the method in the absence of a specific procedure for determining the threshold.

There are many variables to experiment with in Part Two. Unfortunately, in its present form, the genetic algorithm optimization is very slow: one 2000 iteration run of Set 2 with a population of 20 exceeded 100 hours on a PII/550 SMP Linux PC. This is an obstacle to performing the number of trials that should be performed to test the

modifications to the algorithm, especially given the fact that it is a stochastic optimization method. So, improving the speed of the genetic algorithm code is a priority. Given that the selection and reproduction must take place, the set of differential equations must be solved, and the objective function evaluated for each individual of the population at each of thousands of iterations, this is not a simple task. It is, however, a necessary one.

Several measures were taken to speed up the optimization. The number of crossovers at each generation was reduced to 60% of the number of individuals in the population, and only one crossover operator was used for most tests. The MATLAB code was compiled to C code using the `mcc` function. This improved the speed significantly (40% improvement was seen in a test case).

One modification that may improve both the efficiency and the layout of the genetic algorithm is the use of multiple chromosomes in the genome of an individual. The use of multiple chromosomes is not common, but examples do exist [43]. In this application, the use of two chromosomes seems like a natural representation of the parameter space, and genetic algorithms generally do best with the most natural representations [32]. For example, one chromosome would contain all the α_{ij} and another would contain all the g_{ijk}. Then each chromosome would only recombine with the "homologous" chromosome of the other parent. This would likely simplify and improve the crossover operators. As well, it would eliminate the linear storage of the parameters in one long array and thus improve the efficiency of all the operations listed above.

Another change that could be made would be to store only one version of a term, rather than a term and its complementary terms, and indicate the differential equations to which that term is assigned. This would decrease the size of the parameter set significantly.

Assuming that the speed of the program can be improved, many more trials should be done. Some things that should be checked are:

- Fitness function: Right now the fitness of an individual is measured using a least squares function. However, the equation

$$f = \sum_{i=1}^{n} \sum_{t} (X_{calc,i,t} - X_{expt,i,t})^2$$

is being used, rather than

$$f = \sum_{i=1}^{n} \sum_{t} \left(\frac{(X_{calc,i,t} - X_{expt,i,t})}{X_{expt,i,t}} \right)^2$$

as in the implementation of Maki *et al.* This second function is a better choice because, in the first equation, proteins with larger concentration values will contribute more to the sum than those with smaller concentrations. However, the first equation is used in this application because many of the initial experimental concentrations are zero or very small. It would be good to test variations of this objective function to find the one most suited to this application.

- Crossover operators: The crossover operators need to be tested much more than they have been. This testing needs to include combinations of crossover operators and new methods of crossover. The crossover operation is an essential part of the algorithm and the outcome is very dependent on its efficient and successful implementation. An important point that has emerged from the results is that the current errors in the optimal solutions are much too high, even over a large number of iterations. This means that the crossovers need to be able to operator more effectively. An option may be to combine a traditional crossover with a deterministic search algorithm for the real numbered rate constants.

- Effects of choosing the number of generations, number of individuals in the population, and frequency of comet-strikes. Preliminary studies were conducted to determine the effect of population size on fitness of the optimal solution and running time of the optimization. Figure 4.7 shows that error depends on the population size in a nonlinear fashion. Notice that for a population size of 25, a significant minimum in the error is seen. The reason for this is not known.

  Figure 4.8 is a plot of run time *versus* the number of individuals in the population for the same runs used for Figure 4.7. The completion time for a run increases in a nearly linear fashion with the number of individuals. This is not intuitively obvious, since the algorithm involves many loops through the parameter set in the course of the reproduction and evaluation operations. There are several large outliers. The outlier at population size 25 corresponds to the low error seen in Figure 4.7, while the outlier

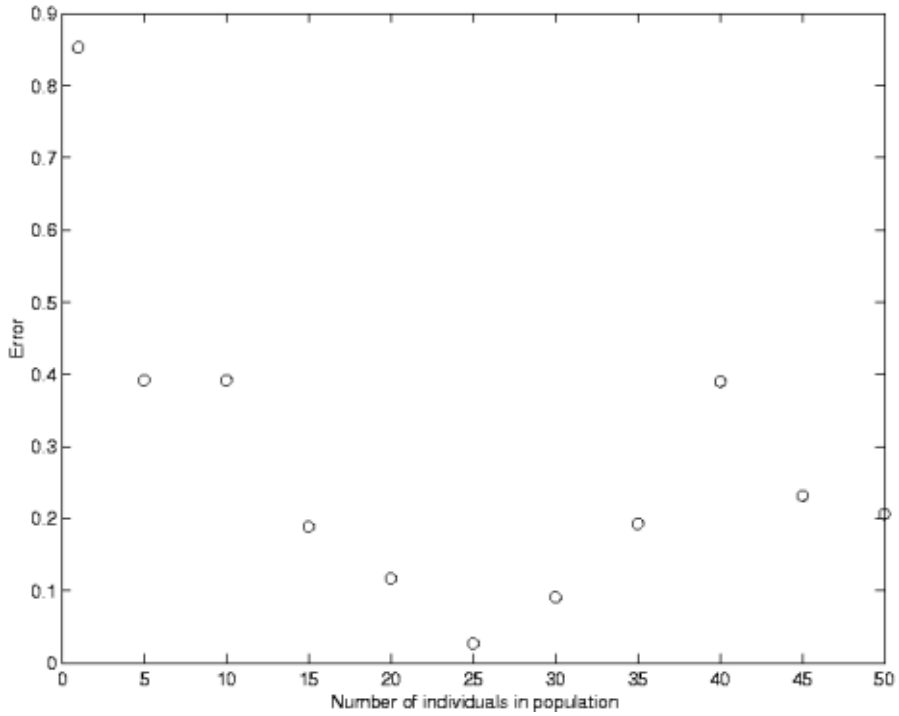

Figure 4.7: A plot showing the relationship between the number of individuals in a population and the fitness value of the optimal solution. The results are for a 50 iteration optimization of the test system (Equation(4.12)).

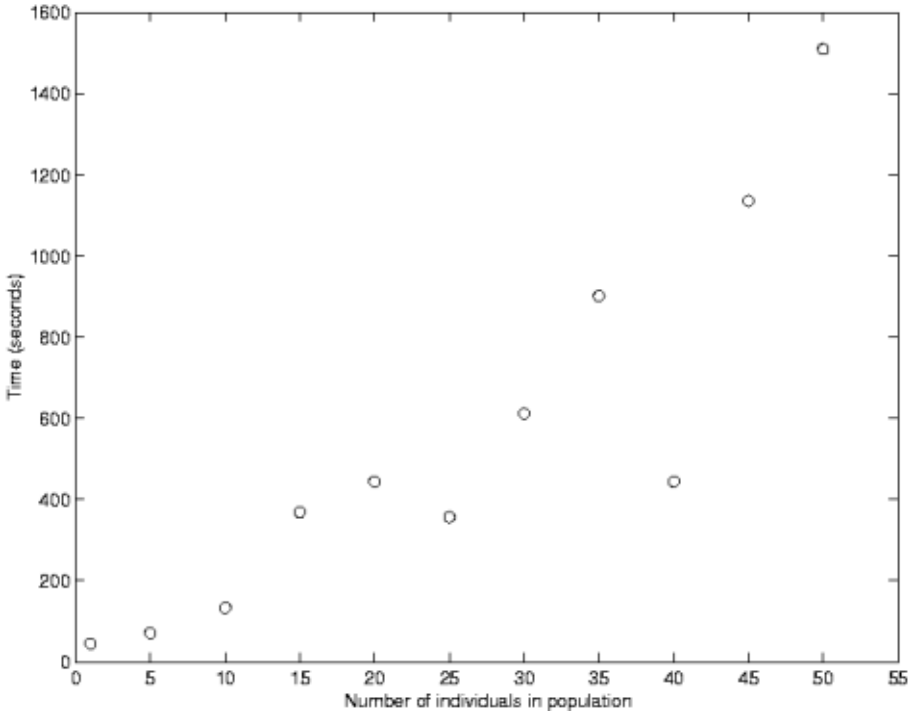

Figure 4.8: A plot showing the relationship between the number of individuals in a population and the run time of the algorithm. The results are for a 50 iteration optimization of the test system (Equation(4.12)).

at population size 40 corresponds to a high error. This latter solution contained only one unique term.

It is important to note that the above figures are for a single run of 50 iterations using the test case. The results are not the same for every run nor for different numbers of iterations. It is necessary to compile results from many more tests to obtain statistical measures for these dependencies. However, the main features of the above figures have been seen in all population test runs performed thus far.

Chapter 5

Concluding Remarks

The preceeding chapters have presented a variety of multivariate and reverse engineering methods for analysing biochemical data. In the absence of concrete sources of kinetic constants, these methods are important tools in the effort to decipher the complex interactions involved in signal transduction.

The difficulty of obtaining data has been mentioned, but not emphasized. One important point is that most of the methods discussed in this study are methods that have been applied to either gene expression data or to simulated data. Beyond the statistical issues (which are very important and which would be the subject of another major investigation) of preparing the data for analysis, there is the issue of whether the needed data can be obtained for signal transduction studies at all.

The results of the work presented in this thesis are not entirely positive. The multivariate techniques discussed in Chapter 2 were able to find natural groupings in the EGF pathway simulation data. This is encouraging in that parallel results have been seen for gene expression data. This suggests that it should be possible to study patterns in large protein data sets in a similar fashion to large-scale gene expression data. Unfortunately, I was unable to identify similarities between proteins within a cluster (beyond similar patterns of concentration change over the time course). Finding this connection among clustered proteins is important for making hypotheses about roles for less well known pathway components. However, even without this knowledge, multivariate analysis could be used to discover proteins that are correlated with patterns in the data using PCA.

Chapter 3 presented three reverse engineering methods that have been applied to gene expression data. Bayesian Networks seem very promising for modeling pathways due

to the flexibility in defining variables, flexibility in defining connections between variables, and probabilistic nature. The main problem with Bayesian networks arises from the the difficulty of scoring networks of the size necessary to study signal transduction pathways. Further, if the approach of Hartemink *et al* is used, then the problem of generating model hypotheses for comparison is a difficult task in itself. Despite these problems, it would be interesting to study Bayesian Networks that include both gene and protein components using both gene expression and protein concentration data to score the models.

In Part One of the method presented in Chapter 4, there are some unresolved issues in the application of the method to a system as complicated as a real signal transduction pathway. As it stands, the method creates equivalence sets that are too large and that do not represent the actual pathway well. This negates the usefulness of the "divide and conquer" strategy and sabotages the results of Part Two.

Part Two has many problems that have been discussed in Section 4.8. The main difficulty with Part Two is that it is very slow to run, which makes it difficult to obtain a statistical measure of its success or failure. In order to improve the efficiency of the optimization, changes to the data structure of the parameter set (to create two chromosomes, for example) may be necessary. Such a change would necessitate changes to all of the programs created to modify the genetic algorithm toolbox: a major overhaul. Another option for improving the speed is to perform the optimization using a more efficient language such as C. [note: I am currently running simulations with a compiled version of the code, which appears to be faster. I will note this in the results section of Chapter 4].

Despite these setbacks, the application of the method of Maki *et al* and the multivariate analysis techniques to the EGF pathway has been a useful exercise in adapting methods intended for the study of genetic networks to the study of protein networks. Specifically, it has become clear that it is important to take into consideration the fact that interactions between proteins are more defined and occur on a smaller time scale than interactions in a genetic network. Overall, I am hopeful that these methods can be applied successfully once these differences are accounted for.

Bibliography

[1] GeneCluster 1.0.

[2] A. Abbot, *Betting on tomorrow's chips*, Nature **415** (2002), 112–114.

[3] B. Alberts et al., *Molecular biology of the cell*, Garland Publishing, 1994.

[4] O. Alter, P. O. Brown, and D. Botstein, *Singular value decomposition for genome-wide expression data processing and modeling*, PNAS **97** (2000), no. 18, 10101–10106.

[5] S. Andreassen et al., *Munin - An expert EMG assistant*, Computer-aided electromyography and expert systems (J. E. Desmedt, ed.), Elsevier Science, 1989, pp. 255–277.

[6] R. Baranowski, personal communication.

[7] A. Chaudhry, Gepasi program modelling early events of the EGF pathway.

[8] S. Chu et al., *The transcriptional program of sporulation in budding yeast*, Science **282** (1998), 699–705.

[9] L. Davis, *Handbook of genetic algorithms*, Van Nostrand Reinhold, 1991.

[10] J. L. DeRisi and V. R. Iyer, *Genomics and array technology*, Curr Opin Oncol **11** (1999), no. 1, 76–9.

[11] P. D'Haeseleer, S. Liang, and R. Somogyi, *Genetic network inference: From co-expression clustering to reverse engineering*, Bioinformatics **16** (2000), no. 8, 707–726.

[12] M. B. Eisen et al., *Cluster analysis and display of genome-wide expression patterns*, PNAS **95** (1998), 14863–14868.

[13] G. S. Fishman, *Monte carlo: Concepts, algorithms, and applications*, Springer, 1996.

[14] N. Friedman et al., *Using Bayesian networks to analyze expression data*, RECOMB 2000 ACM-GIGACT (2000), 0–1.

[15] L. Glass and S. A. Kauffman, *The logical analysis of continuous, nonlinear biochemical control networks*, J Theoretical Biology **39** (l973), l03–l29.

[16] G. Gorry and G. Barnett, *Experience with a model of sequential diagnosis*, Computers and Biomedical Research **1** (1968), 409–507.

[17] A. J. Hartemink, D. K. Gifford, T. S. Jaakola, and R A Young, *Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks*, Pac Symp Biocomput **5** (2001), 422–433.

[18] N. S. Holter et al., *Fundamental patterns underlying gene expression profiles: Simplicity from complexity*, PNAS **97** (2000), no. 15, 8409–8414.

[19] C. Houck, J. Joines, and M. Kay, *A genetic algorithm for function optimization: A matlab implementation*, tech. report, NCSU-IE, 1995.

[20] D. H. Irvine and M. A. Savageau, *Efficient solution of nonlinear ordinary differential equations expressed in S-system canonical form*, SIAM J Numerical Analysis **27** (1990), 704–735.

[21] V. R. Iyer et al., *The transcriptional program in the response of human fibroblasts to serum*, Science **283** (1999), 83–87.

[22] T. Jaakola and M. Jordan, *Variatonal probabilistic inference and the qmr-dt network*, J of Artif Intell Res **10** (1999), 291–322.

[23] F. Jacob et al., *L'operon: groupe de genes a l'expression coordonne par un operateur*, Comptes rendus de l'Acadmie des Sciences **245** (1960), 1727–1729.

[24] F. V. Jensen, *An introduction to Bayesian networks*, Spring-Verlag, 1996.

[25] R. A. Johnson, *Applied multivariate statistical analysis*, second ed., Prentice-Hall, 1988.

[26] I. T. Jolliffe, *Principal component analysis*, Springer, 1986.

[27] G. Karp, *Cell and molecular biology: Concepts and experiments*, John Wiley and Sons, Inc., 1996.

[28] C. L. Karr and L. M. Freeman, *Industrial applications of genetic algorithms*, CRC Press, 1999.

[29] B. N. Kholodenko et al., *Quantification of short term signaling by the epidermal growth factor receptor*, J Biol Chem **274** (1999), no. 42, 30169–30181.

[30] T. Kohonen, *What generalizations of the self-organizing map make sense?*, Proc. of ICANN, Springer-Verlag, 1994, pp. 292–297.

[31] ———, *Self-organizing maps*, third ed., Springer, 2001.

[32] J. R. Koza, *Genetic programming : on the programming of computers by means of natural selection*, MIT Press, 1992.

[33] M. H. Lee and H. Y. Yang, *Contributions in the domain of cancer research: Review negative regulators of cyclin-dependent kinases and their roles in cancers*, Cell Mol Life Sci **58** (2001), no. 12–13, 1907–1922.

[34] S. Liang, S. Fuhrman, and R. Somogyi, *Reveal, a general reverse engineering algorithm for inference of genetic network architectures*, Pac Symp Biocomput **3** (1999), 18–29.

[35] Y. Maki, D. Tominaga, et al., *Development of a system for the inference of large scale genetic networks*, Pac Symp Biocomput **6** (2001), 446–458.

[36] P. Mendes, *Gepasi: A software package for modelling the dynamics, steady states and control of biochemical and other systems*, Comput. Applic. Biosci. **9** (1993), 563–571.

[37] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer-Verlag, 1994.

[38] E. Mjolsness, D. H. Sharp, and J. Reinitz, *A connectionist model of development*, J Theor Biol **152** (1991), no. 4, 429–454.

[39] K. Murphy, *A brief introduction to graphical models and Bayesian networks*, http://www.cs.berkeley.edu/ murphyk/Bayes/bayes.html.

[40] B. O. Palsson and E. N. Lightfoot, *Mathematical modelling of dynamics and control in metabolic networks. part i. local stability analysis of single biochemical control loops*, J Theor Biol **111** (1984), 273–302.

[41] T. Pawson, G. D. Gish, and P. Nash, *SH2 domains, interaction modules and cellular wiring*, Trends Cell Biol **11** (2001), no. 12, 504–511.

[42] J. Pearl, *Fusion, propagation, and structuring in belief networks*, Artificial Intelligence **29** (1986), no. 3, 241–288.

[43] H. J. Pierrot and R. Hinterding, *Using multi-chromosomes to solve a simple mixed integer problem*, Australian Joint Conference on Artificial Intelligence, 1997, pp. 137–146.

[44] W. H. Press et al., *Numerical recipes in C: the art of scientific computing*, second ed., Cambridge University Press, 1988.

[45] J. Reinitz and D. H. Sharp, *Mechanism of eve stripe formation*, Mech Dev **49** (1995), 133–158.

[46] M. A. Savageau and E. O. Voit, *Recasting nonlinear differential equations as s-systems: A canonical nonlinear form*, Math Biosci **87** (1987), 83–115.

[47] C. E. Shannon and W. Weaver, *The mathematical theory of communication*, University of Illinois Press, 1949.

[48] M. A. Shea and G. K. Ackers, *The or control system of bacteriophage lambda. a physical-chemical model for gene regulation*, J Mol Biol **181** (1985), no. 2, 211–230.

[49] P. T. Spellman et al., *Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization*, Mol Biol Cell **9** (1998), 3273–3297.

[50] G. S. Stein et al., *The molecular basis of cell cycle and growth control*, Wiley-Liss, Inc., 1999.

[51] P. Tamayo et al., *Interpreting patterns of gene expression using self-organizing maps: Methods and application to hematopoietic differentiation*, PNAS **96** (1999), 2907–2912.

[52] D. Tominaga et al., *Nonlinear numerical optimization technique based on a genetic algorithm for inverse problems: Towards the inference of genetic networks*, Computer Science and Biology (Proc of the German Conf on Bioinformatics), 1999, pp. 127–140.

[53] V. E. Velculescu et al., *Serial analysis of gene expression*, Science **270** (1995), 484–487.

[54] P. Vincens and P. Tarroux, *Two-dimensional electrophoresis computerized processing*, Int J Biochem **20** (1988), no. 5, 499–509.

[55] D. C. Weaver, C. T. Workman, and G. D. Stormo, *Modeling regulatory networks with weight matrices*, Pac Symp Biocomput **3** (1999), 30169–30181.

[56] A. Zien et al., *Analysis of gene expression data with pathway scores*, Proc Int Conf Intell Syst Mol Biol, vol. 8, 2000, pp. 407–17.

Appendix A

Finding the maximum of a quadratic form under a quadratic constraint

We must find the maximum of a matrix $\mathbf{a_i^t S a_i}$ under the constraint $\mathbf{a_i^t a_i = 1}$. In this section we look at a more general case of this problem, that of finding a vector $\mathbf{u}$ which maximizes $\mathbf{u^t S u}$ such that $\mathbf{u^t M u} = 1$, where $\mathbf{S}$ and $\mathbf{M}$ are symmetric matrices and $\mathbf{M}$ is positive definite.

We find the maximum by setting the derivative of the Lagrangian equation to zero. For this problem the equation is

$$\mathcal{L} = \mathbf{u^t S u^t} - \lambda(\mathbf{u^t M u - 1}) \tag{A.1}$$

where λ is a Lagrange multiplier. The derivative of $\mathbf{u^t S u}$ is found using the form

$$\mathbf{u^t S u} = \sum_i \sum_j s_{ij} u_i u_j \tag{A.2}$$

where the derivative of each part is found successively. Thus, in matrix form, we find that

$$\frac{\partial(\mathbf{u^t S u})}{\partial \mathbf{u}} = 2\mathbf{S u} \tag{A.3}$$

We can now compute the derivative of the Lagrangian equation, which when set equal to

zero gives us the necessary condition for a maximum or minimum. The extrema is found by solving for lambda, as follows

$$\mathcal{L} = \mathbf{u^t S u^t} - \lambda(\mathbf{u^t M u} - \mathbf{1}) \tag{A.4}$$

$$\Rightarrow \quad \frac{\partial \mathcal{L}}{\partial \mathbf{u}} = 2\mathbf{Su} - 2\lambda\mathbf{Mu} = 0$$

$$\Rightarrow \quad \mathbf{Su} = \lambda\mathbf{Mu}$$

$$\Rightarrow \quad \mathbf{u^t Su} = \lambda\mathbf{u^t Mu}$$

$$\Rightarrow \quad \lambda = \mathbf{u^t Su} \tag{A.5}$$

The final line being a result of applying the constraint $\mathbf{u^t Mu} = 1$. Thus, the maximum is given by $\lambda = \mathbf{u^t Su}$.

When $\mathbf{M}$ is a positive definite matrix and therefore nonsingular we can write

$$\mathbf{M^{-1}Su} = \lambda\mathbf{u} \tag{A.6}$$

which implies that the vector $\mathbf{u}$ which maximizes $\mathbf{u^t Su}$ such that $\mathbf{u^t Mu} = 1$ is given by the eigen vector of $\mathbf{M^{-1}S}$ that corresponds to the largest eigenvalue λ.

Let's call this vector $\mathbf{u_1}$. Now, we wish to find $\mathbf{u_2}$ which maximizes $\mathbf{u_2^t S u_2}$ such that the conditions $\mathbf{u_1^t M u_2} = 0$ and $\mathbf{u_2^t M u_2} = 1$ hold. Using the same approach as above, the Lagrange equation now reads

$$\mathcal{L} = \mathbf{u_2^t S u_2} - \lambda_2(\mathbf{u_2^t M u_2} - 1) - \mu_2\mathbf{u_2^t M u_1} \tag{A.7}$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{u_2}} = 2\mathbf{Su_2} - 2\lambda_2\mathbf{Mu_2} - \mu_2\mathbf{Mu_1} = 0 \tag{A.8}$$

Multiplying in front by $\mathbf{u_1^t}$ we find that μ_2 must vanish since the contraints dictate that the other two terms in (A.8) are zero. Therefore we have that

$$\mathbf{Su_2} = \lambda_2\mathbf{Mu_2} \tag{A.9}$$

If we continue this process, we find that the vector $\mathbf{u}_\alpha$ satisfying $\mathbf{u_\alpha^t M u_\alpha} = 1$ that is M-orthogonal to all vectors $\mathbf{u}_\beta$ that are already determined (that is $u_\alpha^t M u_\beta = 0$ for $\beta < \alpha$),

and maximizes $\mathbf{u}_\alpha^\mathbf{t}\mathbf{S}\mathbf{u}_\alpha$ satisfies

$$\mathbf{S}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{M}\mathbf{u}_\alpha \tag{A.10}$$

So, for $\mathbf{M}$ nonsingular

$$\mathbf{M}^{-\mathbf{1}}\mathbf{S}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{u}_\alpha \tag{A.11}$$

where α is not greater than the order of matrix $\mathbf{S}$.

Appendix B

Modifications to the genetic algorithm toolbox

This appendix contains the main MATLAB programs that I wrote to carry out Part One of the method of Maki *et al* and to implement the major changes to GAOT (the Genetic Algorithm Optimization Toolbox). Also included is the MATLAB function that evaluates the objective function for the optimization, `testegfeval`.

```
**************************************************************************
function [binM]=binaryR(inM,threshold)

%
%[binM]=binaryR(inM,threshold)
%
%  binM - output binary matrix
%  inM - input matrix
%  threshold - values above threshold or below 1/threshold are set to 1,
%              all other entries are 0
%
**************************************************************************

[numrows,numcols]=size(inM);

for col=1:numcols
  for row=1:numrows
    if (inM(row,col) > threshold) | (inM(row,col) < 1/threshold)
      if (inM(row,col) > 0)
        binM(row,col)=1;
      end
```

```
    else
      binM(row,col)=0;
    end
  end
end
```

```
************************************************************************
function [Rstar]=UnionR(R)

%
%[Rstar]=UnionR(R)
%
%  Rstar - output acessibility matrix
%  R - input binary matrix
%  side note: Rn - matrix of all the Rn (size msize x msize(msize+1)
%
************************************************************************

[numrows,numcols]=size(R);

if numcols ~= numrows
  error('Need a square matrix')
end

msize=numrows;

% first block of msize x msize is R0, ie ones on the diagonal
for k=1:msize
  Rn(k,k)=1;
end

% build the matrix Rn by tacking on R1,...,Rn+1
% a matrix of size msize x msize(msize+1) is created

for n=0:msize

for i=1:msize
  for j=1:msize
    for k=1:msize
      checkit(k)=Rn(i,k+n*msize)*R(k,j);
    end
    Rn(i,j+(n+1)*msize)=min([1 sum(checkit)]);
  end
end

end

% take the union of all the matrices stored in Rn, coordinate by coordinate
for j=1:msize
  for i=1:msize
    for n=0:msize+1
      if Rn(i,j+msize*n) == 1
    Rstar(i,j)=1;
      end
```

```
    end
  end
end
```

```
************************************************************************
function [ER]=eqrel(Rstar)

%
% [ER]=eqrel(Rstar)
%
%  ER = matrix detailing the equivalence relation between each gene
%       ER(a,b)= 1 if Rstar(a,b)=1 $ Rstar(b.a)=1
%                0 otherwise
%  Rstar = input binary accessibility matrix
%
************************************************************************

[numrows,numcols]=size(Rstar);

if numcols ~= numrows
  error('Need a square matrix')
end

msize = numcols;

for i=1:msize
  for j=i:msize
    if (Rstar(i,j)==1) & (Rstar(j,i)==1)
      ER(i,j)=1;
      ER(j,i)=1;
    end
  end
end
```

```
**********************************************************************
function [ES,Rii]=getEQset(ER,Rstar)

%
% [ES,Rii]=eqset(ER,Rstar)
%
%  ES = output vector giving number of all prots in the EQ set
%  Rii = output of nonredundant version of Rstar
%  ER = matrix detailing the equivalence relation between each gene
%        ER(a,b)= 1 if Rstar(a,b)=1 $ Rstar(b.a)=1
%                 0 otherwise
%  Rstar = input binary accessibility matrix
%
**********************************************************************

[numrows,numcols]=size(Rstar);

if numcols ~= numrows
  error('Need a square matrix')
end

msize = numcols;

% create a labeled Rstar matrix Ri so that I will be able
% to identify which genes have been combined in equivalence sets.

new1=[(1:msize)' Rstar];
Ri=[0:msize
      new1]


% create a matrix ES where each row lists the members of
% an equivalence set

for i=1:msize
 numeq=1;
 for j=1:msize
  if ER(i,j)==1
   ES(i,numeq)=j;
   numeq=numeq+1;
  end
 end
end

% Eliminate duplicate rows
for i = msize:-1:1
 for k = i-1:-1:1
  if ES(i,:)==ES(k,:)
```

```
   ES(i,:) = [];
   break
  end
 end
end

% copy only one row from each equivalence set to Rii

eqsetcount = size(ES,1);
maxmem = size(ES,2);
for i=1:eqsetcount
 Rii(i,:)=Ri(ES(i,1)+1,:);
end

Rii = [0:msize
       Rii];

% set the columns corresponding to numbers identified as other
% members of the equivalence set to a column of ones to mark
% for deletion

for i=1:eqsetcount
%count the number of members in the equivalence set
 eqmem(i) = 0;
 for j=1:maxmem
  if ES(i,j)~=0
    eqmem(i) = eqmem(i)+1;
  end
 end
 if eqmem(i)>1
  for k = 2:eqmem(i)
   for row = 2:size(Rii,1)
    Rii(row,ES(i,k)+1) = 2;
   end
  end
 end
end

% get rid of any columns in Rii that are all 2

for j=msize+1:-1:2
 if Rii(2,j)==2
  Rii(:,j)=[];
 end
end

% set the diagonals to 0
```

```
for k = 1:size(Rii,1)
 Rii(k,k) = 0;
end
```

```
************************************************************************
function [S,ord]=topsort(Rii)

%
% [S,ord]=topsort(Rii)
%
%  S = topo sorted matrix
%  ord = topological order of columns
%  Rii = input labeled binary matrix
%
************************************************************************

[numrows,numcols]=size(Rii);

if numcols ~= numrows
  error('Need a square matrix')
end

% get rid of labels on the matrix
R = Rii;
R(1,:)=[];
R(:,1)=[];

msize = size(R,1);

% create a vector called indeg which gives the number of
% ones in each column (the indegree for the column's node)

for col=1:msize
 onecount = 0;
 for row = 1:msize
  if R(row,col) == 1
   onecount = onecount + 1;
  end
 end
 indeg(col) = onecount;
end

% create an array holding the indexes of prots with no dependents
% ie) nodes with zero indegree)
count = 0;
nodep = [];

for col = 1:msize
 if indeg(col) == 0
  nodep = [nodep col];
 end
end
```

```
% pick prots with no dependents off the top of the queue and assign
% it the next order and decrement the indeg of the prots affected by
% that prot
while ~isempty(nodep)
 v = nodep(1);
 if length(nodep) == 1
  nodep = [];
 else
  nodep = nodep(2:length(nodep));
 end
 count = count + 1;
 ord(v) = count;
 for col = 1:msize
  if R(v,col) == 1
   indeg(col) = indeg(col) - 1;
   if indeg(col) == 0
    nodep = [nodep col];
   end
  end
 end
end

% If the sort worked with the given graph, rearrange the matrix
% in topological sort order
if count < msize
 error ('Graph is cyclic')
else
 [Y,I] = sort(ord);

 for i = 1:msize
  for j = 1:msize
   S(i,j) = R(I(i),I(j));
  end
 end
end
```

```
%************************************************************************
function [S]=skel2(R)

%
% function [S]=skeleton(R)
%
%  S = output skeleton matrix
%  R = input unlabeled binary matrix
%
%************************************************************************
[numrows,numcols]=size(R);
if numcols ~= numrows
  error('Need a square matrix')
end
msize = size(R,1);

% created list of effect pairs

list = [];
for col = 1:msize
 for row = 1:msize
  if R(row,col) == 1
   list= [list
          row col];
  end
 end
end

listsize= size(list,1);

% remove pairs from the list that represent indirect effects

for i = 1:listsize
 for k = 1:listsize
  if list(k,1) == list(i,2)
   y = list(k,2);
   for k2 = 1:listsize
    if (list(k2,1) == list(i,1)) & (list(k2,2) == y)
     R(list(k2,1),list(k2,2)) = 0;
    end
   end
  end
 end
end

S = R;
```

```
%************************************************************************
function [x,endpop]=testegf(numpop,abound,gbound,nvars,maxgen,best)
%
% [x,endpop]=testegf(numpop,abound,gbound,nvars,maxgen,best)
%
% function called from the MATLAB command line to perform the genetic
% algorithm run for determining the best fit parameter solution for
% the network of protein chamical reactions
%
% x - optimal solution
% endpop - the parameter sets of all individuals in the final generation
% numpop - number of individuals in the population
% abound - bounds on the alpha parameters
% gbound - bounds on the g parameters
% nvars  - number of proteins in the network
% maxgen - maximum number of generations (termination criteria)
% best   - previous solution to include in the new initial population
%
%************************************************************************

npars=nvars^2*(nvars+1);

%gijk bounds
bounds(1:npars,1)=gbound(1)*ones(npars,1);
bounds(1:npars,2)=gbound(2)*ones(npars,1);

%alpha bounds
bounds(1:nvars^2,1) = abound(1)*ones(nvars^2,1);
bounds(1:nvars^2,2) = abound(2)*ones(nvars^2,1);

%set the evaluation function determining the fitness function
evalFN = 'testegfeval';
evalopts=[];

%create the starting population
if size(best,1)>1
 startpop=best
else
 startpop = initpop(numpop,evalFN,abound,gbound,nvars,best)
end
opts=[1e-6 1 0];

%set the termination function and criteria
termFN=['optMaxGenTerm'];
termOps=[maxgen -1.6875e-5 1e-4];

%set the selection function and options
selectFN=['normGeomSelect'];
```

```
selectParams=[0.1];

%set the crossover function and options
nxOver=[round(npars*0.3) round(npars*0.8)];
xOverFNs=['amyXover1 amyXover2'];
xOverParams=[nxOver(1);nxOver(2)];

%set the mutation function and criteria
nmuts=4;
mutFNs=['amyMutation'];
mutParams=[nmuts];

%make the call to the genetic algorithm
[x,endpop,bpop,traceinfo]=...
ga(bounds,evalFN,evalopts,startpop,opts,termFN,termOps,selectFN,selectParams,
xOverFNs,xOverParams,mutFNs,mutParams);
```

```
*********************************************************************
function pop=initpop(num,evalFN,abound,gbound,n,best)
% pop=initpop(num,evalFN,abound,gbound,n)
%
% Creates a new, valid population for the genetic algorithm optimization
% of series of chemical reactions between proteins
%
%  num    - number of individuals in the population
%  evalFN - evaluation function determining the fitness of an individual
%  abound - bounds on the alpha parameters in form [low,high]
%  gbound - bounds on the g parameters in form [low,high]
%  n      - number of components in the network (proteins)
%  best   - option to include a former solution in the new population
%
*********************************************************************

% Determine the number of values that g_ijk can take. This determines
% the number of terms in the termbank.
p=gbound(2)+1;
tbank=termbank(n,p);
maxpicks=p^n;

for i=1:num                  %for each individual

 indiv=zeros(n,n*(n+1)+1);   %initialize the individual's terms

 while all(indiv(:,end)<n)   %while all individuals have less than n terms

%randomly add a valid new term

  curvar=ceil(rand*n);              %randomly choose a variable
  lastpos=indiv(curvar,end)*(n+1);  %get pos of first free spot
  indiv(curvar,lastpos+1)=rand*abound(2);  %randomly choose a pos alpha
  tbank=sortrows(tbank,curvar);     %sort tbank to get zero exp for curvar
  for t=1:maxpicks
   pick=tbank(ceil(rand*p^(n-1)),:);
   if any(pick~=0)               %make sure we don't get any empty terms
    indiv(curvar,lastpos+2:lastpos+n+1)=pick;
    break
   end
  end
%put the complementary term in too
  compvar=find(indiv(curvar,lastpos+2:lastpos+n+1)>0);
  complastpos=indiv(compvar,end)*(n+1);
  for k=1:length(compvar)
   indiv(compvar(k),complastpos(k)+1:complastpos(k)+n+1)=[...
       -indiv(curvar,lastpos+1) indiv(curvar,lastpos+2:lastpos+n+1)];
  end
```

```
  indiv(curvar,end)=indiv(curvar,end)+1;
  indiv(compvar,end)=indiv(compvar,end)+1;

 end

%collect these terms to make individual i

 pop(i,:)=zeros(1,n^2+n^3+1);
 for k=1:n
  lastpos=indiv(k,end);
  pop(i,n*(k-1)+1:n*(k-1)+lastpos)=indiv(k,1:(n+1):lastpos*n);
  for j=0:lastpos-1
   pop(i,n^2+n^2*(k-1)+1+j*n:n^2+n^2*(k-1)+n+j*n)=...
             indiv(k,j*(n+1)+2:(j+1)*(n+1));
  end
 end

%if putting in a previous best one
 if ~isempty(best)
  pop(1,:)=best;
 end

%now evaluate each individual
 estr=['[pop(i,:),pop(i,end)]=',evalFN '(pop(i,:),[]);'];
 eval(estr)
end
```

```
************************************************************************
function [c1,c2]=amyXover1(p1,p2,bounds,Opts)
%
%[c1,c2]=amyXover1(p1,p2,bounds,n)
%
% A crossover operator producing valid children solutions.
% Currently both children can choose from the full bank of
% terms from both parents, though the selection within a child is
% without replacement.
%
% p1     - one parent parameter set selected for reproduction
% p2     - the second parent parameter set
% bounds - vector giving bounds of each parameter
% Opts   - for future options (currently empty)
%
************************************************************************

%determine the number of variables
n=cubit(size(p1,2)-1);
% Determine the number of values that g_ijk can take. This determines
% the number of terms in the termbank.
gbound=bounds(n^2+1,:);
p=gbound(2)+1;
maxpicks=p^n;
numparents=2;

%create a bank of terms from the terms of the parents
origbank=[getsolterms(p1,n);getsolterms(p2,n)];
origbank(find(origbank(:,1)==0),:)=[];
origabank=origbank(:,1);
origbank(:,1)=[];

for i=1:numparents      % for each child
 abank=origabank;       % alphas and terms are pulled from the full bank
 tbank=origbank;
 indiv=zeros(n,n*(n+1)+1);

%add terms while all proteins have less than n terms and there are
%still terms left in the bank
 while all(indiv(:,end)<n) & ~isempty(abank)
  bad=[];
%randomly add a valid new term
  while isempty(bad)
   curvar=ceil(rand*n);              %randomly choose a variable
   lastpos=indiv(curvar,end)*(n+1);  %get pos of first free spot
   picka=ceil(rand*length(abank));   %randomly choose a positive alpha
   indiv(curvar,lastpos+1)=abank(picka);
   tbank=sortrows(tbank,curvar);     %sort tbank so zero exp for curvar
```

```
   bad=find(tbank(:,curvar)==0);
  end
  for t=1:maxpicks
   pickind=ceil(rand*length(bad));
   pick=tbank(pickind,:);
   if any(pick~=0)               %make sure we don't get any empty terms
    indiv(curvar,lastpos+2:lastpos+n+1)=pick;
    abank(picka)=[];
    tbank(pickind,:)=[];
    break
   end
  end
%put the complementary term in too
  compvar=find(indiv(curvar,lastpos+2:lastpos+n+1)>0);
  complastpos=indiv(compvar,end)*(n+1);
  for k=1:length(compvar)
   indiv(compvar(k),complastpos(k)+1:complastpos(k)+n+1)=[...
       -indiv(curvar,lastpos+1) indiv(curvar,lastpos+2:lastpos+n+1)];
  end
  indiv(curvar,end)=indiv(curvar,end)+1;
  indiv(compvar,end)=indiv(compvar,end)+1;

 end

%collect these terms to make individual i

 pop(i,:)=zeros(1,n^2+n^3+1);
 for k=1:n
  lastpos=indiv(k,end);
  pop(i,n*(k-1)+1:n*(k-1)+lastpos)=indiv(k,1:(n+1):lastpos*n);
  for j=0:lastpos-1
   pop(i,n^2*k+j*n+(1:n))=indiv(k,j*(n+1)+2:(j+1)*(n+1));
  end
 end
end

c1=pop(1,:);
c2=pop(2,:);
```

```
************************************************************************
function [c1,c2]=amyXover2(p1,p2,bounds,Opts)
%
%[c1,c2]=amyXover2(p1,p2,bounds,Opts)
%
% A crossover operator producing valid children solutions.
% amyXover1 distributed alphas and terms separately, while this op
% distributes the alphas with the terms that they are with in the
% parents.
%
% p1     - one parent parameter set selected for reproduction
% p2     - the second parent parameter set
% bounds - vector giving bounds of each parameter
% Opts   - for future options (currently empty)
%
************************************************************************

%determine the number of variables
n=cubit(size(p1,2)-1);
% Determine the number of values that g_ijk can take. This determines
% the number of terms in the termbank.
gbound=bounds(n^2+1,:);
p=gbound(2)+1;
numparents=2;

%create the bank of terms
origbank=[getsolterms(p1,n);getsolterms(p2,n)];
origbank(find(origbank(:,1)==0),:)=[];

for i=1:numparents
 tbank=origbank;
 indiv=zeros(n,n*(n+1)+1);

 while all(indiv(:,end)<n) & ~isempty(tbank)
  bad=[];
%randomly add a valid new term
  while isempty(bad)
   curvar=ceil(rand*n);              %randomly choose a variable
   lastpos=indiv(curvar,end)*(n+1);  %get pos of first free spot
   tbank=sortrows(tbank,curvar+1);     %sort tbank so zero exp for curvar
   bad=find(tbank(:,curvar+1)==0);
  end
  pickind=ceil(rand*length(bad));
  pick=tbank(pickind,:);
  indiv(curvar,lastpos+1:lastpos+n+1)=pick;
  tbank(pickind,:)=[];
%put the complementary term in too
  compvar=find(indiv(curvar,lastpos+2:lastpos+n+1)>0); %positive exponents
```

```
  complastpos=indiv(compvar,end)*(n+1);
  for k=1:length(compvar)
   indiv(compvar(k),complastpos(k)+1:complastpos(k)+n+1)=[...
       -indiv(curvar,lastpos+1) indiv(curvar,lastpos+2:lastpos+n+1)];
  end
  indiv(curvar,end)=indiv(curvar,end)+1;
  indiv(compvar,end)=indiv(compvar,end)+1;

 end

%collect these terms to make individual i

 pop(i,:)=zeros(1,n^2+n^3+1);
 for k=1:n
  lastpos=indiv(k,end);
  pop(i,n*(k-1)+1:n*(k-1)+lastpos)=indiv(k,1:(n+1):lastpos*n);
  for j=0:lastpos-1
   pop(i,n^2*k+j*n+(1:n))=indiv(k,j*(n+1)+2:(j+1)*(n+1));
  end
 end
end

c1=pop(1,:);
c2=pop(2,:);
```

```
*********************************************************************
function msol=termmut(sol,n,pos,newval)
% msol=termmut(msol,n,pos,newval)
%
% termmut implements the mutation cases listed in Table 4.6
%
% msol   - the mutated solution
% sol    - solution (parameter set) selected for mutation
% n      - number of proteins in the network
% pos    - position in parameter set selected for mutation
% newval - new value to replace the current parameter in sol
%
*********************************************************************

sol=sol(1:end-1);

if pos<=n^2 % an alpha is changed
 if (sign(sol(pos))==sign(newval)) & (newval ~=0)
  sol=updatemuta(sol,newval,n,pos);
 elseif (sol(pos)~=0 & newval ==0)
  sol=deleteterms(sol,n,pos);
 elseif sol(pos)==0 & newval>0   %not allowed to change if newval<0
  sol=newterm(sol,newval,n,pos);
 end

else % a gijk is changed
 newval=round(newval);  % gijk are integers
 %find i,j,k
 gpos=pos-n^2;
 i=ceil(gpos./n^2);
 j=modmod(ceil(gpos./n),n);
 k=modmod(gpos,n);
 a=(i-1)*n+j;
 if sol((i-1)*n+j)~=0 % only change if alpha > 0
  if (sol(pos)==0) & (newval~=0) % must add a term-put in correct a
   sol=addx(sol,newval,n,[a,i,j,k]);
  elseif (sol(pos)>0) & (newval>0)
   sol=updatemutg(sol,newval,n,[a,i,j,k]);
  elseif (sol(pos)~=0) & (newval==0)
   sol=losex(sol,newval,n,[a,i,j,k]);
  end
 end
end

msol(1:end-1)=sol;
```

```
**********************************************************************
function [sol, val] = testegfeval(sol,options)

%function [sol, val] = testegfeval(sol,options)
%
% Evaluation function for the genetic algorithm stuff for EGF
%    ** Note: this version uses only integers for the gijk params **
% sol - the current parameter set (individual) being evaluated + value
% val - the least squares score for the given parameter set (sol),
% ie) the fitness for the individual under consideration.
%
**********************************************************************

% set some constants and things specific to this run
const=0.001;
datafile = 'gatestt.txt';
TSPAN= [0 15 30 45 60 75 90 105 120];

LP = length(sol)-1;  % sol contains the parameter set plus the value
nvars = cubit(LP);
ntime = length(TSPAN);

% Read from the file containing the experimental data
fid = fopen(datafile);
if fid == 1
 error('uh-oh, not opening the file')
end

% data is read in from file with time pts along cols but
% read in format has time pts along rows
xexp=fscanf(fid,'%g',[ntime,nvars]);
fclose(fid);

%For when an initial pop comes through - set everything to integers
% note - should probably just do this in initpop...
for k=(nvars^2+1):LP
 if sol(k)<0.5
  sol(k)=0;
 elseif sol(k)<1.5
  sol(k)=1;
 elseif sol(k)<2.5
  sol(k)=2;
 else
  sol(k)=3;
 end
end

% get the initial value for solving the system of differential equations
```

```
 init = log(xexp(1,:)+const);
 options=[];

% testlogde is the file that contains the differential equation - it
% is evaluated for the current parameter set by ode23
 [timepts,ycalc]=ode23('testlogde',TSPAN,init,options,sol(1:LP));
 xcalc=exp(ycalc)-const;

% Calculate the fitness by finding the diff bet calc and expt.
% on the condition that none of the constraints are violated
 val = 0;
 for i = 1:nvars
  for t = 1:ntime
   val = val+(xcalc(t,i) - xexp(t,i))^2;
  end
 end
end

% If the ode solver was not able to successfully evaluate the
% system of differential equations, then set a value of 1000
% to punish the solution
if isnan(val) | isinf(val)
 val=1000;
end

%We're doing a minimization so...
 val=-val

**********************************************************************
```