CONTROLLING DISCRETE GENETIC REGULATORY NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OSMAN ABUL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JANUARY 2005

Approval of the Graduate School of Natural and Applied Sciences.

Prof.Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof.Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Prof.Dr. Reda Alhajj	Prof.Dr. Faruk Polat
Co-Supervisor	Supervisor

Examining Committee Members

Prof.Dr. Halil Altay Güvenir	(Bilkent Univ.)	
Prof.Dr. Faruk Polat	(METU)	
Prof.Dr. Kemal Leblebicioğlu	(METU)	
Assoc.Prof.Dr. Göktürk Üçoluk	(METU)	
Assoc.Prof.Dr. İsmail Hakkı Toroslu	(METU)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Osman Abul

Signature :

ABSTRACT

CONTROLLING DISCRETE GENETIC REGULATORY NETWORKS

Abul, Osman
PhD., Department of Computer Engineering
Supervisor: Prof.Dr. Faruk Polat
Co-supervisor: Prof.Dr. Reda Alhajj

January 2005, 141 pages

Genetic regulatory networks can model dynamics of cells. They also allow for studying the effect of internal or external interventions. Selectively applying interventions towards a certain objective is known as controlling network dynamics. In this thesis work, the issue of how the external interventions affect the network is studied. The effects are determined using differential gene expression analysis. The differential gene expression problem is further studied to improve the power of the given method. Control problem for dynamic discrete regulatory networks is formulated. This also addresses the needs for various control strategies, e.g., finite horizon, infinite horizon, and various accounting of state and intervention costs. Control schemes for small to large networks are proposed and experimented. A case study is provided to show how the proposals are exploited; also given is the need for and effectiveness of various control schemes.

Keywords: microarray, data mining, learning, differential gene expression, genetic regulatory network modeling, genetic regulatory network control, saccharomyces cerevisiae.

ÖZ

KESİKLİ GENETİK DÜZENLEYİCİ AĞLARIN KONTROLÜ

Abul, Osman
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof.Dr. Faruk Polat
Ortak Tez Yöneticisi: Prof.Dr. Reda Alhajj

Ocak 2005, 141 sayfa

Genetik düzenleyici ağlar hücre dinamiklerini modelleyebilmektedirler. Bu ağlar aynı zamanda dahili ve harici müdahalelerin etkilerini araştırmaya izin vermektedirler. Bu çalışmada harici müdahalelerin genetik ağlara etkileri konusu üzerinde çalışılmıştır. Etkiler farklı gen ifadesi analizi kullanılarak belirlenmiştir. Farklı gen ifadesi problemi üzerinde yöntemin gücünü artırmak için ilave çalışma da yapılmıştır. Kesikli dinamik düzenleyici ağlar için kontrol problemi formüle edilmiştir. Ayrıca bu formülasyonda, sonlu kontrol, sonsuz kontrol, ağ durumu ve müdahale masraflarının farklı biçimlerde hesaba katılması gerekleri gibi çeşitli hususlar da göz önünde bulundurulmuştur. Küçük ve büyük ölçekli ağlar için kontrol düzenleri önerilmiş ve örneklendirilmiştir. Bir durum çalısmaşı üzerinde önerilerin nasıl çalıştığı gösterilmiştir. Aynı zamanda, önerilen yöntemlere olan gerekler ve bu yöntemlerin etkililiği verilmiştir.

Anahtar Kelimeler: mikroarray, veri madenciliği, öğrenme, farklı gen ifadesi, genetik düzenleyici ağ modelleme, genetik düzenleyici ağ kontrolü, saccharomyces cerevisiae

To

AĞCA & Ağca

ACKNOWLEDGEMENTS

I would like to express my sincere and endless gratitude and thanks to my supervisors, Dr. Faruk Polat and Dr. Reda Alhajj, for sincere and endless support and concern.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The last decade witnessed the revolution of biological sciences; transition from data poor science to data rich science. High-throughput technological advancements made this transition possible. Maintenance, analysis and interpretation of huge data made Biology a computation intensive science as none of them can be done manually as used to be. This in turn created an interdisciplinary field of science called *bio-informatics*, an intersection of Biology, Computer Science and Statistics.

After the announcement of the complete genetic sequence of bacterium in 1995 and S.cerevisiae in 1997, genetic blueprints of lots of organisms have been announced to date including homo-sapiens. The analysis (similarity search, alignment, gene finding, prometer analysis, etc) of such sequences categorized as structural analysis. Structural analysis is usually performed on gene sequence data and gene product (protein) data obtained with sequencing technologies [38, 11]. The former is known as structural genomics and the latter is structural proteomics. On the other hand, functional genomics/proteomics studies functions and roles of genes/proteins in the organism. Functional (behavioral) genomics/proteomics studies are mostly using *microarray technology.*

The advent of the microarray technology enabled researchers to measure the expression levels of thousands of genes in a single experiment. As a result, a huge amount of gene expression datasets are being produced. The challenge is finding methods to analyze them for different needs, and to interpret the obtained results for discoveries.

Microarary gene expression data analysis tasks include, *finding differentially expressed genes, clustering, classification, regulatory network induction*, and *controlling genetic networks.* To date most of the efforts are devoted to the first four of them. Controlling genetic networks is relatively new compared to others. This is

because the others are usually pre-requisite for the controlling process. To be specific, to explore the controlling process almost correct genetic models are needed. The need for almost a correct model makes the controlling process and the other tasks tightly coupled. Having this identified, the problem studied in this thesis is *controlling genetic regulatory networks* as defined next.

1.1 Problem Definition

To analyze the temporal behavior (dynamics) of cells, a number of microarray experiments are conducted on samples collected at different times. If each experiment is associated with a time information, a time series gene expression dataset is obtained.

Given the time series gene expression data, the dynamic model that generated the data can be reverse engineered. Numerous approaches are proposed to construct the model. The approaches include, boolean networks [8, 9, 61, 82, 81, 58], probabilistic boolean networks [78, 80, 79], bayesian networks [41, 55, 95, 26, 67], differential equations [13, 12], petri nets [65], linear/quasi linear regression [17, 91], markov chains [54], etc.

The variety of approaches is mainly because the mechanism (cell model) generating the data is not known exactly. The other reasons include complexity, dimensionality, size, and quality of data, among others. But, all the approaches try to recover the underlying true model (or a simplification of the true model). The models are usually generated for human consumption. That is, the results are presented to the biologists for interpretation. On the other hand, using the model for other purposes is also possible. One of the directions is the *controlling* process.

By controlling what is meant is interacting with the model in a certain way for achieving a specific goal. Given the model, it can be simulated for any starting state; this way, the resulting states can be observed. If we pursue to reach particular states, intervention to the model is required most of the time as normal transitions may not lead to there.

Given the model and the available control actions (interventions), deciding on the kind of intervention for every state is known as policy generation. The objective is to generate optimal or near-optimal policy which increases likelihood of being in desired states.

The kind of intervention can be dichotomized as *internal* or *external*. Internal interventions are usually in the form of gene deletions and gene over-expressions (in the model this corresponds to setting respective variable to its lowest and highest

values, respectively). On the other hand, external interventions include applying treatments (chemical, radiation, etc.) or forcing to live under some stress (e.g. heat shock, acidic/salty environment, etc), where deciding on the effects of respective action on the model is not trivial. In both of the cases, effects of every action on the model should be identified, i.e., how actions change the dynamics of the underlying model. In case of internal actions, it is given by definition, e.g., deleting a gene means setting its expression level to zero. But, in case of external control, the effects of them on the model is not easy to understand. Automatic methods for this have not been studied in the literature yet.

Although every state (of cell or model) can be attributed as desirable or undesirable (cost of state). They should be quantized in a way to be biologically valid. Also, needed is to account for the cost of intervention itself. This is because for example, even though some interventions provide high percentage of success, they might not be feasible economically. Clearly, like state desirability, costs of external action interventions should be quantized. Both of these require good domain engineering. It must be emphasized here that quantization of desirability of states and costs of interventions are just basics that must be accounted. In other words, there may be other relevant measures or these costs can be broken down, e.g., cost of interventions can be defined separately in terms of labor, money, time, etc.

From these discussions, it is clear that the problem is not unique. That is, two researchers can easily come up with completely different instances to be solved. For instance, one can desire (based on horizon): "I will apply intervention lifelong", and another can state: "I can only apply 3 steps of intervention" etc. For another instance, the requests can be based on the cost of interventions, e.g., one can have cost of interventions, while others do not care this, etc. So, the key issue here is while developing methods how to address all these varieties. Even, they may be unaware of the kind of intervention that can be applied. In such case and based on the objective, the interventions should be evaluated and their effect on the model should be estimated.

To sum-up, the problem can be stated as *how to control dynamics of regulatory networks for various accounting cases of both the desirability of states and the cost of intervention.*

1.2 Motivation

The work by Datta *et al* [16] presents a dynamic programming method for controlling dynamic discrete genetic regulatory networks. Specifically, the method derives an optimal policy based on the definition of the control problem. By executing this optimal policy, desirable states of the model can be reached while avoiding the undesired states. This is akin to applying treatments in medicine to bring an ill patient to healthy state. Obviously, an optimal policy in this case is choosing proper treatment option for the case.

Although their work is pioneering in this area, the following weaknesses have been identified as a result of my extensive analysis.

1. To apply the method, the domain must be engineered well;
2. The method only scales to small networks;
3. The method developed only addresses finite control;
4. Although it is termed external, it is effectively internal.

The first weakness originates from the fact that both the desirability of states (or costs of states) and cost of interventions are assumed additive for a given horizon of control. So, for example when the horizon changes the domain should be re-engineered to make the addition of these costs makes sense. The second weakness originates from the assumption of tabulating all model components and also from exact computation of value functions. So, large networks (which are not amenable for efficient tabulation) can not be handled because of huge storage requirements and huge time requirements for sweeping this storage. Their method assumes that the state just after the control period is important but nothing else, i.e., they do not care about the fact that there are further transitions even the control is terminated (the third weakness). In other words, their method is not general in terms of horizon of control and ignores the monitoring stage. They assume that effects of interventions on the model are known and they develop optimal control strategies based on that (the fourth weakness). But, in cases for which the internal effects of external actions are not known apriori, the effects must be determined for each external actions.

Unfortunately, to the best of my knowledge, these weaknesses are not addressed in the literature. This study is mainly motivated to tackle these deficiencies. So, this thesis develops a principled approach for controlling dynamic discrete regulatory networks.

The first three of the above four weaknesses can be modeled under the umbrella of developing control strategies. But, determining effects of interventions on the model require other approaches. Here, I realized that the *differential gene expression analysis task* can be used to automate this process.

Although, sophisticated methods for finding differentially expressed genes (e.g., *PaGE* of Manduchi *et al* [22], and *q-values* of Storey [45]) have been proposed and already cited in the literature. If the number of differentially expressed genes is small, then this can lead empty set of intersection of these genes and the genes included in the model. So, this is another motivation for finding more differentially expressed genes within a given confidence level so as to increase power.

1.3 Contributions

This thesis contributes in different aspects to the ongoing research in the area of analyzing gene expressions. The contributions are enumerated next chapter by chapter. To start with, contributions in Chapter 3 are:

- Improving the power of PaGE: This is done by estimating the prior probability involved in the confidence computation.
- Computing q-values for up-regulation and down-regulation separately: In its basic form, q-values are computed on the basis of differential expression. Computing them for up and down regulation separately enables us with improved power and more specific interpretation.
- Finding the up and down cutoffs simultaneously: In PaGE, up and down cutoffs are found separately and independently. Here, I show how they can be computed simultaneously and for non-symmetric rejection regions.

Also, experiments on real and synthetic data are performed to show the effectiveness of the proposed methods. Parts of this chapter mainly those focusing on generating patterns from differentially expressed genes has already been published [3, 6].

Contributions in Chapter 4 are:

- Deciding on the required number of non-random time series experiments for induction of Boolean Networks: Using information theoretic lower bounds, the required number of experiments is proven to be $\Omega(\sqrt{n} \cdot logn)$.

- Determining the effect of interventions (external control) on a given model: This is done by applying differential gene expression analysis (from Chapter 3) between the dataset of the condition for which the model is obtained and the external control condition.

- Using multiple predictor sets for inducing Markov Chains directly: It is considered that the predictive power of some other predictor sets can be very close to that of the best predictor. The utility of this is shown in Chapter 6.

It is also noted that boolean networks, probabilistic boolean networks, discrete dynamic bayesian networks, and direct Markov chain models can be abstracted as Markov chains. This is especially important for developing the optimal control strategies presented in Chapter 5.

Contributions in Chapter 5 are:

- A principled approach for the control problem of dynamic discrete regulatory networks: The control problem and its ingredients are formulated and cases of the problem (Finite Control (FC), Finite Control/Infinite Monitoring (FCIM), Finite Control/Finite Monitoring (FCFM), Infinite Control (IC)) are identified.

- Generalization of Datta *et al* [16] control method: Their approach is identified as addressing only the FC case. It is further generalized for FCFM, FCIM, and IC cases. Discounted settings are addressed to account for uncertainty as well. For all of the cases, optimal control strategies are given.

- Derivation of optimal control strategies for cases where the state-action costs are unavailable: The problem is formulated and for all of the four cases, formulas for control strategies are derived.

- Completeness of finite control methods for sequential control: FC, FCFM, and FCIM together are shown to be complete for sequential control applications.

- Design of a multi-objective control strategy: A general multiple-objective control method is introduced. This is motivated from the cases where the state costs and state-action costs are not additive.

- Development of methods for scaling up to large state-spaces: The model components are stored implicitly (so compactly) instead of the tabulation used for small scale networks. The methods employed are approximate computations

of value functions and random sampling. The effectiveness of these methods are shown in Chapter 6.

On a running example, the utility of FC, FCFM, FCIM, IC, and multi-objective control approaches are shown. Parts of this chapter have already been published [4, 5].

In Chapter 6, S.Cerevisiae (budding yeast) is selected as a case study to show how the methods proposed in Chapters 3, 4, and 5 can be used in a holistic manner. Holistic use of these approaches is shown to be important as the control problem can not be posed and solved in isolation. Also, all the steps starting from the raw microarray data and ending with optimal control policy are shown.

1.4 Roadmap of the thesis

This thesis is organized in 7 chapters. In addition to this introductory chapter, the other 6 chapters are organized as follows.

Chapter 2 includes an overview of the background necessary to understand the contributions presented in the other chapters. In particular, a brief introduction to molecular biology, gene expression data and microarray technology are given. Overview of the data mining process is presented. In this context, pre-processing methods and data analysis tasks are briefly given with emphasis on their usage for gene expression data. The considered pre-processing methods are used for pre-processing the dataset utilized for the case study presented in Chapter 6.

In Chapter 3, one of the analysis methods for gene expression data, called finding differentially expressed genes, is presented along with our contributions. The PaGE method is presented along with its main drawback of conservatively underestimating the true confidences. This study attempts to solve this problem by estimating prior probabilities needed to compute confidences. I extend the work of Storey [45] for this estimation for one-sided tests. I further generalize the problem definition of PaGE to find all cutoffs for a given minimum confidence level. Under this formulation of the problem, Storey's estimation method can be directly incorporated to improve power. Examples are also given to show and assess the power of the proposed methods.

In Chapter 4, genetic regulatory network modeling methods are surveyed and how the DEGs analysis can be used for determining effects of interventions on induced genetic networks is explained. The proof on the required number of experiments for induction of boolean networks from non-random time series data is provided as well.

In Chapter 5, methods for controlling genetic regulatory networks are given. First, the problem is formalized. Then based on the availability of the state-action cost function, formulas for optimal policies are derived for the addressed cases of FC, FCIM, FCFM, and IC. Also a multi-objective control method for this problem is provided. On a running example, proposed methods are evaluated. Finally, to solve the scaling problem, methods for medium size and large size networks are developed.

In Chapter 6, a case study for S.cerevisiae organism from pre-processing to genetic network inference to controlling is experimented. 23 genes are selected for model building using prior biological knowledge and a network is induced using Markov chains approach. This network is termed as large and forward sparse sampling algorithm is applied to find near-optimal control policy. This set is further reduced down to 9 genes (medium network) and 6 genes (small network) to explore the effects of different actions under the proposed methods. Control actions and their internal effects are determined using differential gene expression analysis.

In Chapter 7, a summary of the work done in this thesis is provided along with the contributions. Future research directions are also outlined.

CHAPTER 2

MINING GENE EXPRESSION DATA

This chapter prepares for the necessary background required to understand the other chapters. It includes overview of molecular biology, microarray gene expression data, the ingredients of data mining, and literature review on pre-processing and analysis tasks for gene expression data. The pre-processing methods presented are employed in Chapter 6.

2.1 Molecular Biology

Molecular biology concentrates on the basic building constituents of excess number of species to analyze their structure and functionality. In this sense, the estimated number of species is between 5 and 50 million [11]. Species can be broadly classified as prokaryotic (e.g. bacteria) and eukaryotic (e.g. cat, yeasts, worms), depending on the kind of their cells. In prokaryotic organisms, genetic material is not enclosed within a nucleus; it is floating in the cytoplasm. On the other hand, eukaryotic organisms are complex and genetic material is enclosed within a nucleus. Most of the eukaryotic organisms are multi-cellular, but there are uni-cellular eukaryotic organisms as well, like yeasts. Species show very diverse number of cell counts, e.g., 1 for bacteria and 6×10^{13} for humans [38].

In most of the multi-cellular organisms, cells form tissues (a tissue corresponds to a group of specialized cells), and tissues form organs. For example, in human there are 320 different tissue types. Cell is in turn composed of nucleus, cell wall, cytoplasm and a number of organelles (golgi, mitochondria, etc). Organelles work in an orchestrated manner. For instance, mitochondria provides cell respiration while golgi provides for the transportation of micro molecules.

DNA (Deoxyribose Nucleic Acid) is stored in the nucleus and contains the genetic material. All the cells in an organism have the same genetic material; however, cells

do differ and specialize. So far, it is not totally and exactly known (there are a number of hypothesis but no consensus) how the same code specializes and creates diversity across tissues.

DNA is organized into chromosomes (23 pairs for human) and has a three dimensional double stranded helix shape (one named Watson and the other Crick to honor the discoverers). Each strand is composed of very long sequence of repeated units called nucleotides. All nucleotides in all the chromosomes make the genome of the organism. The genome size of human is about 3 billion pairs of nucleotides (called base pairs or bp for short). So, human chromosomes contain 130 millions bp on the average. Genome size and chromosome count differs among organisms.

Each nucleotide is composed of 3 main parts: sugar group, phosphate group and base group. Depending on the chemical structure of the base group, every nucleotide is classified into one of four types, namely A, T, C and G. Given the sequence of one strand, the sequence of the mating strand can be determined exactly. This is because only A-T and G-C pairings are possible, and connected by weak hydrogen bonds. This is clearly a redundancy; but this redundancy is important when one strand is damaged and needs repairing. Also, only this redundancy makes cell division possible.

Some specialized regions in the long strands are called *genes*. They encode *proteins*, which are the functional building blocks for cells. Not all the nucleotides are part of a gene, which means that some genome parts are not functional at all. These parts are called *introns*. Estimated number of genes for human is about 50000. Average size for human gene is about 3000 bp, i.e., about 95% of human genome is composed of introns.

Basically, corresponding to a gene is a protein. Proteins are composed of sequences of amino acids. Each amino acid contains exactly 3 consecutive nucleotides, e.g., the nucleotide sequence ATG encodes amino acid *methionine*. For instance, a gene with 3000 bp nucleotides encodes a protein with 1000 amino acids. Proteins are very important for cell life as they function as sensors, actuators, transforming substances, enzymes, immune system, basic building blocks, breaking foods, etc. Proteins are identified from sequence information of constituting amino acids. Besides sequence information, the 3D shape is very important as they form complexes based on 3D shapes. Predicting the 3D shapes of proteins from amino acid sequence is an active research area.

Some of the proteins are known as *transcription factors* (TFs). They bind to a specific region (*prometer*) of genes on the DNA to enable or disable the production

of corresponding protein to that gene. So, they adjust the abundance levels of other proteins and themselves. This is a complicated machinery and resembles the auto-regressive feedback systems. *Gene expression* means production of genes and gene expression level is the quantity. According to the central dogma of biology, a protein is produced with two consecutive events called *transcription* and *translation*. During transcription, DNA temporarily opens up at specified location corresponding to respective gene and RNA (Ribo Nucleic Acid) is synthesized by complementation. This RNA is called messenger RNA, since it carries the code of its encoding gene. The messenger RNA moves out of the nucleus and enters the *ribosome* (a machinery producing protein from the messenger RNA). The process of ribosome is known as translation.

Thousands of genes can express this way in parallel, and the overall process is known as *genetic regulation*. When and which genes are going to be expressed, and when and which genes are going to cease expression is again encoded in the genome. This is a complex process and there are extensive research on this topic to recover the governing rules (more details are provided in Chapter 4). The rules may change across tissue types, developmental stages, various diseases, environmental factors, etc.

2.2 Microarray Technology and Gene Expression Data

Microarray is a tool for measuring the expression levels of genes in the genome. This section presents an overview of the *cDNA microarrays*, which is the most common technology [92]. cDNAs can not measure the expression levels directly. They rather measure them indirectly by comparing the expression levels of genes across two samples. So, they give the relative abundance of genes across two samples. This is convenient, since multiple experiments can be done and compared by using the same reference sample.

Microarray experiments basically follow the steps enumerated next [23]:

1. Preparation of Target DNA
2. Preparation of Slides
3. Printing of DNA microarrays
4. Preparation of Probes
5. Hybridization

6. Scanning

In Step 1, target genes are selected and amplified. The set of target genes can change depending on the experiment. For organisms (like yeast) having $\leq 10,000$ genes, the set usually contains all the genes. But, for higher order organisms (like human with $\approx 50,000$ genes), the set is usually selected from a library, which is a subset of all the genes. This is mainly because of the capacity of the microarrays used in the process. The typical capacity of a microarray is $10,000 - 20,000$ genes [92].

In Step 2, the type of slides are decided and pre-processed. The pre-processing include the coating and cleaning of the slide. Usually glass microscope slides are used.

In Step 3, the DNA prepared in Step 1 is printed on the slide prepared in Step 2. This is done by an automated hardware (robot) called *arrayer*. An arrayer has print tips which are used for slightly tapping on the slides and leaving a small drop of the DNA. Every tapped place is called *spot*. Each spot is filled with a pre-specified gene. Spots are almost equally spaced and aligned both horizontally and vertically, resembling matrices. During the printing process, the sharpness of the print tips changes, which in turn changes the geometry and size of the spot on the slide (i.e., print tips do not behave identically). For this reason, print tips are periodically replaced with new ones.

In Step 4, two probes from two samples in two different conditions are selected. The mRNA is extracted and reverse transcribed to complementary DNA. Then each probe is labeled with fluorescent dyes. Usually, one probe is dyed with red and the other with green.

In Step 5, the two probes prepared in Step 4 are hybridized on the same slide (Figure 2.1). Since, the genes in the probes and the spots are complementary to each other, they bind. Each gene only binds to its complement spot on the array.

In Step 6, abundance of genes can be determined by measuring the intensity of red and green fluorescent dyes. The idea is that if a gene is highly expressed in a sample then its product is abundant and its intensity is high. By measuring the intensities, expression levels can be deduced. The intensity measurement is done by a separate hardware called *scanner*. It scans all the slide and generates its colored picture. The output of the scanner is high-resolution color picture of the slide.

The color picture output by the scanner is processed by image analysis programs to detect boundaries of the spots. Because of the inherent noise, the scanned image is not perfect. The background is expected to be completely black but hybridization and scanning errors introduce noise. The two channels (red and green colors) are

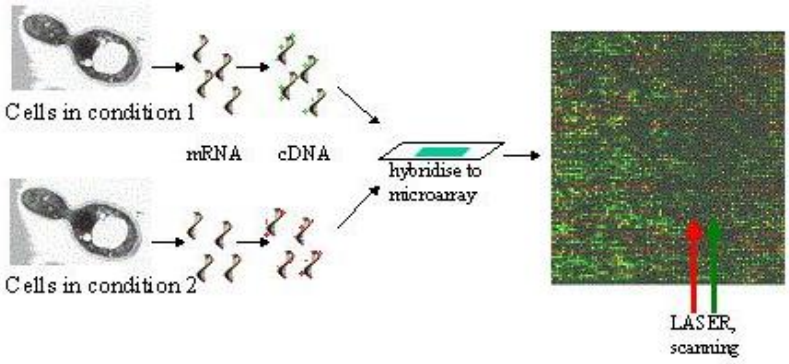


Figure 2.1: Steps of microarray experiments (from Brazma *et al* [11])

measured separately and combined to represent the gene expression levels. If a particular spot is red, then its corresponding gene is more abundant in probe dyed with red than the one dyed with green and, vice versa. If they have almost equal quantity, the color is yellow (red+green). Having the spot black means that the gene is not expressed in both samples.

To summarize, a microarray experiment contains probe preparation, target preparation, scanning and low level data analysis (image processing). After these steps, the raw dataset is obtained. In the raw dataset, there is a row for each gene on the slide. Each row has the two intensity levels for green and red channels. Also each row has the two background colors for these channels (see Section 2.4).

2.3 Overview of Data Mining

Data mining (DM) is defined as the nontrivial extraction of implicit, previously unknown, interesting, and potentially useful information from data [14]. Data mining has wide range of application domains including (but not restricted to), medicine, control theory, engineering, marketing and finance, public administration, fraud detection and scientific databases (e.g., biological databases). Basically, data mining process consists of the following steps in order [14, 21],

1. Defining the problem

2. Collecting and selecting data

3. Preparing and preprocessing data

4. Selecting and applying a model or an algorithm

5. Final evaluation

Data mining process starts with a well defined problem. Then, relevant data to the defined problem is collected. Among the collected data useful data is decided to be selected for use. The selected data is prepared and preprocessed. The selected model or algorithm usually requires data in a certain format and conforms some rules. In case the raw data selected can not be used by the selected algorithm, it is preprocessed. On the preprocessed data, the selected algorithm or model is applied. Performance (correctness, validity, run-time performance, etc.) of the algorithm is assessed in the final stage. The result produced also needs to be interpreted towards the objective of the problem specified to get *knowledge*. In short, data mining starts with data and ends with knowledge.

Closely related to data mining is *structural querying* (e.g., SQL) and *on-line analytical processing* (OLAP). They also query data and extract useful information. OLAP works on multidimensional data and extends the aggregation operators of SQL. Common to both structural querying and OLAP is the precise results they produce. On the other hand, data mining results are almost always vague. The other difference is that structural querying and OLAP have predefined operators, while DM has no predefined operators. Yet, DM emerged as a discipline at the intersection of algorithms, artificial intelligence, statistics, information retrieval and databases. Techniques of DM include, parametric-nonparametric modeling, similarity measurements, decision trees, neural networks, genetic algorithms, etc [21].

To date, several DM tasks have been identified, including *characterization, regression, classification, clustering*, and *association rule mining*. Among them the last three are more common. The techniques considered are used for these tasks for problem solving. It must be emphasized here that not all the tasks are applicable to all DM problems. That is, problem specification determines the kind of task to be used.

In the following section, the common pre-processing techniques applied for microarray gene expression data are presented. The considered techniques are used in the case study presented in Chapter 6. Data mining tasks for microarray gene expression data are presented in Section 2.5.

2.4 Pre-Processing Gene Expression Data

Omitting the categorical data (location on the slide, the print tip used, etc), a microarray experiment gives four values (one foreground and one background intensities for each of red and green colors) for every gene. Let the number of genes be n, and let the intensities of red and green colors be Rf_i and Gf_i for gene i, $i = 1, 2, \ldots, n$. Usually, the intensities in the vicinity of spots are not zero. This means that, Rf_i and Gf_i contain some background intensities even if there is no binding. So, the background color must be subtracted for finding true intensities for both channels. Let, background colors be Rb_i and Gb_i for gene i and for red and green colors, respectively. So, the correct intensities of spots are calculated as follows: $R_i = Rf_i - Rb_i$ and $G_i = Gf_i - Gb_i$. The reason for the subtraction is because of the fact that the foreground color contains both the true expression intensity and the background color.

The relative gene expression level for gene i is $\frac{R_i}{G_i}$ (green color is assumed to correspond to the reference sample). The ratio is ≈ 1 in case of equal expression, and > 1 for over expression in the experimented sample, and < 1 for over expression in the reference sample. Usually, the $\frac{R_i}{G_i}$ values are log-transformed for analysis. This is done for ease of interpretation because the equal amount of under and over expression becomes symmetric around 0, also note that 0 corresponds to equal expression. Besides ratio, $M_i = log_2(\frac{R_i}{G_i})$ and $A_i = log_2(\sqrt{R_i \cdot G_i})$ values are computed for every gene. Clearly, M_i provides the information of relative expression and A_i provides the information of total intensity. $MvsA$ plots (Figure 2.2) are very helpful for explorative data analysis.

2.4.1 Normalization

As it is already discussed in Section 2.2, lots of steps are involved in the experimentation process, each introducing its own error/bias/noise. So, the resulting measured expression levels are not fully accurate, i.e., contain noise. Fortunately, this noise can be removed with high rates using normalization. Normalization in this context refers to removing systematic errors. This makes sense because all genes undergo the same procedure. In the microarray experiments, systematic biases are assumed to be multiplicative. So, if the overall factor is determined then systematic errors can be removed easily.

Yang *et al* [94] surveys the normalization for cDNA microarray experiments. In the most widely used *global normalization* method, a constant k is assumed to

capture the relation $R = k \cdot G$ for a particular gene set. After finding k, the normalized expression values are shifted as:

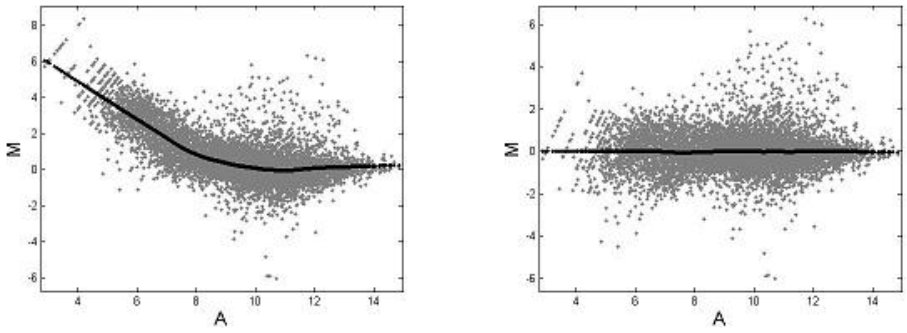
$$log_2 R/G \rightarrow log_2 R/G - log_2 k = log_2 R/(k \cdot G) \tag{2.1}$$

To find the value for k, usually the mean or median of R/G values are used for a particular gene set, which can be decided based on the previous biological knowledge. The methods of *exogenous spiked controls* and *housekeeping genes* all fall in under this category [33]. In the exogenous spiked control methods, equal amount of dummy genes are inserted into both probes. In the housekeeping genes method, a set of genes already known not to change over almost all experimental conditions are used. But, in practice k is selected so that the distribution of log-ratios is shifted to zero over all genes.

The other normalization method (*intensity dependent normalization*) assumes that the multiplicative factor changes across intensity levels, i.e., systematic bias is not the same across intensities. So, k becomes a function of A, denoted $k(A)$. The function $k(A)$ is estimated from the scatter-plot smoothers such as *lowess*, a robust locally linear fitter. Additionally, if the print tip information (a kind of location data) is available, then the *lowess* smoother can be estimated within the print tip data. That is, the genes printed with the same print tip normalized independent of other genes printed with other print tips. In this case, the function is indexed by print tip number t as $k_t(A)$. Although print tip normalization makes all log-ratios within a print tip centered around zero, the spread (variance) of them can change. A post processing to this can roughly make the spreads equal. This is called *scale normalization* [94].

The experimental results reported in [94] show that the shape of *lowess* curve is not horizontal in $MvsA$ plot. This means that intensity dependent biases are considerable, as evident in Figure 2.2. The results also show that using print tip location information is negligible.

All the methods considered are for single slide normalization. However, scaling might be needed for making expression levels of genes across multiple experiments comparable. In such case, scaling methods used for within slide print tip normalization can be used. But, in practice every experimental data is normalized using one of the already discussed methods, and no normalization is applied across experiments.



(a) before normalization (b) after normalization

Figure 2.2: Effect of intensity dependent normalization (on M *vs* A plot)

2.4.2 Missing value imputation

Although there are analysis methods (like Bayesian networks) capable of operating on missing valued data, some analysis methods (like boolean networks) require complete data for operation, so bringing the *missing value imputation* is an issue.

Assume that the number of genes is n and the number of experiments (samples) is m. In the $n \times m$ matrix, some entries might be missing or not computed. This is because of noise and very low level of measured intensities; some genes are not assigned a value and the field is left blank, so causing missing values. Estimating and filling the blanks in the dataset is known as *imputation*.

There are some simplistic approaches including, imputing with a fixed value (usually 0), and imputing with the average of non-missing gene values. The subject has been well-studied in statistics, like likelihood methods, but almost all of them are model-based. Since, the missing value mechanism may change across experiments and it is not known exactly, these sophisticated methods are not applicable. For this reason, simplistic approaches are usually the case.

Troyanskaya *et al* [88] reviewed the missing value estimation methods for cDNA microarrays. They explored *KNNImpute*, *SVDImpute* and *row averaging* methods. Row average of a gene is the average value of its expression values over a number of experiments. The authors favor for the KNNImpute method, which first finds K genes similar to the gene the value for which is to be estimated. Then, a weighted average of the values for the most similar K genes on the same experiment determines the missing value. Here, similarity metric must be decided; the authors discuss the Euclidean Distance as an appropriate norm. After applying KNNImpute for every

missing value we get a complete dataset.

2.4.3 Discretization

Discretization is the process of converting continuous (real valued) data into discrete values. The purpose with discretization is two-fold; first many algorithms require discrete data, and second for some tasks improved performance over continuous values is possible. There are numerous methods proposed in the literature. Liu *et al* [62] provides a hierarchical framework for existing methods. Critical to the problem is selecting the number of discretization levels. Methods automating the selection of the number of discretization levels is also available (such as CAIM algorithm of Kurgan *et al* [57] and ChiMerge method of Kerber [62]).

The simplest and most common methods are *equal width* and *equal frequency*. Given the number of discretization levels, the former divides the interval between minimum and maximum values for a given attribute into equal length intervals (bins). These intervals are numbered in ascending order, and the values within the interval is assigned to respective bin. The latter does the same thing, but it ensures that each interval has equal number of attribute values. So, in the latter the bins may have variable width, on the other hand in the former bins may have variable instances. Besides these two basic methods some statistical and ad-hoc methods have been developed.

Statistical methods are general purpose. They include Bayesian discretizers, enthropy-based, information gain, and Chi2 methods, etc [62]. Ad-hoc methods are usually application specific. For example, for microarray gene expression data, usually the number of the discretization level selected is 3, to represent over-expression, under-expression and baseline. This leveling approach is adopted by microarray community, e.g., [26, 95, 20]. Friedman *et al* [26] discretizes based on the *log* transformation normalized expression data. The cut-point (boundary between two consecutive levels) they use is 0.5 and -0.5, i.e., values larger than 0.5 are over-expressed, smaller than -0.5 are under-expressed and baseline otherwise. On the other hand, mean (μ) and standard deviation (σ) of gene profiles across samples are used as well [95, 20]. Usually, $(\mu + \sigma, \mu - \sigma)$ or $(\mu + \sigma/2, \mu - \sigma/2)$ are selected as cut-points.

2.5 Gene expression data analysis

Microarray gene expression data analysis can be grouped under four basic categories, finding differentially expressed genes (DEGs), classification, clustering, and model

induction. I note that although association rule mining tasks can address microarray data, the literature lacks use of this task. One possible use is explained in Section 7.1.

Association rule mining addresses finding data dependent rules in the form of *if-then* rules. The items (attributes) in the rules are frequent item sets, meaning coexistence rate (*support*) of these items in the dataset is more than a pre-defined threshold called minimum support. Also, the *confidence* of the rules must be above a certain threshold called minimum confidence. A number of algorithms have been developed for finding such rules, among them the **Apriori Algorithm** and its variants are more popular [21]. Rules found by these algorithms show tendency of relationships between attributes, a good clue for understanding the domain.

Since DEGs analysis is considered in detail in the next chapter and model induction is considered in the following chapters, only clustering and classification are reviewed here.

2.5.1 Clustering

Basically, *Clustering* is grouping similar patterns containing no class information. Jain *et al* [44] provides a good review of clustering. Actually, clustering has been extensively studied in the literature as there are hundreds of algorithms proposed using various techniques such as genetic algorithms, neural networks, simulated annealing, graph theory, etc. The variety is mainly because of the diversity of domains, e.g., categoric/numeric data, large/small volume, etc.

The algorithms can be dichotomized as follows, agglomerative/divisive, monothetic/polythetic, hard/fuzzy, deterministic/stochastic, incremental/non-incremental, and possibly others. For example, the well-known *k-means* algorithm is agglomerative, polythetic, hard, stochastic and non-incremental.

The integral part of the clustering is similarity metric. Objects are clustered based on the similarity measure. Most of the time, algorithms require only the similarity matrix as an input, i.e., not the patterns themselves. The similarity metrics include, Euclidean distance, Manhattan distance, Mahalanobis distance, Pearson correlation, etc. The similarity metric is important since different metrics used with the same algorithm usually give different results.

Also important is deciding on the number of clusters a dataset contains. Most of the algorithms (like k-means) require this value as an input. But some algorithms estimate this value from the data and clusters accordingly.

Clustering is one of the most applied tasks to gene expression data. Both genes or

samples can be clustered, or they can be handled together (this case is known as *two-way clustering*). The clustering of the genes is found to be useful for determining the their function. Particularly, if a gene with unknown function clusters together with a gene with known function, then it is a sign that they have the same (or similar) function. That is co-expression means co-regulation. Clustering samples is also useful as this corresponds to identifying similar tissues, environmental conditions, drug treatments, etc. Since gene expression data is high dimensional, clustering also helps in dimension reduction for other tasks such as modeling.

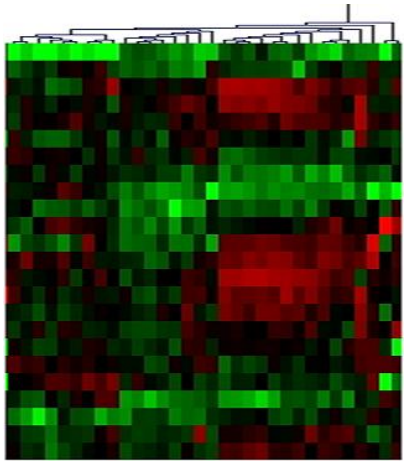


Figure 2.3: A dendrogram representing a sub cluster of cdc15 dataset

K-means, *hierarchical clustering* and *self-organizing maps (SOM)* algorithms are common for gene expression data. Particularly, hierarchical clustering is most common as it also provides explorative data analysis. That is, it shows the relationship at all the levels. The output of the hierarchical clustering algorithm is a dendrogram where leafs are individual genes (in case of clustering genes). At the leaf level, every gene is a cluster and they unite bottom up pairwise and reach just a single cluster containing all the genes at the top. Figure 2.3 shows a sub-cluster of *cdc*15 dataset having 24 time course data for every gene (the dataset is described in Chapter 6).

One of the pioneering work of applying hierarchical clustering to gene expression

data is by Eisen *et al* [24]. The authors applied hierarchical clustering to budding yeast data and detected clusters. Application of SOM to gene expression data is studied by Hautaniemi *et al* [34]. The article by Jiang *et al* [47] provides a good survey of clustering analysis for gene expression data.

2.5.2 Classification

Classification is attributing predefined classes to given patterns. Usually, it involves two stages: 1) Model building; and 2) Applying model for attributing class labels to given patterns (possibly novel). The first step is known as induction, and the second step is prediction. There are numerous techniques for model building including, decision trees (e.g., ID3, C4.5), support vector machines, neural networks, simple statistics (e.g., 0R and 1R), naive bayesian, k-nearest neighbor, and rough sets. Classification can be dichotomized almost with the same dimensions as clustering.

Classification is studied by the Machine Learning community and numerous algorithms and techniques have been discovered with applications to real world problems, including hand-written character recognition, speech recognition, and tissue classification using gene expression data, etc.

Described in the literature are several approaches on applying classification on gene expression data. For instance, Dudoit *et al* [85] compares the different classification methods on 3 gene expression tumor dataset. The authors also explore the aggregating classifiers (bagging and boosting). Hedenfalk *et al* [40] identifies the gene set to differentiate between two classes of tumors, namely BRCA1 and BRCA2. Actually their study is related to finding differentially expressed genes, but can also be used for classification. Keller *et al* [51] applied naive Bayes classification on three datasets for tissue classification. Fuzzy classification methods are also employed [69, 93]. The use of boosting methods has been studied for the classification of gene expression data as well, e.g. [63].

Feature selection is an important integral of classification as this boosts the predictive performance. Ding *et al* [20] explores this for gene expression data. It is also found that, feature selection leads to improved classification performance.

CHAPTER 3

FINDING DIFFERENTIALLY EXPRESSED GENES

3.1 Introduction

It is true that using the microarray technology for gene expression measurement is considered as a breakthrough; however, it has some inherent problems. In particular, there are three main problems specific to gene expression data. These are:

1. Dimensionality: the number of genes is expressed in thousands; and hence it is relatively large compared to the number of samples, which is in general expressed at most in hundreds.

2. Noise: the major cause of which is the substantial amount of steps taken to produce data and each step causes some systematic measurement errors.

3. Natural variability: genes tend to have varied expression levels under the same conditions, even if there is no measurement error. This leads to the fact that, in addition to systematic measurement errors, gene expression is inherently variable.

For these reasons, expression levels of genes are generally measured in replicas in order to alleviate these deficiencies. So, if there is a large number of replicas for a specific gene for a specific condition, then we can use such replicas to induce its distribution. But mostly it is not the case, and in general only few replicas are available.

In general, the normalized gene expression matrix consists of real numbers as shown in Table 3.1, where there are 3 conditions and n genes; and for each condition there are 3 replicas. Note that the condition identified as 0 in Table 3.1

Table 3.1: Gene expression matrix with one control and two treatment conditions; each has 3 replicas

Genes	Condition 0	Condition 1	Condition 2
Gene 1	x_{101}, x_{102}, x_{103}	x_{111}, x_{112}, x_{113}	x_{121}, x_{122}, x_{123}
Gene 2	x_{201}, x_{202}, x_{203}	x_{211}, x_{212}, x_{213}	x_{221}, x_{222}, x_{223}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Gene n	x_{n01}, x_{n02}, x_{n03}	x_{n11}, x_{n12}, x_{n13}	x_{n21}, x_{n22}, x_{n23}

serves as control (reference) for the other two treatment conditions. Usually, the control condition measures the expression level of genes under normal condition, and treatment conditions measure expression levels under particular exposures: like heat shock, salty/acidic environment, medication, etc.

The problem of identifying differentially expressed genes (DEGs) can be stated as follows: *given replicated gene expression measurements of two conditions (control and treatment), find a subset of all genes having significant expression levels across these two conditions.* This statement implies that there are some other subsets where the change is not significant (i.e., expression levels are almost the same). So, given the set of differentially expressed genes, we can further group them as over-expressed (up-regulated) and under-expressed (down-regulated) genes. The former set corresponds to genes having significant difference where treatment condition measurements are very high compared to control condition. Genes in the latter set are treated similarly.

In case of a large number of replicas, the average gene expression may be estimated using the well-known statistics approaches. However, classical statistics approaches are very conservative when few number of replicas are available. For 2 and 3 replica experiments, differentially expressed genes are, respectively, $|\overline{x}_{g,t} - \overline{x}_{g,c}| > 22.3\widehat{\sigma}$ and $|\overline{x}_{g,t} - \overline{x}_{g,c}| > 5.2\widehat{\sigma}$, where $\widehat{\sigma}$ is the estimated standard error, $\overline{x}_{g,k}$ is the average expression of gene g in condition k, and the statistical significance level is 1% [48]. Biologically, these values are found very conservative for differential expressions, especially in case only few replicas are available. This motivated researchers to design new methods for finding differentially expressed genes.

Schena *et al* [64] developed a threshold method where expression intensities which are not within half average distance from the overall average are considered as noise, and hence are omitted. For the remaining intensities, differentially identified genes are those which have at least two-fold ratios: $\frac{\overline{x}_{g,t}}{\overline{x}_{g,c}} \geq 2$ or $\frac{\overline{x}_{g,t}}{\overline{x}_{g,c}} \leq \frac{1}{2}$. This method produces results with very small significance (less than 5%), and causes increase

in wrong identification. Finally, there are some other more sophisticated methods to handle this problem, including *Patterns from Gene Expression* (PaGE) [31] and *q-values* [45]. Statistical hypothesis testing is common to both methods.

In this chapter, I first give statistical hypothesis testing. Next, the PaGE method and its main drawback of conservatively underestimating the true confidences are presented. To overcome this drawback, the method of estimating prior probabilities needed to compute confidences is introduced. Then, the DEGs problem is reformulated under the PaGE approach to find all cutoffs (instead of one symmetric cutoff) for a given minimum confidence level. Examples are also provided to show effectiveness of the methods proposed.

3.2 Multiple Hypothesis Testing

Given only two datasets and posing the question: whether these datasets are generated from the same distribution or not requires a statistical hypothesis testing. The datasets can have more than one feature and the question can be restated for a particular feature, i.e., whether the feature values make difference across two datasets. Or the problem can be asked as "find all those features having differential value across these datasets". This kind of problems are studied under statistical hypothesis testing discipline. Note that finding differentially expressed genes forms a specialization of this general problem.

Let $H_g = 0$ be the two-tailed null hypothesis: gene g is not differentially expressed across the given two conditions. Similarly, let $H_g = 1$ be the alternative hypothesis: gene g is differentially expressed (either up-regulated or down-regulated). Finally, let X_g be the random variable measuring the expression level of gene g from a random sample.

Statistically, when the null hypothesis is rejected, there is a probability of wrong rejection, i.e., the decision is not 100% correct. This probability of being uncertain about the decision is expressed as the *p-value*. So, p-value is an important measure for the uncertainty of a particular decision (e.g., gene $g1$ is differentially expressed).

Definition 3.1 (p-value). p-value is the probability of rejecting the null hypothesis against an observation $(X_g = x)$, it is expressed as:

$$p - value(x) = \inf_{\Gamma_\alpha : x \in \Gamma_\alpha} Prob(X_g \in \Gamma_\alpha | H_g = 0)$$

where Γ_α is the rejection region corresponding to significance level α. □

The nested set of rejection regions are usually parameterized with α. Nested property implies $\Gamma_\alpha \subseteq \Gamma_{\alpha'}$ for $\alpha \leq \alpha'$. The interpretation of the p-value is the degree of evidence against the null hypothesis. So, the smaller the p-value, the more evident it is against the null hypothesis. For single hypothesis testing, if the p-value of observation $X_g = x$ is less than a predefined significance value ($0 < \alpha < 1$), then the null hypothesis is rejected (gene g is decided to be differentially expressed), otherwise it is retained.

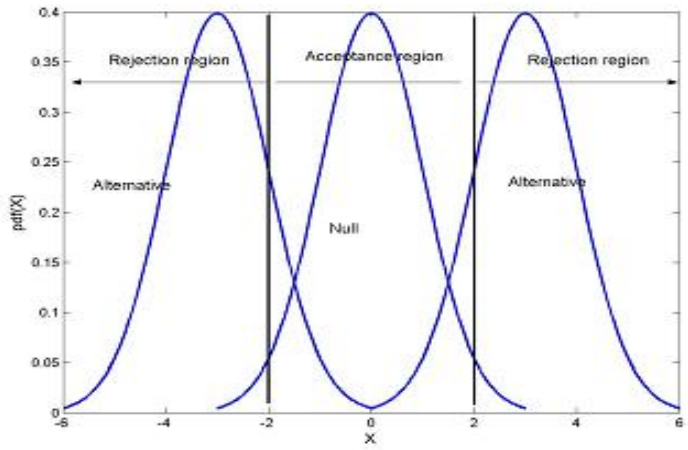


Figure 3.1: Acceptance and rejection regions for two-sided hypothesis

A representative mixture model for a random variable X_g is illustrated in Figure 3.1, where all the components have density of normal distributions. Also shown in Figure 3.1 is a sample rejection region for the null hypothesis of $X_g = 0$.

There are four rates at the heart of the hypothesis testing: *False Positive Rate* (FPR), *False Negative Rate* (FNR), *True Negative Rate* (TNR) and *True Positive Rate* (TPR). These rates express the uncertainties of decisions when the outcome of the hypothesis is given. In other words, these rates numerically quantize the probability of true/false decisions under the true/false null hypothesis.

Definition 3.2 (FPR, FNR, TNR, and TPR). The four rates: FPR, FNR, TNR, and TPR are interpreted as follows (the rejection region is denoted as Γ_α for significance level α):

- FPR is the probability of rejecting the null hypothesis given that it is true, i.e.,
  $FPR = Prob(X \in \Gamma_\alpha | H_g = 0)$.

- FNR is the probability of rejecting the alternative hypothesis given that it is

true, i.e.,
$FNR = Prob(X\ not\ \in \Gamma_\alpha | H_g = 1)$.

- TNR is the probability of accepting the null hypothesis given that it is true, i.e.,
  $TNR = Prob(X\ not\ \in \Gamma_\alpha | H_g = 0)$.

- TPR is the probability of accepting the alternative hypothesis given that it is true, i.e.,
  $TPR = Prob(X \in \Gamma_\alpha | H_g = 1)$

□

Note that it is easy to compute all the rates given in Definition 3.2 when the probability density functions for X are given and rejection regions are set.

Any statistic resulting in the region Γ_α rejects the null hypothesis within the significance level α. If we extend the rejection region (by reducing the value of α), FPR and TPR will decrease, but FNR and TNR will increase. The *power* of the test is $1 - FNR$. The objective of hypothesis testing is finding the most powerful tests. Note that the power of the test depends on the distribution of the alternative hypothesis. So, estimation of the power requires the distribution of the alternative hypothesis which is usually not known as it is the case in gene expression domain.

Parametric methods can not be used because there is no prior information on the distribution of genes in microarray experiments. In other words, the use of non-parametric methods should be adopted. So, given the replicas of particular treatment and control samples, it is possible to compute the t-statistics for any gene g for differential expression by using the following formula under the assumption that genes have differing standard deviations [74],

$$t_g = \frac{\overline{x}_{g,t} - \overline{x}_{g,c}}{\sqrt{\frac{s^2_{g,t}}{n_t} + \frac{s^2_{g,c}}{n_c}}} \tag{3.1}$$

where $\overline{x}_{g,t}$ and $\overline{x}_{g,c}$ are means of replicas of treatment and control conditions with respective standard deviations $s^2_{g,t}$ and $s^2_{g,c}$, and replica counts n_t and n_c for gene g. It is clear that t-statistics favor for large mean differences and small standard deviations and it is a good balance between them.

The p-value for each gene can be computed using the *Permutation algorithm*, under the assumption of no differential expression. The method exploits the fact that permutation of columns of treatment and control conditions are immaterial

when the null hypothesis is true. So, to estimate the null distribution of gene g, the t-statistics is computed for each permutation. From these statistics, the p-value of gene g can be computed non-parametrically easily by using the following formula [29] (in case of two-tailed tests).

$$p_g = \frac{\#\{b : |t_{g,b}| \geq |t_g|, b = 1, 2, \ldots, B\}}{B} \tag{3.2}$$

where B is the number of permutations generated, t_g is the t-statistics for the original sample and $t_{g,b}$ is the t-statistics of b ́th permutation of gene g.

There are usually thousands of features (genes) in microarray experiments, and this necessitates considering thousands of tests at the same time. Therefore, multiple-hypothesis testing rather than single-hypothesis testing should be adapted.

Table 3.2 characterizes multiple-hypothesis testing scenario of $M_{(-|-)}$ number of genes (features). The columns indicate the true case (either null true or alternative true) while rows indicate the decision (either reject or retain).

Table 3.2: Multiple-hypothesis testing scenario of $M_{(-|-)}$ genes

	$H = 0$	$H = 1$	
retain $H = 0$	$M_{(0\|0)}$	$M_{(0\|1)}$	$M_{(0\|-)}$
reject $H = 0$	$M_{(1\|0)}$	$M_{(1\|1)}$	$M_{(1\|-)}$
	$M_{(-\|0)}$	$M_{(-\|1)}$	$M_{(-\|-)}$

The number of false-positives (Type I error), false-negatives (Type II error), true-negatives, and true-positives are $M_{(1|0)}$, $M_{(0|1)}$, $M_{(0|0)}$, $M_{(1|1)}$, respectively. Also, $M_{(0|-)}$ and $M_{(1|-)}$ are, respectively, predicted number of non-differentially and differentially expressed genes; while, $M_{(-|0)}$ and $M_{(-|1)}$ are, respectively, true number of non-differentially and differentially expressed genes; and $\frac{M_{(1|1)}}{M_{(-|1)}}$ gives the *power* of the test.

In multiple hypothesis testing, we can compute the p-values of genes using t-statistics of the other genes. This results in the increased accuracy in p-values since the number of t-statistics is huge. Formula 3.3 given next, can be used for p-value computation [46] from the t-statistics computed using Formula 3.1 (in case of two-tailed tests).

$$p_g = \sum_{b=1}^{B} \frac{\#\{j : |t_{j,b}| \geq |t_g|, j = 1 \cdots M_{(-|-)}\}}{B \cdot M_{(-|-)}} \tag{3.3}$$

In this regard, there are a number of error measurement schemes proposed to

control the error for multiple-hypothesis testing using p-values, including *Per Comparison Error Rate* (PCER), *Family Wise Error Rate* (FWER), *False Discovery Rate* (FDR), and *positive False Discovery Rate* (pFDR). [45];

Definition 3.3 (PCER, FWER, FDR and pFDR). The four rates PCER, FWER, FDR and pFDR are computed as follows:

$$\begin{aligned} PCER = & \quad \frac{E[M_{(1|0)}]}{M_{(-|-)}} \\ FWER = & \quad Prob(M_{(1|0)} \geq 1) \\ FDR = & \quad E[\frac{M_{(1|0)}}{M_{(1|-)}}|M_{(1|-)} > 0] \cdot Prob(M_{(1|-)} > 0) \\ pFDR = & \quad E[\frac{M_{(1|0)}}{M_{(1|-)}}|M_{(1|-)} > 0] \end{aligned} \tag{3.4}$$

After ordering the p-values in increasing order: $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(M_{-|-})}$, methods controlling PCER and FWER estimate a $\widehat{k}$ and reject all hypothesis corresponding to $p_{(1)}, p_{(1)}, \ldots, p_{(\widehat{k})}$. For example, letting $\widehat{k} = max\{k|p_{(k)} \leq \alpha\}$ strongly controls PCER while $\widehat{k} = max\{k|p_{(k)} \leq \alpha/M_{(-|-)}\}$ (known as Bonferroni correction) strongly controls FWER at level α. Strong control means that the procedure does not require the number of null hypothesis ($M_{-|0}$) in estimating the rejection region.

FDR and pFDR are relatively new and more exploratory. Note that FDR and pFDR become equivalent when the rejection region contains at least one gene. Storey [45] presents strong control methods for both FDR and pFDR. Because of the exploratory nature, FDR and pFDR well suit the microarray data analysis. FDR and pFDR can be computed from the p-values as well as the original t-statistics. In case they are computed from p-values, rejection regions can be defined by more concrete concept of t $(0 < t < 1)$, instead of abstract concept Γ_α. In this case, the interpretation is reject all genes having p-values smaller than t. So, FDR can be predicted for the rejection region $[0..t]$ as given next in Formula 3.5,

$$\widehat{FDR}(t) = \frac{\widehat{\pi}_0 \cdot M_{(-|-)} \cdot t}{\#\{p_i \leq t\}} \tag{3.5}$$

where the only unknown item in the formula is $\widehat{\pi}_0$; an estimate of $\pi_0 = \frac{M_{(-|0)}}{M_{(-|-)}}$ (it is considered in Section 3.4). For the microarray domain, π_0 is the prior probability that any gene is not differentially expressed: $\pi_0 = Prob(not\ de)$. Finally, the author defined FDR analogue of p-values called *q-values*.

Definition 3.4 (q-value). The q-value for gene g is defined as the expected rate

of false-positives in the set of genes having p-values smaller than or equal to that of g. i.e.,
$q_g = min_{t \geq p_g} \widehat{FDR}(t)$. $\square$

The interpretation of the q-value for gene g can be stated as follows: q-value of g, denoted q_g, gives the minimum FDR when all genes having smaller or equal q-values than g are selected as significant. For example, assume genes are sorted in ascending order based on their q-values, and consider a gene g having index $i = 500$ and let $q_g = 0.02$. This indicates that among the first 500 genes, on the average at most 10 of them are false-discoveries (not significant) and 490 of them are true-discoveries. However, this does not necessarily mean that the first 490 are true-positives.

As another example, assume we take the risk of 1% false-discoveries, then we should choose q-value cutoff as 0.01. This mechanism also guides researchers to fine tune the selection of genes for further experiments based on the objective of the experiment. In other words, results can be used multiple times without extra computations for differing needs. Another advantage of q-value is related to the fact that by bounding false-discoveries, the amount of waste of time and money can also be bounded with the same rate of false-discoveries beforehand. This is true because in microarray experiments, biologists consider significantly selected genes to experimentally verify in further experiments.

3.3 PaGE Method

The work described in [22, 31] studies the problem of confidence estimation for identifying differentially expressed genes using the ratio statistics. The authors have developed a method called PaGE. In their settings, there are several conditions for every gene, and for each condition there are replicas as presented in Table 3.1. Their process works as follows. First, they select a treatment and a control condition. Second, average values of replicas are calculated. Third, the shifted average expression value of the treatment condition of each gene is divided by that of the control condition. A gene for which the ratio is above a cutoff ratio Cr ($Cr > 1$), is considered up-regulated in treatment condition compared to control condition. The main problem here is how to find $Cr > 1$ values for each treatment-control condition pair with a guaranteed false-positive rate on the average. By achieving this, reasonable confidence can be satisfied. Finally, it is worth noting that down-regulation can be considered in a similar way.

Consider a gene, say g, which is not up-regulated and let $\mu_{g,t}$, $\mu_{g,c}$, $X_{g,t}$ and $X_{g,c}$, respectively, be the true mean value of expression level and the expression level of samples for the treatment and control conditions of g. Then, the probability of false-positives (FPR, a.k.a. Type I error) -the probability of attributing g as up-regulated is:

$$Prob\Big(\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}} > Cr|\frac{\mu_{g,t}}{\mu_{g,c}} \leq 1\Big) \tag{3.6}$$

Based on the fact that $\frac{\mu_{g,t}}{\mu_{g,c}} \leq 1$, an upper bound on Formula (3.6) can be obtained as follows:

$$Prob\Big(\frac{\frac{\overline{X}_{g,t}}{\mu_{g,c}}}{\frac{\overline{X}_{g,t}}{\mu_{g,c}}} > Cr|\frac{\mu_{g,t}}{\mu_{g,c}} \leq 1\Big) \tag{3.7}$$

It is required to find a least cutoff value of Cr, satisfying the maximum false-positive rate s (say 1%, for example). This is because any cutoff larger than Cr increases confidence but at the same time reduces power; Cr is the best balance of FPR and power. Using the fact that the two events on both sides of the symbol "|" are independent in Formula (3.7), we get the following formula,

$$Prob\Big(\frac{\frac{\overline{X}_{g,t}}{\mu_{g,t}}}{\frac{\overline{X}_{g,c}}{\mu_{g,c}}} > Cr\Big) < s\% \tag{3.8}$$

The following approximation is suggested for the distribution of $\frac{X_{g,j}}{\mu_{g,j}}$, $j \in \{t, c\}$ (see [22] for details):

$$\frac{\frac{X_{g,j,k}}{\overline{X}_{g,j}} - 1}{\sqrt{t_j - 1}} + 1 \tag{3.9}$$

where g varies over genes for condition j, k varies over replicas and t_j is the replica count. Values from the dataset are used for the estimation of variables: $x_{g,j,k}$ for $X_{g,j,k}$ and $\overline{x}_{g,j}$ for $\overline{X}_{g,j}$. Let $f_c(x)$ and $f_t(x)$ be, respectively, the density functions (of distribution given in Formula 3.9) of the control and the treatment conditions. Then, for a particular Cr value, the following integral computes FPR.

$$\int_x \int_{s>Cr\cdot x} f_c(x) f_t(s) ds dx \tag{3.10}$$

The correct value of Cr is found by a binary-search: if the result of the integral is above FPR, then Cr is raised, otherwise it is lowered.

Let $Prob(not\ up)$ ($Prob(not\ down)$) be the prior probability that any gene is not up-regulated (down-regulated). In PaGE, the Bayes' formula can be used to

determine the confidence level for each gene for up-regulation and down-regulation separately. Note that the conditional probability in Formula 3.11 is FPR, and $(1 - Confidence)$ is FDR for a given rejection region Cr. In other words, the only unknown term is $Prob(not\ up)$. Since this is neither known nor can be estimated, the approximation is done based on the worst-case that $Prob(not\ up) = 1$.

$$\begin{aligned} Confidence = & \quad 1 - Prob(not\ up|predicted\ up) \\ = & \quad 1 - \frac{Prob(not\ up)Prob(predicted\ up|not\ up)}{Prob(Predicted\ up)} \\ \geq & \quad 1 - \frac{Prob(predicted\ up|not\ up)}{Prob(predicted\ up)} \end{aligned} \tag{3.11}$$

In Formula 3.11, it is clear that if exact value of $Prob(not\ up) \leq 1$ is known, then the power will be increased within the same confidence level. This means that large number of differentially expressed genes will be recalled. In the next section, I will show that this probability can be estimated from the dataset itself.

3.4 Improving the Power of PaGE

This section present one approach for improving the power of PaGE within the same confidence level. The method is based on estimating $Prob(not\ up)$ and $Prob(not\ down)$. As discussed in the previous section, $Prob(not\ up)$ is needed to compute the confidence from the given FPR. Since this prior probability is not known nor can be estimated, the worst conservative case that $Prob(not\ up) = 1$ is assumed to estimate the confidence. In other words, all the genes are assumed to follow the null distribution of no differential expression.

On the other hand, even though most genes are not differentially expressed, not all genes in microarray experiments follow the null-distribution. So, if we estimate the proportion of them, the power of PaGE within the same confidence level can be increased. This is evident from Formula 3.11, where instead of using $Prob(not\ up) = 1$, any value $Prob(not\ up) < 1$ increases the right hand side of the inequality. That is, it produces results more close to the predefined $Confidence$ value.

The method presented next estimates value of $Prob(not\ de)$. Then, $Prob(not\ up)$ is estimated using the similar arguments of estimating $Prob(not\ de)$.

3.4.1 Estimating $Prob(not\ de)$

Figure 3.2 shows the histogram of p-values for the BRCA dataset under the null hypothesis that no gene is differentially expressed. I use the same BRCA dataset used in [46] where there are two conditions (BRCA1 with 7 replicas and BRCA2 with 8 replicas) for 3170 features (genes). The p-values are computed from t-statistics using Formula 3.3. The formula requires computing both t-statistics of every gene and t-statistics of genes under null distributions. I computed t-statistics of genes under the assumption of genes having different variances. Null t-statistics are computed by the method of permutation (Formula 3.1).

If we make two reasonable assumptions: 1) Almost all genes having p-values close to 1 are null; and 2) null genes have uniform distribution of p-values between 0 and 1; exploiting both together, the proportion of null genes can be estimated.

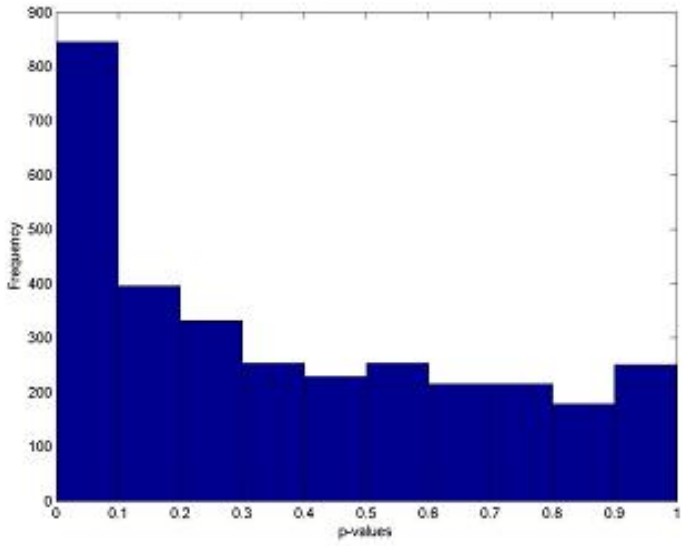


Figure 3.2: Histogram of p-values computed from 250 random permutations under the null hypothesis of no differential expression

As it is clear from Figure 3.2, we see that after a certain p-value of λ, $(0 \leq \lambda \leq 1)$ the right of the histogram shows a uniform distribution. For this example, the value of λ seems to be between 0.2 and 1. If somehow the value of λ is fixed (say at 0.5), the proportion of null genes can be estimated easily using Formula 3.12 (known as Storey estimator);

$$\widehat{\pi}_0(\lambda) = \frac{\#\{p_i > \lambda\}}{M_{(-|-)}(1-\lambda)} \tag{3.12}$$

The rationale of the formula for estimating the fraction of non-differentially expressed genes is based on the fact that most genes in the region to the right of λ must be null, and all null genes should be distributed uniformly between 0 and 1.

For a fixed λ, it is easy to compute $\widehat{\pi}_0(\lambda)$; but the decision on the value of λ is not trivial because there are many p-values, the right of which shows almost uniform distribution. Fortunately, Storey [45, 46] provided two automatic methods for selecting λ, namely *bootstrapping* and *curve fitting*.

Bootstrapping Method

The bootstrap method estimates the value of λ at which the mean squared error (MSE) is minimum. This is because there is a bias variance tradeoff for selecting the value of the best λ. The algorithm developed for this process is given next [45].

Algorithm 1 Bootstrap Estimation of $\widehat{\pi}_0$

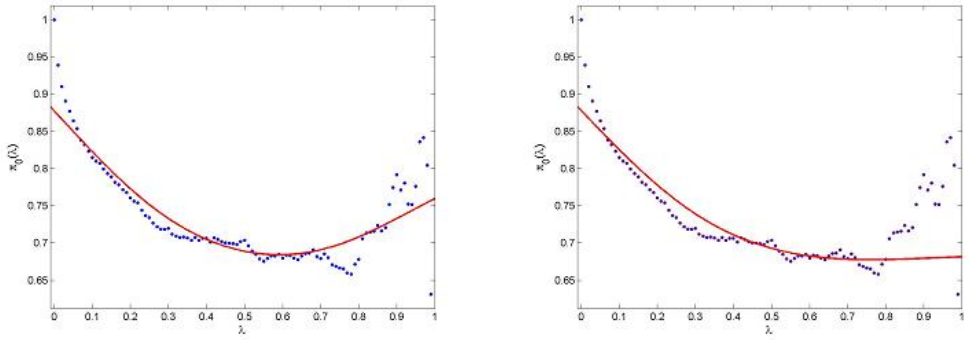
Require: *p-values*, B given
Ensure: Estimate of π_0 computed
  for $\lambda = 0$ to 0.99 by 0.01 **do**
    Compute $\widehat{\pi}_0(\lambda)$ using *p-values*
    $MSE(\lambda) \leftarrow 0$
  end for
  $\widehat{\pi}_0^{min} = min\{\widehat{\pi}_0(\lambda)\}$
  for $b = 1$ to B **do**
    p-values $^b \leftarrow$ bootstrap sample of original p-values
    for $\lambda = 0$ to 0.99 by 0.01 **do**
      Compute $\widehat{\pi}_0^{*b}(\lambda)$ using *p-values*^{b}
      $MSE(\lambda) += sqr(\widehat{\pi}_0^{*b}(\lambda) - \widehat{\pi}_0^{min})$
    end for
  end for
  $\widehat{\pi}_0 = min\{\widehat{\pi}_0(\lambda) | MSE(\lambda) \leq min_{\lambda'} MSE(\lambda')\}$
  $\widehat{\pi}_0 = min\{\widehat{\pi}_0, 1\}$

In Algorithm 1, first estimates are computed for original dataset. Second, for B bootstrap samples, the p-values of the samples are computed. Minimum value of λ giving the smallest MSE is selected as the best value. This is reasonable because it gives best balance between bias and variance.

Curve Fitting Method

The curve fitting method is based on fitting the cubic splines to $(\lambda, \widehat{\pi}_0(\lambda))$ values computed for $\lambda = 0, 0.01, 0.02, \cdots, 0.95$. The best value for λ is expected near 1 ($\lambda \geq 0.95$) because almost all genes are null in that region. However, the number

of p-values is small in that region because the region contains small number of p-values. Changing λ by small amounts (say 0.01) causes instability again due to the small number of p-values. This is why curve fitting is essential to estimate the best λ.



(a) without weighting (b) weighting $(\lambda, \widehat{\pi}_0(\lambda))$ by $1 - \lambda$

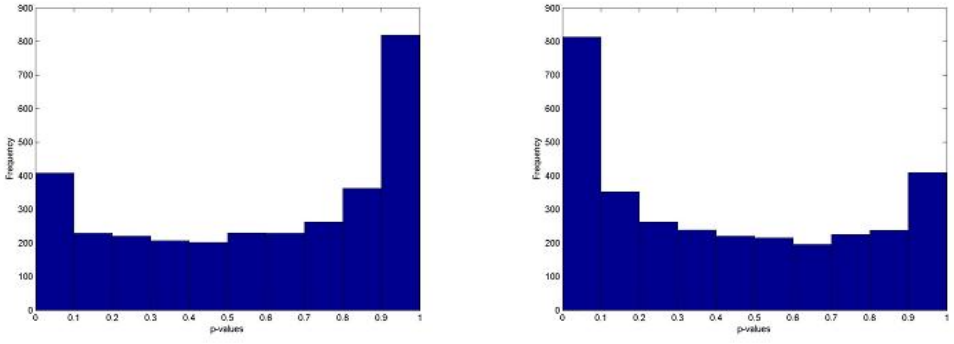
Figure 3.3: Estimation of $\widehat{\pi}_0(\lambda)$ using cubic splines

So, the main use of curve fitting is to prevent instability because $\widehat{\pi}_0(\lambda)$ becomes unstable when λ gets closer to 1, (see Figure 3.3). Let f be cubic splines fit, then the natural estimate of $\widehat{\pi}_0(\lambda)$ is obtained at $f(\lambda = 1)$, the value of spline at $\lambda = 1$. There are two methods that can be used to fit points either by weighting or without weighting. In the case of weighting, the weights of $(1 - \lambda)$ is suggested [46], where smaller weights are given to large values of λ because of instability. The estimations and their cubic splines fits are given in Figure 3.3.

From Figure 3.3, it is clear that $f(1)$ for without weighting case is larger than $f(1)$ for weighting by $1 - \lambda$ case. For this reason (underestimation is never desired) and from my experience, I suggest following the no weighting case because cubic splines already remove the instability and the values of $\widehat{\pi}_0(\lambda)$ are more reliable when λ is very close to 1.

3.4.2 Estimating $Prob(not\ up)$ and $Prob(not\ down)$

Storey estimator estimates $Prob(not\ de)$, which is smaller than both $Prob(not\ up)$ and $Prob(not\ down)$. So, it is not possible to substitute $Prob(not\ de)$ to be used in PaGE. Therefore, estimates for both of these probabilities should be found directly and separately.



(a) up-regulation (b) down-regulation

Figure 3.4: Histogram of p-values computed from 250 random permutations

The null hypothesis now is either genes are not up-regulated, or genes are not down-regulated. And, the alternative hypothesis is one-sided. Under the null hypothesis assumption, I computed the p-values of genes from BRCA dataset and present their histograms in Figure 3.4a and Figure 3.4b for up-regulation and down-regulation cases, respectively.

Based on the techniques for estimating $\widehat{\pi}_0(\lambda)$, I developed similar techniques for estimating $Prob(not\ up)$ and $Prob(not\ down)$. These quantities are denoted with $\widehat{\pi}_0^+(\lambda)$ and $\widehat{\pi}_0^-(\lambda)$, respectively.

Controlling the Directional Error

In a previous work, Abul *et al* [3], we have proposed to use the Storey estimator for estimating $Prob(not\ up)$ and $Prob(not\ down)$ using the estimate of $Prob(not\ de)$. The method employed the use of the q-value approach developed by Storey. We just find the ratio of up-regulated genes in likely differentially expressed genes (down-regulation case is similar). In its simplest settings, the method controls the directional error and is given in Algorithm 2.

Although our previous approach is meaningful, it lacks the power of direct estimation. This study extend that intuition to directly estimate this quantity using analytical methods similar to the estimation of $Prob(not\ de)$ in Storey estimator.

Algorithm 2 Estimation of $Prob(not\ up)$ and $Prob(not\ down)$

Require: Estimate $\widehat{\pi}_0$ computed (e.g. by the Algorithm 1), $M_{(-|-)}$ given
Ensure: Estimates for $Prob(not\ up)$, $Prob(not\ down)$ computed

```
M_(-|1) = (1 − π̂_0) * M_(-|-)
Let p_(1) ≤ p_(2) ≤ ··· ≤ p_(M_(-|-))
UP ← 0, DOWN ← 0
for i = 1 to M_(-|1) do
  if x̄_(i),t ≥ x̄_(i),c then
    UP++
  else
    DOWN++
  end if
end for
Prob(not up) = 1 − UP / M_(-|-)
Prob(not down) = 1 − DOWN / M_(-|-)
```

Using Bootstrapping

The bootstrap method can be directly used here to estimate $Prob(not\ up)$ and $Prob(not\ down)$. But, the p-values should be computed one-sided as given in Figure 3.4. Consider the up-regulation case (down-regulation case is similar), most of the p-values close to 0 belong to up-regulated genes, and most of the p-values close to 1 belong to down-regulated genes. As long as there are down-regulated genes, it is the case that $\lim_{\lambda \to 1} \widehat{\pi}_0(\lambda) > 1$ (check Figure 3.5a). This makes no sense because it is impossible to have $\pi_0 > 1$. So, I find an upper bound for λ based on which I consider an optimum value for λ. Let this upper bound be λ^{max}; λ^{max} is found by the maximum value for which $\widehat{\pi}_0(\lambda) \leq 1$:

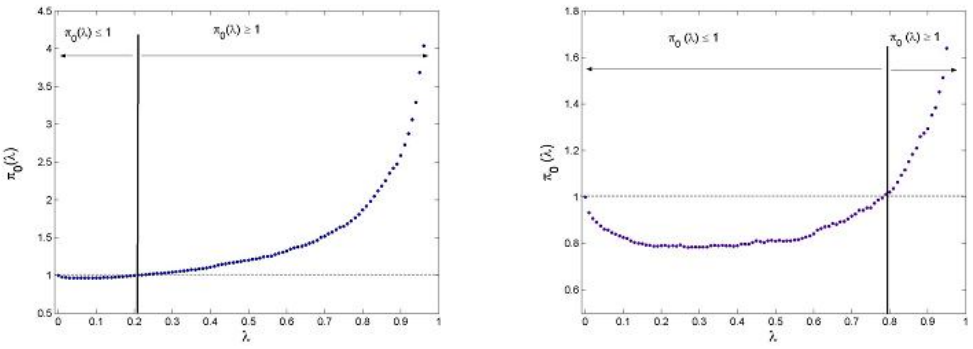
$$\lambda^{max} = max\{\lambda | \lambda \in [0..1], \widehat{\pi}_0(\lambda) \leq 1\} \tag{3.13}$$

The proportion of up-regulated genes can be computed using Algorithm 1 with the restriction that the values for λ are searched from the region $[0..\lambda^{max}]$. The output of Algorithm 1 is the estimation of $Prob(not\ up)$; the down-regulation case is similar.

Using Curve Fitting

Similar to Figure 3.3, I plot $(\lambda, \widehat{\pi}_0(\lambda))$ with their cubic splines fit; Figure 3.5a for up-regulation and Figure 3.5b for down-regulation.

As it is clear from Figure 3.5a and Figure 3.5b, the ratios in both cases become larger than 1 when λ gets close to 1. This is not surprising because the distribution



(a) up-regulation (b) down-regulation

Figure 3.5: Behavior of $\widehat{\pi}_0(\lambda)$

of p-values is bimodal; and when λ gets closer to 1, the average number of genes in the region $[\lambda..1]$ is much larger than the average number of genes in the region $[0..\lambda]$.

Let f be the cubic splines fit to the plotted data, if we estimate the proportion of null genes at $f(\lambda = 1)$, this will not make sense because it is larger than 1. So, we can not use the cubic splines fit method proposed by Storey directly.

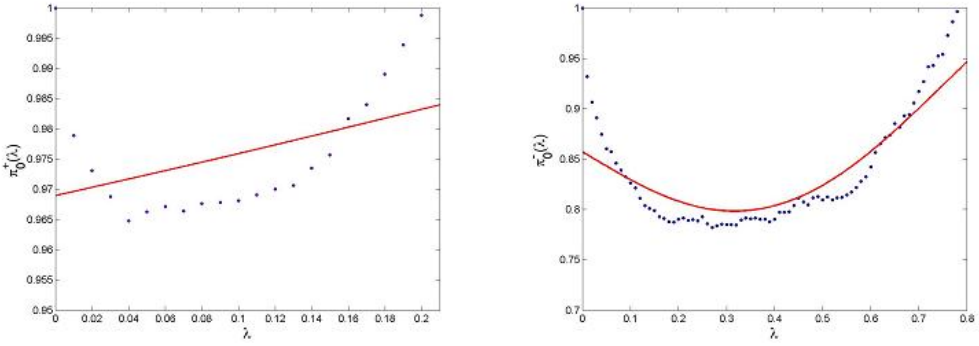
From the p-values histograms of up-regulation and down-regulation, it can be easily observed that in a region $[\alpha..\beta]$, $(\beta > \alpha)$, the p-values show uniform distribution. From Figure 3.4a, it can also be noted that p-values have uniform distribution within the vicinity of λ^{max} (for this example $\lambda^{max} = 0.21$).

Using this observation, I hypothesize that the ratio of null genes can be found only by considering the value pairs $(\lambda, \widehat{\pi_0}(\lambda))$, where $\lambda \in [0..\lambda^{max}]$ for cubic splines fitting. This is because value pairs for $\lambda \in [\lambda^{max}..1]$ are outliers.

The cubic spline fits for BRCA dataset is shown in Figure 3.6. Here, λ^{max} is 0.21 for up-regulation and 0.79 for down-regulation. There is no instability problem here when λ gets closer to λ^{max}. Finally, I estimate the proportion of $Prob(not\ up)$ by the minimum value of cubic spline fits.

Experiments

This section is dedicated to experimentally assess the accuracy of the developed methods. Reported here are the result of some experiments conducted on artificial datasets. The reason for using artificial datasets is to compare true values to



(a) $\widehat{\pi}_0^+(\lambda)$ (b) $\widehat{\pi}_0^-(\lambda)$

Figure 3.6: Estimation of $\widehat{\pi}_0^+(\lambda)$ and $\widehat{\pi}_0^-(\lambda)$

estimated values.

Let π_0 be the rate of non-differentially expressed genes, and π_1^+ (π_1^-) be the rate of up-regulated (down-regulated) genes, where $\pi_0 + \pi_1^+ + \pi_1^- = 1$. The artificial datasets for the treatment condition are generated from the respective mixing of distributions of F_0, F_1^+ and F_1^-; while for the control condition the dataset is generated from F_0 independently across genes. So, the datasets are created using the following setting:

$$X_{g,t} \sim \pi_0 \cdot F_0 + \pi_1^+ \cdot F_1^+ + \pi_1^- \cdot F_1^-$$
$$X_{g,c} \sim F_0$$

where $Prob(not\ up) = 1 - \pi_1^+ = \pi_0^+$ and $Prob(not\ down) = 1 - \pi_1^- = \pi_0^-$.

Basic assumptions and parameters used in this study are: rates of the mixture vary for fixed number of genes and distributions; let $M_{(-|-)}$ be 4000, F_0 be $N(6,1)$, F_1^+ be $N(9,1)$ and F_1^- be $N(3,1)$. Also, the number of replicas is fixed to 7 for control samples and to 8 for treatment samples. The p-values are calculated over 250 random permutations (full null model) of treatment and control conditions. Since the number of genes is huge (4000), the conducted experiments use only one sample dataset from each settings of mixing rates. Experiments are done with 5 datasets with the mixing rates given in Table 3.3.

The estimates of $Prob(not\ up)$ and $Prob(not\ down)$ are given in Table 3.4, from which it is evident that both methods correctly estimate the true values. For only 1

Table 3.3: Mixture rates used in Artificial Datasets

Dataset #	π_0	π_1^+	π_1^-
1	0.50	0.25	0.25
2	0.60	0.30	0.10
3	0.70	0.10	0.20
4	0.80	0.15	0.05
5	0.90	0.07	0.03

case (dataset 5), bootstrap method underestimated the true value. The cubic splines method seems to be more conservative as it never underestimated the true values; yet for some datasets it overestimates the true values. But the overestimation rate is only 1%. Since underestimation is not desired, results of cubic splines are more reliable.

Table 3.4: True values and Estimates of $Prob(not\ up)$ and $Prob(not\ down)$

	True		Bootstrap		Cubic Splines	
Dataset	π_0^+	π_0^-	$\widehat{\pi}_0^+$	$\widehat{\pi}_0^-$	$\widehat{\pi}_0^+$	$\widehat{\pi}_0^-$
1	0.75	0.75	0.75	0.75	0.76	0.76
2	0.70	0.90	0.70	0.90	0.71	0.90
3	0.90	0.80	0.90	0.80	0.90	0.80
4	0.85	0.95	0.85	0.95	0.85	0.96
5	0.93	0.97	*0.91*	0.97	0.93	0.97
BRCA	-	-	0.96	0.79	0.97	0.80

Also given in Table 3.4 are estimated values for the BRCA dataset (the same dataset used in [46]), which has 7 replicas for BRCA1 type tumors and 8 replicas for BRCA2 type tumor for 3170 genes. The BRCA1 tumor type is used as the control condition and BRCA2 as the treatment condition.

The effect of estimating $Prob(not\ up)$ and $Prob(not\ down)$ are shown for confidence levels [0.5..0.95] in Table 3.5. The results clearly show increased power of estimation over not estimating prior probabilities.

3.4.3 Computing q-values for Up-regulation and Down-Regulation

In cases for which the direction of the error is pre-specified, finding confidence in only up-regulated or down-regulated genes instead of differentially expressed genes

Table 3.5: Improvement of power for BRCA dataset

	$\widehat{\pi}_0^+ = 1, \quad \widehat{\pi}_0^- = 1$			$\widehat{\pi}_0^+ = .97 \quad , \widehat{\pi}_0^- = .80$		
Confidence	up	down	up+down	up	down	up+down
0.50	164	664	828	175	1108	1283
0.55	134	518	652	138	884	1022
0.60	129	358	487	131	636	767
0.65	80	178	258	87	443	530
0.70	49	119	168	55	248	303
0.75	43	13	56	43	142	185
0.80	19	3	22	19	13	42
0.85	17	3	20	19	3	22
0.90	0	2	2	0	2	2
0.95	0	0	0	0	0	0

is more accurate. In other words, specification of more specific information should not be neglected but taken into account. How this kind of information can be incorporated is shown in this section.

In the previous section, it is shown how to compute the ratio of up and down genes separately. So, by incorporating these estimations into the computation of FDR (Formula 3.4), the q-value associated to each gene can be computed. The resulting q-value for any gene g gives the measure about the confidence in up-regulation (or down-regulation) of gene g.

To compute the q-values for genes, it is enough to know the null distribution and the prior probabilities ($Prob(not\ up)$ and ($Prob(not\ down)$). The latter can be computed using the presented curve fitting method and the former can be computed using the one-sided p-value computation.

The interpretation of the q-value changes as well. For up-regulation case and for gene g, it can be interpreted as follows: average proportion of up-regulated genes in all genes having q-value larger than or equal to that of g. So, it gives only the proportion of up-genes (i.e., not proportion of up and down genes together). In other words, it contains much information by specializing the region of interest. The down-regulation case is similar.

3.5 Simultaneously Finding Up and Down Cutoffs

Recall that PaGE separately and independently finds up and down cutoff ratios. Here, I cast the problem to find up and down cutoff ratios simultaneously and

dependently. Under this formulation, the null hypothesis becomes: *any gene g is not differentially expressed*, i.e., their means are the same in both conditions t and c. If the used statistics is the ratio of mean of samples, $\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}}$, then FPR for a selected cr and Cr pair is:

$$FPR = Prob\Big((\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}} > Cr) \ or \ (\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}} < cr)|\frac{\mu_{g,t}}{\mu_{g,c}} = 1\Big) \tag{3.14}$$

Since the events are disjoint (because Cr is always larger than 1 and cr is always less than 1), FPR can be rewritten as follows;

$$FPR = Prob\Big(\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}} > Cr|\frac{\mu_{g,t}}{\mu_{g,c}} = 1\Big) + Prob\Big(\frac{\overline{X}_{g,t}}{\overline{X}_{g,c}} < cr|\frac{\mu_{g,t}}{\mu_{g,c}} = 1\Big) \tag{3.15}$$

Using similar arguments while rewriting Formula 3.8 from Formulas 3.6 and 3.7, Formula 3.15 can be rewritten as follows;

$$Prob\Big(\frac{\frac{\overline{X}_{g,t}}{\mu_{g,t}}}{\frac{\overline{X}_{g,c}}{\mu_{g,c}}} > Cr\Big) + Prob\Big(\frac{\frac{\overline{X}_{g,t}}{\mu_{g,t}}}{\frac{\overline{X}_{g,c}}{\mu_{g,c}}} < cr\Big) \tag{3.16}$$

It is already given that the distribution of each of the components in Formula 3.16 follows the distribution given in Formula 3.9. So, given the distribution of the null hypothesis that genes are not differentially expressed and given cr and Cr, FPR can be easily computed. Note that this exactly computes FPR, whereas in Formula 3.8 it is underestimated for the sake of being conservative. So, this is a possible increase in the power.

The confidence is computed using the following formula;

$$\begin{aligned} Confidence = & \quad 1 - Prob(not\ de|pred\ de) \\ = & \quad 1 - \frac{Prob(not\ de)Prob(pred\ de|not\ de)}{Prob(pred\ de)} \end{aligned} \tag{3.17}$$

Using Storey estimator, after substituting FPR for $Prob(pred\ de|not\ de)$, the estimated confidence can be written as follows:

$$Confidence = 1 - \frac{\widehat{\pi}_0 \cdot FPR}{Prob(pred\ de)} \tag{3.18}$$

Let r^{max} and r^{min} be the maximum and minimum average expression ratios

over all genes, respectively. The up-regulation (down-regulation) cutoff clearly lies in the region $(1..r^{max}]$ ($[r^{min}..1)$). Without loss of information, we can discretize these rejection regions. This is because discretization level is arbitrary; it can be increased if it is not sufficient. Let $Numbins$ be the number of discretizations then, the rejection region corresponding to indexes (i, j) is;

$$\begin{aligned} Cr_i &= 1 + \frac{(r^{max} - 1) \cdot i}{Numbins}, \forall i \in \{1, 2, \ldots, Numbins\} \\ cr_j &= 1 - \frac{(1 - r^{min}) \cdot j}{Numbins}, \forall j \in \{1, 2, \ldots, Numbins\} \end{aligned} \tag{3.19}$$

So, rejection regions (i, j) and (Cr_i, cr_j) can be used interchangeably.

Since our rejection regions are two-sided, (Cr, cr) pairs should be considered together. Given the null distribution and Type I error rate α, there are usually multiple values for (Cr, cr) satisfying the given α. But, if we consider only symmetric rejection regions ($Cr = 1/cr$), which is mostly employed because of ease, then (Cr, cr) pair is unique. Since using symmetric rejection regions introduces selection bias, I avoid it and consider all rejection regions with Type I error rate α. By adopting this, the problem becomes more exploratory (analogues to FDR), since in case multiple values satisfy the desired quantity, the best is to present all to the user and let him/her choose how to use it. This is important in microarray domain, since biologists always have some prior information to utilize while choosing from the presented solution set. Particularly, they may be interested only in the number of up-regulated genes, in the number of differentially expressed genes, etc.

It is possible to represent confidences attached to all pairs (Cr_i, cr_j) using $Numbins \times Numbins$ matrix $Conf$. Let entry (i, j) be $Conf(Cr_i, cr_j)$, e.g., the confidence attached to pair (i, j) (and hence (Cr_i, cr_j)). The reasonable assumption for the nested set of rejection regions is the following;

$Conf(Cr_i, cr_j) \geq Conf(Cr_{i'}, cr_{j'})$, if $i' \leq i$ and $j' \leq j$.

This is because $\Gamma_{(Cr_i, cr_j)} \subseteq \Gamma_{(Cr_{i'}, cr_{j'})}$ (i.e., larger evidence means larger confidence).

The confidence for cutoffs (Cr_i, cr_j) is first computed with Formula 3.18 (let it be $Conf_{cr_j}^{Cr_i}$) and then updated as follows (this is because of nested rejection regions);

$$Conf(Cr_i, cr_j) = max\Big\{Conf^{Cr_i}_{cr_j}, max\big\{Conf(Cr_{i'}, cr_{j'}) \mid i' \in \{1,\ldots,i\}, j' \in \{1,\ldots,j\}, (i \neq i' \text{ and } j \neq j')\big\}\Big\} \tag{3.20}$$

At first glance, it seems prohibitive to compute the confidence for any cutoff pair as it requires all of its subsets computed. But, as it is articulated in Theorem 3.1, it can be computed efficiently.

Theorem 3.1. *Assuming that $Conf(Cr_{i-1}, cr_j)$ and $Conf(Cr_i, cr_{j-1})$ are computed as given in the Formula 3.20, no other value (other than $Conf^{Cr_i}_{cr_j}$) is required to compute $Conf(Cr_i, cr_j), \forall i, j > 1$.*

Proof : First of all, the need for the computation of $Conf^{Cr_i}_{cr_j}$ is obvious. Consider all legal pairs (i', j') of the inner max operator in Formula 3.20. Clearly, all such pairs either fall in $i' \in \{1, 2, \ldots, i-1\}, j' \in \{1, 2, \ldots, j\}$ or $i' \in \{1, 2, \ldots, i\}, j' \in \{1, 2, \ldots, j-1\}$, or both. By Formula 3.20, the maximum confidence for the former set is (by definition) $Conf(Cr_{i-1}, cr_j)$, and the maximum confidence for the latter set is (by definition) $Conf(Cr_i, cr_{j-1})$. Then, Formula 3.20 simplifies to $Conf(Cr_i, cr_j) = max\{Conf^j_i, Conf(Cr_i, cr_{j-1}), Conf(Cr_{i-1}, cr_j)\}$; hence, completing the proof. □

Presented in different word, it is sufficient to solve the following *dynamic programming* problem (see Section 5.2.2 for a concise introduction to dynamic programming);

$$Conf(Cr_i, cr_j) = \begin{cases} 0, & i = 1, j = 1; \\ max\{Conf^{Cr_i}_{cr_j}, Conf(Cr_i, cr_{j-1})\}, & i = 1, j \neq 1; \\ max\{Conf^{Cr_i}_{cr_j}, Conf(Cr_{i-1}, cr_j)\}, & i \neq 1, j = 1; \\ max\{Conf^{Cr_i}_{cr_j}, Conf(Cr_{i-1}, cr_j), Conf(Cr_i, cr_{j-1})\}, & \text{otherwise.} \end{cases} \tag{3.21}$$

For a given minimum confidence threshold, $Conf^{min}$ provided by the user, the cutoff pairs having confidence just above $Conf^{min}$ should be reported for not losing much power. Also, note that if $Conf(Cr_i, cr_j) \geq Conf^{min}$, then there is no need to consider rejection regions $\Gamma_{(Cr_{i'}, cr_{j'})}$ such that $\Gamma_{(Cr_i, cr_j)} \subseteq \Gamma_{(Cr_{i'}, cr_{j'})}$, (i.e., $i' \geq i$

and $j' \geq j$). In other words, $\Gamma_{(Cr_{i'}, cr_{j'})}$ is more specific than required.

All put together, Algorithm 3 finds all simultaneous cutoffs having confidence just above the given minimum confidence level $Conf^{min}$.

Algorithm 3 Simultaneously finding up and down cut-off ratios

Require: Intensities for all genes for two conditions, $M_{(-|-)}$, $Numbins$, $Conf^{min}$
Ensure: All maximal up-down cutoffs satisfying minimum confidence $Conf^{min}$

```
Compute average intensities x̄_{g,t} and x̄_{g,c} for every g
r_g ← x̄_{g,t} / x̄_{g,c}  {r_g = ratio of gene g}
r^max ← max{r_g | g ∈ [1..M_{(-|-)}]}
r^min ← min{r_g | g ∈ [1..M_{(-|-)}]}
Compute π̂_0 using cubic splines fit method
Compute null distribution using the Formula 3.9
Let Conf be Numbins × Numbins matrix {Default value for Numbins is 100}
for i = 1 to Numbins do
  for j = 1 to Numbins do
    Compute Cr_i using the Formula 3.19 with parameter i
    Compute cr_j using the Formula 3.19 with parameter j
    Conf_{cr_j}^{Cr_i} = 1 − π̂_0·FPR(Cr_i, cr_j) / Prob_{Pred de}(Cr_i, cr_j) , if (i ≠ 1, j ≠ 1)
    Compute Conf(Cr_i, cr_j) using the Formula 3.21
  end for
end for
limit ← Numbins
for i = 1 to Numbins do
  for j = 1 to limit do
    Compute Cr_i using the Formula 3.19 with parameter i
    Compute cr_j using the Formula 3.19 with parameter j
    if Conf(Cr_i, cr_j) ≥ Conf^min then
      limit ← j − 1
      Output (Cr_i, cr_j)
    end if
  end for
end for
```

3.5.1 Experiments

For the BRCA dataset, the sample output of the code presented in Algorithm 3 is given in Table 3.6.

From Table 3.6, it can be easily seen that no two solutions are nested, as guaranteed by the algorithm. It can be also noted that confidences of the solutions are just above the minimum confidence (the difference is less than 1%). If one is forced to select a solution among a set of solutions, there are lots of alternatives such as: maximizing total number of up and down regulated genes, maximum number of

Table 3.6: BRCA dataset cutoffs for .65 and .85 confidences

	$Conf^{min} = .65$				
Solution #	Cr	cr	Conf(Cr, cr)	up-genes	down-genes
1	1.63	0.67	0.652	176	383
2	1.67	0.68	0.657	162	441
3	1.77	0.71	0.658	133	532
4	1.80	0.73	0.650	125	594
	$Conf^{min} = .85$				
Solution #	Cr	cr	Conf(Cr, cr)	up-genes	down-genes
1	2.77	0.22	0.851	21	2
2	2.80	0.27	0.852	20	3
3	2.84	0.30	0.850	20	4

up-genes, solution pair having closest confidence to minimum confidence, the most symmetric solution, etc. From this consideration, using symmetric regions is just a special case of the method proposed in this thesis. Finally, it is worth noting that given these criteria the selection of a solution from the solution set can be easily automated.

Instead of selecting one of the solutions, one can get a more conservative solution obtained by considering the least value of cr and the greatest value of Cr among all solutions. For example, this kind of solution for .65 confidence is $(1.80, 0.67)$, and it is $(2.84, 0.22)$ for .85 confidence. Of course, in this case the confidence level increases and the number of differentially expressed genes decreases. For a cutoff of $(1.80, 0.67)$, the total number of differentially expressed genes is $383 + 125 = 508$, whereas the mean value of all solutions is 636.

Table 3.7: Effect of $Prob(not\ de) = 1$ vs. $Prob(not\ de) = \widehat{\pi}_0$

$Conf^{min}$	$Prob(not\ de) = 1$	$Prob(not\ de) = \widehat{\pi}_0 = .76$
0.50	717	1340
0.55	469	1011
0.60	339	745
0.65	237	561
0.70	142	320
0.75	49	193
0.80	30	60
0.85	22	29
0.90	0	4
0.95	0	0

Table 3.7 shows the effect of $Prob(not\ de) = 1$ vs. $Prob(not\ de) = \widehat{\pi}_0$ for confidence levels 0.50 to 0.95 for BRCA dataset. From Table 3.7, it can be concluded that the estimation of $Prob(not\ de)$ increases the power of the method within the same confidence level for all the experimented minimum confidences. The increase is significant (note the about two-fold increase).

3.6 Discussion

PaGE is a method based on ratio statistics to find differentially expressed genes across the two given conditions. It outputs all differentially expressed genes having confidence larger than a user supplied minimum confidence threshold. The method is very useful in the sense that it is exploratory; this is because minimum confidence is supplied by the user. I identified the drawback of the method as underestimating the confidence in the differential expression of genes. That is, if a gene g's true confidence is say 87%, due to underestimation, it can assign a value less than this (say 78%) to this gene. If the user supplied minimum confidence is 80%, then gene g will not be called differentially expressed. This clearly results in a reduced power for the sake of being more conservative than required.

I tackle the problem of avoiding underestimating true confidences as much as possible, and at the same time never overestimating them. True confidences can be computed given the prior probabilities $Prob(not\ up)$ and $Prob(not\ down)$, but they are unavailable. I have developed two methods to estimate these probabilities. The first method uses a bootstrapping method while the second one uses cubic splines fitting. For artificially generated datasets, both methods are demonstrated to be accurate. The increase of power is shown on BRCA dataset (a real gene expression dataset) using the estimates produced by the proposed estimation methods. Although, the increase depends on the distribution of p-values and the minimum confidence value specified, more than two-fold increases are observed for some minimum confidences (see Table 3.7).

The PaGE is one-sided and finds up and down cutoffs independently. In this case, the problem turns to be very easy since, all the rejection regions are nested. I extend the problem definition of PaGE to find all up and down cutoffs simultaneously and dependently.Also, I show that, the null distribution for this case can be computed in a way similar to PaGE. But, for non-symmetric rejection regions, the solution set is not unique. For a given minimum confidence level, I develop an algorithm that finds all pairs of cutoffs satisfying this using dynamic programming methods. The nice

property is the ability to incorporate Storey estimator to this algorithm directly to increase the power. I show that this incorporation increases power significantly on the BRCA dataset.

PaGE uses only the average ratio statistics for identifying differentially expressed genes, but other statistics (e.g., t-statistic) can be used as well with the presented algorithms. But the main advantage of the ratio statistic is its ease of interpretation, even though it has some drawbacks (e.g., the two ratios 1/5 and 20/100 are the same, but the difference between the nominator and denominator is 4 and 80, respectively). But for normalized datasets, it is considered that ratio statistic is useful and suitable for extrapolatory data analysis.

CHAPTER 4

DISCRETE GENETIC REGULATORY NETWORK MODELING

4.1 Introduction

A cell contains thousands of genes with their associated products, i.e. proteins. The machinery in the genetic material determines which/when genes are going to be expressed and their abundance. Genetic regulatory network modeling studies uncovering code of this machinery. So, the uncovered models are abstractions of this machinery. Since gene expression data is a snapshot of what this machinery does, it helps to uncover its code. Note that this is a kind of reverse engineering. Lots of snapshots (samples) are usually required to reach meaningful models. The models for various cases are possible, e.g., model under normal conditions, model under a specific stress, dynamic model from temporal data, model for genetic mutations, and others.

The objective in the modeling is to get insights in the cell machinery. If almost correct model is induced, then computational methods can be applied for a number of goals like, simulating, predicting, controlling, and treatment.

Since the exact mechanism is not known, several modeling approaches have been proposed for identifying gene regulation network from datasets of microarray experiments. So the main theme of this chapter is covering discrete genetic regulatory network induction methods. The methods discussed are, Boolean Networks, Probabilistic Boolean Networks, Bayesian Networks, and Markov Chains Model. The learning, inference and dynamics properties of these methods are considered. Some other approaches are also mentioned (differential equations, linear models, genetic algorithm methods, etc). In the context of the Boolean networks, the data requirements problem for non-random time series data is analyzed. Also, on a given model

the problem of finding internal effects of external interventions are addressed. How the differential gene expression analysis can be used for this purpose is shown.

Boolean networks and Probabilistic Boolean networks are discrete; and Bayesian networks can be either discrete or continuous. Modeling continuous gene expression levels with discrete models makes sense because human reasoning is most of the time qualitative. The inference of discrete models from gene expression data is not trivial. Reasons for such difficulty include:

- High dimension,
- Continuous valued expression data,
- Lack of large number of experiments,
- Coarse and uneven sampling,

Dynamics of discrete networks can be explored within the discrete Markov Chains framework. That is, their dynamics can be collapsed into Markov chains. This is particularly important for this study as dynamic control of discrete networks is addressed in the next chapter.

4.2 Markov Chains

Markov Chains (MCs) are general tools to model the dynamic properties of a process having finite or infinite state-space [78, 90]; they define how the process (deterministic or stochastic) evolves. MCs have the property (known as Markovian property) that evolution of the process depends only on the past k states, where the value of k is known as the order of the model. If this value is 1, it is said that MC is first-order and has the nice property that the next state depends only on the current state. The generality of first-order MCs originate from their capability to represent any order MCs. This is done by just increasing the state space [90].

Let $\mathcal{S}$ be the finite state space of process, and consider a particular state s. Usually, states form vector of components, $s =< s_1, s_2, \ldots, s_n >$, where n is the dimension of the state space. State to state transition probabilities are defined by the function $T : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$. The stationary transition probability from state s to state s' is $T_{ss'} = Prob\{S(t+1) = s' \mid S(t) = s\},\ \forall t \in \mathcal{N}$, where $S(t)$ and $S(t+1)$ are random variables representing the state at time step t and $t+1$, respectively. Equivalently, the transition probabilities can be defined as a matrix where states are indexed. If the index of state s is i and s' is j, then entry (i, j) represents the value

of $T_{ss'}$. The constraint here is that $\sum_{s' \in \mathcal{S}} T_{ss'} = 1$, $\forall s \in \mathcal{S}$; this kind of matrices is known as *stochastic matrices*, where every row sums up to 1.

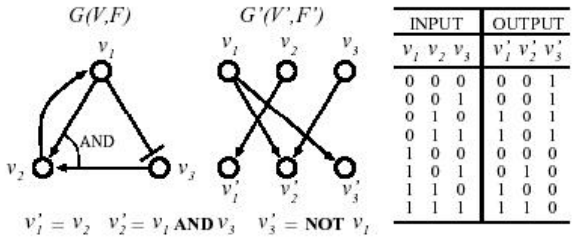
Given any initial state distribution, the unconditional probability of states at any time can be obtained. Assume that the initial distribution given (at $t = 0$) is a_0, the probabilities at time t is $a_t = a_0 \cdot T^t$ where T^t is the t'th power of T. A state s is called *absorbing* if $T_{ss} = 1$. Likewise states can be attributed as *recurrent, transient, periodic, ergodic*, etc (for a concise introduction see [90]). Among them, particularly important is the Markov chain being ergodic. If a Markov chain is ergodic, as $t \to \infty$ the initial state distribution becomes immaterial, i.e., the same steady-state distribution exists.

4.3 Boolean Networks

Boolean (Genetic) Networks (BNs) were originally introduced by Kauffman [75], and extensively studied later [7, 8, 9, 61]. In this approach, expression levels of genes are represented as boolean variables. For any gene, the expression level is 1 when the gene is expressed (activated), and 0 when the gene is repressed (not-activated). Each gene is assigned a boolean variable which is a boolean function of values of some other variables (genes). A particular state of the network is an instantiation of all of its variables. So, given n genes, the state space contains 2^n states. Since state transitions happen at discrete times and the state space is finite, BNs are sometimes identified as *discrete BNs*. Binary discretization of continuous expression values are pre-requisite to model gene networks using boolean values.

Definition 4.1 (Boolean Network). A boolean network (BN) is a tuple $G =< V, F >$, with a set of nodes V and a set of boolean functions $F = \{f_v | v \in V\}$. Function f_v specifies gene regulation rule for node v. Gene v is expressed (repressed, resp.) if f_v computes to 1 (0, resp.). The function f_v for v is a deterministic boolean function of a subset of V with k elements, i.e., $f_v(v_{i_1}, \ldots, v_{i_k})$; so the value of the k-subset completely determines the state of gene v; the genes not in the k-subset are either irrelevant or indirectly relevant to v. □

To define the dynamics of BNs, let the state of nodes at time step t be ψ_t; if gene v is expressed at time t, then $\psi_t(v) = 1$, otherwise $\psi_t(v) = 0$. The state of the nodes at time t+1 is defined by ψ_{t+1} and this value is determined based solely on ψ_t of the k-subset: $\psi_{t+1}(v) = f_v(\psi_t(v_1), \ldots, \psi_t(v_k))$. The state of the network is defined as a vector s and its value at time t is given by $s(t) =< \psi_t(v_1), \psi_t(v_2), \ldots \psi_t(v_n) >$, where $n = |V|$.



INPUT			OUTPUT		
v_1	v_2	v_3	v_1'	v_2'	v_3'
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	0

Figure 4.1: A Boolean network and its wiring diagram (from Akutsu *et al* [8])

The operations of BNs are usually represented by diagrams for ease of understanding. The wiring diagram of $G = (V, F)$ is another graph $G' = (V', F')$ showing inputs and outputs with different nodes. In this diagram, input nodes are genes at the current time step t and output nodes are genes at the next time step $t + 1$. In the wiring diagram G' of G, nodes are replicated, i.e., $V' = \{v_1, \cdots, v_n, v_1', \cdots, v_n'\}$. The nodes $\{v_1, \cdots, v_n\}$ are input nodes, while the nodes $\{v_1', \cdots, v_n'\}$ are output nodes. The function set F' is essentially the same as F. The only difference is renaming of the input and output variables. A sample boolean network and its wiring diagram is shown in Figure 4.1.

It is trivial to simulate any BN given an initial state $s(0)$; just apply $f_v, \forall v \in V$. For any initial state, since the state space is finite and the transitions are deterministic, after some time T' the simulation starts looping with period T, namely $s(T') = s(T'+T), s(T'+1) = s(T'+T+1)$, and so on. It is clear that both $T', T \leq 2^n$. The cycle $s(T'), s(T'+1), \ldots, s(T'+T)$ is called an *attractor* of the respective BN. If the length of this cycle is 1 (self-loop), it is called *point attractor*. A BN may have more than one attractor. Biologically, these attractors correspond to stable states of the cell, since non-attractor states are transient. It is evident that without any external perturbation, there is no exit from any of the attractors. So, it is clear that states of BN can be partitioned into equivalence classes based on their attractors.

Time series gene expression data is needed to infer dynamic BNs. After the proper pre-processing (feature selection, discretization, etc), the input dataset to the inference procedure is obtained in the following way. Let the dataset be D and its length be $M + 1$; D is indexed from $D(0), D(1), \ldots, D(M)$. The input to the inference procedure is input-output pairs of the form $< D(t), D(t+1) >, \forall t \in \{0, 1, \ldots, M-1\}$. By this way, one-step back dependency is assumed. Although

this is the usual way of casting experiments to input-output pairs for dynamic BNs, there are approaches (like the one described in [82]) that assume multi-step back dependency.

The *consistency* problem is defined as finding functions for every node of the network with a complete agreement to data. Closely related to the consistency problem is the *best-fit extension* problem. In the best-fit extension approach, the function with the smallest disagreement to the input-output pair is searched. In this sense, the consistency problem (in which disagreement is 0) is a special case of the best-fit extension [81, 58]. So, inference can be done under this approach as well.

In [7], the authors analyze the number of controlled experiments (instead of random) to be done for guaranteed identification of BNs. Their main result can be stated as follows: if the number of manipulated (intensionally set as either 0 or 1) genes are at most n, the problem can be solved with $O(logn)$ experiments. However, if the manipulated genes are restricted to at most d, the problem can be solved with $O(n^{2d})$ experiments; for details see [7].

Akutsu *et al* [8, 9, 7] studies the required number of observed input-output pairs (not controlled) needed for identifying and inferencing BNs. It has been shown that the number of experiments required for identification from uniformly random input-output pairs is $O(log(n))$; but with a considerable constant factor. Indeed $O(log(n))$ complexity has been first observed by Liang *et al* [61] in their simulations.

Akutsu *et al* [8] provides the number of experiments required for identifying BNs from uniformly sampled random data. Their main result is articulated next in Theorem 4.1.

Theorem 4.1 (Akutsu *et al.* [8]). *If $O(2^{2K} \cdot (2K+\alpha) \cdot log(n))$ INPUT expressions are given randomly, then the following holds with probability of at least $1 - \frac{1}{n^{\alpha}}$: there exists at most one Boolean network of n nodes with maximum in-degree $\leq K$, which is consistent with the given input/ouput pairs.*

Proof : Refer to [8] for the complete proof. □

The information theoretic lower bound is also provided as given in the following theorem.

Theorem 4.2 (Akutsu *et al* [8]). *$\Omega(2^K + Klogn)$ input/output pairs are necessary in the worst case to identify the Boolean network of maximum indegree $\leq K$.*

Proof : Noting that for each node $\Omega(n^K)$ possible combinations of input nodes and 2^{2^K} possible boolean functions per node, there are $\Omega((2^{2^K} \cdot n^K)^n)$ boolean networks whose maximum in-degree is at most K. Therefore, $\Omega(2^K n + n \cdot K \cdot log(n))$ bits

are required to identify a boolean network. Since the information obtained from a random single input/ouput pair is n bits, therefore $\Omega(2^K + K \cdot log(n))$ input/output pairs are necessary in the worst case. $\square$

The corollary to Theorems 4.1 and 4.2 is $O(log(n))$ input/output pairs are necessary and required to induce a random boolean network.

Both Theorems 4.1 and 4.2 apply for random boolean networks with in-degree of at most K. On the other hand, time series gene expression data is not random, but the next sample (at time $t+1$) is completely dependent over the current sample (at time t). The authors address this case as well and show that the number of required samples is still $O(log(n))$ from their simulations, i.e., this is not an analytic result. They obtained $O(log(n))$ complexity (but with a much larger constant factor of ≈ 20 times) with the simulation method given in Algorithm 4;

I observed from the results of their simulations of time series experiments that neither the number of attractors nor the required number of input/output pairs do grow logarithmically with the number of genes as they claim. Actually, required number of input/output pairs do depend on the number of attractors. For this reason, interpretation of simulation results may mislead. Instead of using simulations, getting an analytic lower bound for time series experiments motivated me.

In the following, it is proved (using information theoretic lower bounds) that the necessary number of experiments in this case is not $\Omega(log(n))$ but $\Omega(\sqrt{n} \cdot log(n))$.

Algorithm 4 Uniquely identifying a BN from time series simulation

Require: number of genes n, indegree K given
  $Network \leftarrow$ Randomly construct a random n, K, BN
  repeat
    select an initial state $s(0)$ randomly
    $s \leftarrow s(0)$
    repeat
      if Is the $Network$ uniquely identified **then**
        Return
      end if
      $s \leftarrow$ state of $Network$ following s
    until a cycle is detected
  until Forever

In Algorithm 4: for each $s(0)$, when the chain reaches to a state that is already visited, the strategy is terminating the simulation at that point and restarting from another random initial state. This is because there is no rationale to continue the simulation from an already visited state. This continues until the network is identified uniquely. Note that only the initial state $s(0)$ is selected randomly.

Basin of an attractor is defined as the states with ultimate transitions to an attractor. For boolean networks with in-degree of at most K, the number of states in an attractor grows with $O(\sqrt{n})$ on the average and the number of attractors grows with $O(\sqrt{n})$ on the average [75, 18]. So, these results clearly means that each cycle of Algorithm 4 is $O(\sqrt{n})$ (note that even a single movement leads to an attractor state and $O(\sqrt{n})$ movements are required to come back to that state).

Lemma 4.1. *The average number of states visited, starting from an initial state, before reaching a state that has already been observed is at least* $O(\sqrt{n})$

Proof : From any initial state $s(0)$, to reach a state that has already been visited requires the sequence,

1. visit states until an attractor state (say s) is visited
2. visit on average $O(\sqrt{n})$ states before returning back to s (see [75, 18])

this lemma proves even when the first step takes zero visits. □

Lemma 4.2. *The average number of information units obtained from m random experiments for a network with n nodes is n.*

Proof : Consider a network of n nodes, and let X be a random variable denoting the outcome from a random experiment. n bits are required to specify any outcome because X can have 2^n values. For m independent random experiments, the total number of information units obtained is nm. So, the average is n bits.□

Without loss of generality, any value that is in the order of $O(\sqrt{n})$ can be used to get an asymptotic result for total visits for Lemma 4.1. Both lengths in the lemma are assumed to be $\sqrt{n}$ in the following. So, a trial is totaling to $2\sqrt{n}$.

Lemma 4.3. *The average number of information units obtained from time series experiments for a network with n nodes and $2\sqrt{n}$ cycle length is* $\frac{n+(2\sqrt{n}-1)log2\sqrt{n}}{2\sqrt{n}}$.

Proof : Total information content can be calculated as the total information content of the first random experiment and the information content of $2\sqrt{n}-1$ dependent experiments. By Lemma 4.2, the first is n. Since each of the dependent experiments can be in one of $2\sqrt{n}$ states, the information content of one dependent experiment is $log(2\sqrt{n})$. Since there are $2\sqrt{n}-1$ dependent experiments, their total information value at most sums-up to $(2\sqrt{n}-1)log2\sqrt{n}$; so the average of dependent experiments and one random experiment is $\frac{n+(2\sqrt{n}-1)log2\sqrt{n}}{2\sqrt{n}}$. □

Theorem 4.3. $\Omega(logn)$ *trials each containing* $2\sqrt{n}$ *experiments are necessary for identifying boolean networks from time series data when cycle length is* $2\sqrt{n}$ *on the average.*

Proof : From Theorem 4.2, the required total number of bits that should be obtained is $\Omega(2^K n + n \cdot K \cdot log(n))$; and from Lemma 4.3, the total number of bits obtained from a trial is $n+(2\sqrt{n}-1)log(2\sqrt{n})$. By taking the ratio $\frac{2^K n+n\cdot K\cdot log(n)}{n+(2\sqrt{n}-1)log(2\sqrt{n})}$ and after simplifying, it can be easily found that $\Omega(log(n))$ trials are needed. □

Corollary 4.1. *The necessary number of experiments required for identifying BN from time series data is* $\Omega(\sqrt{n} \cdot log(n))$.

Proof : From Theorem 4.3, the number of trials found is $\Omega(log(n))$ with each having $2\sqrt{n}$ input/output pairs. So, the necessary number of total experiments is $\Omega(\sqrt{n} \cdot log(n))$. □

Corollary 4.1 counter-proves the claim of Akutsu *et al* [8], where their claim (from simulations) is that the required number of experiments is still $\Omega(log(n))$ for time series data as well.

Note that the approach for proving the lower bound for finding the necessary number of experiments is general. That is, for instance assuming the cycle length is $log(n)$, the required number of experiments becomes $\Omega(log(n) \cdot log(n))$.

In [8], the authors presented an exhaustive algorithm for finding consistent boolean functions (functions not violating input-output patterns) with the given input-output pairs under the constraint of bounded in-degree (number of input nodes). The algorithm considers all boolean functions for all K-subsets of V for any node v. For the maximum in-degree K, the number of boolean functions becomes at most 2^{2^K}, for any node v and for a specific K-subset of V. For n genes and $\binom{n}{K}$ combinations, the run time complexity of the exhaustive algorithm is clearly $O(2^{2^K} \cdot n^K \cdot n \cdot M)$. In [9], the authors presented an improved randomized algorithm, but the order of complexity is between $O(n^{K-1})$ and $O(n^K)$.

An information theoretic algorithm (known as **REVEAL**) is provided by Liang *et al* [61] for the inference of boolean networks from experimental data. The maximum in-degree is assumed to be bounded by a constant K; otherwise the problem is NP-Hard. The algorithm is iterative by first considering 1-subsets, then 2-subsets and finally K-subsets. If any combination of subsets explains the whole variability consistently, the respective subset is selected with corresponding function as determiners for any node v. The measure for variability is mutual information. They have observed that the probability of identifying incorrect solutions reduces exponentially with the number of input-output pairs.

Most of the time gene expression levels are quantized into ternary values (-1, 0 and 1), where 0 is used for baseline level, -1 for repression and 1 for over expression. General rules for boolean networks apply for ternary networks, and in general for n-ary networks.

4.4 Probabilistic Boolean Networks

The recently introduced PBNs [78] form an extension of the standard BNs. Boolean networks can compute deterministically the next state given the current state and only randomness in BNs is the starting state. However, this is not the case in the true genetic networks, where the transition is almost always stochastic; i.e., there may be more than one next state. PBN approach for this situation is allowing more than one deterministic boolean function (called predictors) for every node. The selection probability of a predictor determines whether it is going to be used. In this respect, BNs are a special case of PBNs, where the number of predictors is one with selection probability 1.

At any time, it is assumed that one of the predictors is selected for every gene; and the selected predictor function determines the next state. For simplification purposes, the probability of selecting predictor for a gene is assumed to be independent of predictors of the other genes. For any gene g, assume there are l_g number of boolean functions: $f_1^{(g)} \dots f_{l_g}^{(g)}$ with selection probabilities: $c_1^{(g)} \dots c_{l_g}^{(g)}$. So, the number of the different function set for the whole network of n nodes is $l_1 \times l_2 \times \dots \times l_n$. Then every instantiation of the function set can be assigned a probability by multiplying individual probabilities.

Consider a gene g whose state at time t is going to be determined from $S(t)$. Clearly, the state of g at time $t+1$ is 1 with the probability $f_1^{(g)}(S(t))c_1^{(g)} \times f_2^{(g)}(S(t))c_2^{(g)} \dots \times f_g^{(g)}(S(t))c_g^{(g)}$. So, we can compute the transition probability from any given $S(t)$ to any $S(t+1)$. We can easily create a stochastic matrix in the state space of genes. This stochastic matrix is MC because every stochastic matrix is MC. After getting this matrix, the analysis of the dynamics of the network can be done using MC theory.

Note that the selection probabilities of predictors are independent of the state. The selection of predictor sets and their probabilities are important issues in PBNs. Usually the strategy is similar to BNs. That is, each gene is allowed to have at most in-degree K. For each of the K-subsets, the utility of each subset is computed for every possible boolean network. Recall that there are 2^{2^K} such functions for

every K-subset. Error of each function is computed using the coefficient of the determination (CoD) method [78]. The number of predictors are determined apriori. The predictors with highest CoDs are selected, and their selection probabilities are assigned based on their predictive power. Shmulevich *et al* [80] studies the effect of random gene perturbations on a given PBN. The authors show that MC of any PBN with a positive perturbation probability for every gene is ergodic (refer to Section 4.6).

4.5 Bayesian Networks

A Bayesian network (BN) models the probabilistic relationships among a set of variables (continuous or discrete). The network consists of two parts, structure S and probability distribution P associated with S. P is a directed acyclic graph where there is a node for each variable. The lack of edge in the graph encode conditional independencies between variables. Associated with each node (hence each variable) is a conditional probability distribution (CPD), conditioned on parent nodes. P is the collection of all these CPDs.

Let $\mathbf{X} = \{X_1, \cdots, X_n\}$ be the set of variables and $\mathbf{Pa}(X_i)$ be the set of parent variables of X_i in S. By using chain rule of probabilities and conditional independencies, the joint probability distribution (JPD) is:

$$P(X_1, \cdots, X_n) = \Pi_{i=1}^{n} P(X_i | \mathbf{Pa}(X_i)) \tag{4.1}$$

Equation (4.1) states that JPD can be easily computed given S and P. From JPD, any probability of interest can be computed. This process is called *inference*. There are several algorithms for this task, but the general problem of exact inference is NP-Hard [35, 36].

Inference only works when a Bayesian network is given. For cases where the experts are not available to create networks, or the domain is not known well (e.g., regulation of genes) the network should be learned from the data. Learning involves both learning the structure of network and its parameter values. Also, the objective network might be desired to model the static or dynamic nature.

4.5.1 Learning Bayesian Networks

In Section 4.5, it is assumed that BN is given. But most of the time, one is resort to learn (induce) it from data and background knowledge because it might be too

complex and sometimes impossible for experts to specify, e.g., genetic regulation networks.

Learning BNs from data is classified into 4 kinds of tasks [67],

- Known Structure, Full Observability
- Known Structure, Partial Observability
- Unknown Structure, Full Observability
- Unknown Structure, Partial Observability

In the known structure cases, it is only needed to learn P, whereas in the unknown structure cases, both S and P must be induced from the data and background knowledge (if available). In the full observability cases, there is no missing value in any cases of dataset. In addition, it is assumed that there is no hidden variable (i.e., attribute set is complete). In the partial observability case, there are hidden variables and/or missing values in the dataset.

4.5.2 Learning Probabilities

Since the distribution of any variable is only dependent on its parent set, we can consider the independent parameters of these distributions. The possible values of these parameters determine the physical probabilities of joint distribution. Given the value of the parameters, the joint physical distribution is given as:

$$P(x_1, \cdots, x_n|\theta_s, S) = \Pi_{i=1}^{n} P(x_i|\mathbf{pa}(X_i), \theta_i, S) \tag{4.2}$$

where S is the known network structure, θ_s is the vector of parameters for each node, i.e., $\theta_s = (\theta_1, \cdots, \theta_n)$, x_i is an instantiation of X_i, and $\mathbf{pa}(X_i)$ is an instantiation of $\mathbf{Pa}(X_i)$.

The problem reduces to estimating θ_s given the network structure and dataset $D = \{x_1, \cdots, x_N\}$. In other words, estimate the posterior distribution $P(\theta_s|D, S)$. Note that not a single value of θ_s, rather a distribution of it is insisted. Each entry, x_i, in D is called a *case*, and each case have a single value for each variable (attribute). Since, parameters for each node are assumed to be independent of those of the other nodes, each parameter distribution can be estimated individually and independently. By this way, the problem is actually the same as probabilistic classification/regression problem. Here, any probabilistic supervised learning method can be used for this task.

Assume each variable X_i is discrete and has r_i possible values. Consider the multinominal distribution, and there are r_i parameters for any single parent configuration $\mathbf{Pa}(X_i)$ of variable X_i. Let the number of parent configurations be q_i for variable X_i. So, the parameter set is:

$$P(x_i^k|\mathbf{pa}^j(X_i), \theta_i, S) = \theta_{ijk} > 0 \tag{4.3}$$

where $j \in \{1, 2, \ldots, q_i\}$ is an index over parent configurations.

Under parameter independence and random sample assumption, the following is obtained:

$$P(\theta|D, S) = \Pi_{i=1}^{n} \Pi_{j=1}^{q_i} P(\theta_{ij}|D, S) \tag{4.4}$$

To compute $P(\theta_{ij}|D, S)$, Bayes rule can be used. Consider that S and any other background knowledge is represented by ξ:

$$P(\theta_{ij}|D, \xi) = \frac{P(\theta_{ij}|\xi)P(D|\theta_{ij}, \xi)}{P(D|\xi)} \tag{4.5}$$

where (by conditioning on θ_{ij})

$$P(D|\xi) = \int P(D|\theta_{ij}, \xi)P(\theta_{ij}|\xi)d\theta_{ij} \tag{4.6}$$

Given the parameter value, the observations in D are mutually independent and their distribution depends only on parameters (i.e., do not depend on background information). So,

$$P(D|\theta_{ij}, \xi) = \Pi_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}} \tag{4.7}$$

where N_{ijk} counts the number of cases of particular instantiations of node X_i together with a configuration of its parents. Assuming Dirichlet priors, posterior distribution of the parameters is,

$$P(\theta_{ij}|D, \xi) = Dir(\theta_{ij}|\alpha_{ij1} + N_{ij1} \cdots \alpha_{ijr_i} + N_{ijr_i}) \tag{4.8}$$

The closed-form formula above can be used to obtain prediction of interest. This is done by averaging over possible values of the parameters.

When a dataset contains missing values, Formula (4.8) can not be used as the number of Dirichlet distributions is exponential in the number of missing value cases. So, approximation techniques are devised to compute posterior probabilities. There

are a number of approximation methods including, *Monte-Carlo methods, Gaussian approximation, expectation maximization (EM) algorithm*, etc [67].

4.5.3 Learning Structure

Structure learning problem is more difficult than learning parameters problem. In this problem, given only the dataset, it is expected to induce both the structure and its parameters. By using Bayes rule, probability of structure for the given dataset is,

$$P(S|D) = \frac{P(S)P(D|S)}{P(D)} \tag{4.9}$$

In the above formula, $P(D)$ is independent of structures and can be considered as normalization constant. For the complete data case (under Dirichlet priors and multinominal distribution), the following is obtained,

$$P(D|S) = \Pi_{i=1}^{n}\Pi_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\Pi_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})} \tag{4.10}$$

The full Bayesian approach presented above is not efficient, since the number of possible model structures are at least exponential. To solve this problem, some heuristics are proposed including, best-first search, greedy search, greedy search with restarts, simulated annealing, among others. These search techniques are local search techniques; so they may stuck to local maxima. For example, greedy search starts from a random structure and considers all possible local changes and computes their score, then it selects the operation that produces maximum score (of course, if maximum is greater than zero). It continues this way until a local maxima is found.

An important problem with model scoring is how to devise the scoring function. First of all, it should be decomposable to be used by local search techniques like greedy search. The *Binary Information Criterion* (BIC) is one such a decomposable scoring function. Besides, it does not need prior distributions to approximate the score of a network for a given dataset.

$$BIC = logP(D|\theta_s, S) - d/2logN \tag{4.11}$$

where $d = \Sigma_{i=1}^{n} q_i(r_i - 1)$ is the dimensionality of the network, and N is the sample size.

To estimate the score of $P(S|D)$ in formula (4.9) we need to estimate the prior $P(S)$ (If not using scoring functions that do not depend on priors). There are

a number of methods proposed for this task. The simplest one is assign every structure the same probability. Another proposal is have an ordering (probably given by users) among variables and reduce structure space and assign remaining ones equal probabilities.

4.5.4 Dynamic Bayesian Networks

Bayesian networks can be used to model dynamic systems. In that case, (like wiring diagrams of BNs) nodes are replicated and indexed with either 0 or 1. For example, g_0 and g_1 are used to represent expression level of gene g at present and next states, respectively. The nodes indexed with 0 are not allowed to have parents, while nodes indexed with 1 can have parents from nodes indexed with 0. So, given the state at any time t, the probability distribution of the next states at time $t+1$ can be found using inference. Similarly, stochastic states at any time (or up to any time) (say $t+k$) can be found by unfolding the network k times.

If the dataset is organized in the form of input/output pairs and forcing the constraint that nodes indexed with 1 can have parent only from nodes indexed with 0 and the nodes with index 0 can not have any parent, the learning problem becomes exactly the same as learning static bayesian networks.

The stochastic property of discrete dynamic bayesian networks implies that these dynamics can be modeled by MC.

In the literature, bayesian networks are used most dominantly for genetic network inference from gene expression data. This is mainly because their result can be visualized and interpreted easily. Also, lots of schemes can be used and explored within bayesian learning framework, such as discrete/continuous data, prior knowledge on the structure, various form for conditional probability distribution, ability to cope with missing values, ability to model latent variables, selecting scoring function, etc. On the other hand, both learning and inference can be very slow depending on the network size.

Friedman *et al* [26] build a discrete state static bayesian model of yeast cell cycle. They do not use any prior biological knowledge (i.e., just use expression data). They apply the bootstrap method [25] to assess the robustness of the algorithm. How to apply techniques for learning models of gene expression data is discussed in [68]. Imoto *et al* [41] propose to use nonparametric regression to construct conditional densities and a scoring method for selecting structure. The proposals are run on the cell cycle data of yeast. How to combine the scoring method proposed with available

prior biological knowledge and dynamic modeling is given in [42] and [55], respectively. Hartemink *et al* [32] presents a scoring method for using gene expression data together with location data (location data specifies which genes are possible targets for transcription factors). They also build bayesian model of galactose system for yeast [33]. How to find interactions between genes and subnetworks using bayesian networks is given in [70]. The case where both experimental (under some external or internal stress) and observational data are to be used for network inference is addressed in [95]. Latent variables are used to represent experimental conditions.

4.6 Markov Chain Model

The work described in [54] constructs MC directly from ternary discretized time course gene expression data. The values for time step t are used as input to time step $t+1$. For each gene (say gene g) a predictor gene set (3 genes) is selected. The conditional distribution of each value (-1, 0, 1) of g over the predictor set is computed as follows. Clearly, $3^3 = 27$ different values of any predictor are possible. For each of these values conditional distribution of -1, 0, and 1 is computed by counting frequencies. For the non-observed case of predictor values, the prior probabilities of -1, 0, and 1 are used for respective conditional probabilities. Combining the conditional distributions of all genes, we get a single MC having 3^n states where n is the number of genes. Note that, even though the space of the MC becomes huge for moderate n values, the method is applicable since it implicitly stores transition probabilities.

Note that the induction method considered does not impose any method for predictor gene selection, but selects only one predictor set for each gene. I consider that selecting only one predictor may harm the effectiveness of the induced model. That is, more than one predictor set can be selected if their predictive powers are almost same.

Noting this, I propose using more than one predictor set for every gene. The justification for this proposal is given next. For instance, assume there are 2 predictor sets with equal or very close predictive power, then selecting only one of them strongly biases the model. In the case of multiple predictor sets, the value is determined again based on conditional distribution of predictors weighted by their power. The other advantage of this scheme is that the probability of prior distribution selected can be reduced, as for a certain state one of the predictor may have no occurrence of its projection in the dataset but the other may have. In Chapter 6,

the proposal is exploited and it has been found that the predictive power of top two predictors are very close.

I note that this is like an analogy between boolean and probabilistic boolean networks (c.f., Sections 4.3 and 4.4). In the former, there is a function determining the output with 100 percent probability, while in the latter multiple functions compete for the determination based on their predictive power.

4.7 Other Modeling Approaches

Beside the methods already discussed in this chapter, there are other methods that have been proposed. They include Linear/Quasi Linear Models (LQLM) [17, 91], Differential Equations (DE) [13, 12], Neural Network (NN) [87, 91], Genetic Programming (GP) [56], and Petri Nets (only representation, i.e. no induction) [65].

In LQLM, each gene is modeled as a linear combination of other genes. Sometimes a small white noise is allowed. Being quasi linear means applying a non-linear function (e.g. sigmoid) on the linear combination. This is done to prevent unbounded growing of expression values.

DE models model the rate of change of the concentration for genes. Usually in the models, there are other variables other than genes, such as mass of the cell, expression levels of proteins and protein complexes, etc. All the variables have continuous values, and given the initial state, the state at any (continuous) time can be computed by integration. Depending on the form of the rate equations, they are further classified as Ordinary DEs, Piece-wise linear DEs, Non-linear Ordinary DEs, etc.

In NN approach, inputs and outputs to the neural network are gene expression values corresponding to a set of genes. Given the input/output pairs, networks weights are learned by training.

GPs identify a target gene for which the network is to be modeled. Possible networks (each network represents a chemical reaction) and weights on arcs are searched in such a way that the model fits the observed data.

4.8 Using DEG Analysis for Interventions

Given enough time series data for an external or internal environmental condition, it is possible to build a corresponding dynamic model. This is irrespective of the employed modeling approach (BNs, PBNs, Bayesian, and Markov Chain modeling). Induced models may vary across the environmental conditions, even same set of

genes are used as model variables. This is because depending on the environmental condition, cells adjust gene specific expression levels (called **homeostatis**) using alternative machineries. So, to induce a regulatory network for each of the different conditions, a time series dataset obtained under that condition is required. This clearly requires time series experiments (usually > 20) for each condition. By this way, the behavior of the organism or the cell under a given condition can be simulated by running the respective model.

On the other hand, since experimentation is expensive and difficult, it might be the case that only few experiments (≈ 5) are available for a specific condition. Since few number of experiments are not enough for model building, we need other methods to solve the problem. In this respect, my proposal is to use the results obtained from the DEGs analysis.

Assume that under normal conditions (treated as reference condition), there are enough experiments and the model is induced. Also assume that for a certain external environmental condition, few experimental data is available. Here, the concern is how to simulate the original model for the given environmental condition. So, this clearly suggests the need for prior information of how the specific condition affects the normal behavior.

In case there is no prior biological knowledge of special effects, these can be acquired by applying the DEGs analysis between the datasets of the normal and a given experimental condition if few samples are available for both conditions. Recalling that DEGs results give the sets of up-regulated and down-regulated genes, i.e., they make the only difference between condition pairs. This clearly suggests whenever a gene is up-regulated, its effect can be simulated by letting its value be over-expressed; the down-regulated case is analogous. Non-differentially expressed genes are left intact.

For BNs, PBNs and bayesian networks, it is given that perturbations to network is possible. The perturbations can be random or pre-defined. Under any perturbation, the behavior of the networks can be observed.

Assume that a PBN genetic network $G =< V, F >$ is induced from normal condition dataset. Also assume that only gene $g \in V$ is up-regulated and no gene is down-regulated between the experimental and normal conditions. So, to simulate model $G =< V, F >$ under the experimental condition, we always set the value of g to 1 and do not change other gene' values. This way, we can simulate a model inferred under a given normal condition for other conditions.

Usually, interventions to organism or cell are classified as external and internal.

Internal interventions are genetic changes (gene deletions and mutations) applied to genome of a cell. External interventions are not related to genetic changes, but the environmental situations (like heat shocks, acid, and varying nutrients) that cells are forced to live in. Given the model $G =< V, F >$ inferred under the normal conditions, I distinguish three cases,

1. The intervention is internal and all the intervened genes are included in V.
2. The intervention is internal and not all the genes are included in V.
3. The intervention is external.

In the first case, the model can be simulated without requiring any dataset for experimental conditions, i.e., just set the appropriate genes to their intervened values. For the second case, although internal intervention is considered, it is effectively external from the perspective of the model. So, this requires experimental data. The third case is trivial and requires experimental data.

In all of the intervention cases, the objective is to infer dynamics of the cell or organism in terms of the selected genes. After modeling the dynamics, the next step is to use them for controlling cell dynamics towards a particular biological objective by means of selectively applying interventions. This is studied in the next chapter. In the case study presented in Chapter 6, the DEGs analysis results are employed to find the effects of certain external interventions. The resulting models (one for each condition) are used to control cell dynamics.

4.9 Discussion

Any dynamic finite state gene expression network can be modeled by MCs. Because any given network can be in exactly one state at any time. This suggests that even the transitions are stochastic, the next state is exactly one of the states in the state space. In this respect, dynamics of BNs can be represented with MCs in which state transitions are deterministic. Actually, dynamics of PBNs and discrete bayesian networks define MC. Furthermore, each instantiation of them corresponds to exactly one MC. However, they differ only in how they store, induce and infer.

CHAPTER 5

INTEGRATED CONTROL AND MONITORING FOR REGULATORY NETWORKS

5.1 Introduction

The advance in microarray technology has influenced the ongoing research on computational biology, and hence attracted the attention of several research groups who have mainly concentrated on analyzing microarray datasets for identifying models for gene regulation network. However, existing achievements can be only considered as exploration towards the true regulation model. Realizing that correctly modeling the regulation process should not be the target of the current research has been the main motivation for producing the novel approach presented in this chapter. Actually, the already handled process is kind of acquiring domain knowledge. It is possible to use induced domain knowledge for different directions in the analysis, although little has been done and reported in the literature. One direction is how to interact efficiently with the model to find perturbations or external controls leading to desirable states of the model.

The approach can be used for scientific discoveries, such as finding how to escape from a given mutant-state to a normal-state, like treatment of cancer. The other advantage is to help scientists in making further experiments with some recommendations, as well as in generating plans from the domain in order to achieve the target of reaching the desired states.

In this chapter, first the control problem is formulated. The required components are state transition function, state cost function and state-action cost function. Four basic scenarios are identified for controlling genetic networks, namely finite control, finite control/infinite monitoring, finite control/finite monitoring, and

infinite control, where the term *control* means applying an external action and the term *monitoring* means watching without applying any external actions. For each of these scenarios, optimal control policies are defined separately for whether the state-action cost function is available or not. Next, multi-objective optimal control method is presented for the case when both the state cost function and state-action cost functions are available, but the domain is not engineered well. Then, on a running example of a PBN model, the control problem is formulated and solved for the cases identified. Finally, how to scale to large networks is addressed.

5.2 The Necessary Background

5.2.1 Markov Decision Processes

From the definition of MC, there is no way for controlling the process; the system evolves without external input when the starting state and transition probabilities are specified. Markov Decision Processes (MDP) extend MCs from control point of view; MDP allow external control on the process in a way that forces the process to be in desirable states.

Definition 5.1 (MDP). A finite MDP mdp is a quadruple [84, 66], $mdp =< \mathcal{S}, \mathcal{A}, T, C >$, where

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action (control) space,
- T is the transition function- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. The transition function T specifies one-step transition probabilities: $T^a_{ss'} = Prob\{S(t+1) = s' \mid S(t) = s,\ A(t) = a\}$, $\forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}$.
- C is the cost function- $C : \mathcal{S} \rightarrow \mathcal{R}$. □

Transition probabilities for every state-action pair sum up to 1. The cost function C assigns a non-negative real value to each state. Some MDP problems use the reward function $R : \mathcal{S} \rightarrow \mathcal{R}$, instead of cost function (C). Sometimes, C (or R) is defined as a function from (state, action) tuples or (state, action, next-state) triples to real values, instead of state to real values.

In MDPs, it is assumed that each action takes a unit amount of time to execute. For some problems, however, the execution time of actions vary depending on the state being in and the type of action. Semi-MDPs address this kind of problems. For a brief introduction to Semi-MDPs refer to the article [19].

5.2.2 Dynamic Programming

Dynamic Programming (DP) is a method to solve optimization problems [10]. It is based on the principle of optimality, which states that every subsolution of an optimal solution must be optimal. It is applicable for problems where the optimal solution can be decomposed into optimal subproblems and there are overlapping subproblems. Due to overlapping subproblems, the solution of these subproblems are stored in a table to avoid same computation again and again for efficiency purposes. For this reason, it is called table based method. Although the method requires decomposing the original problem top down, the computation is bottom up, i.e., combining optimal subsolutions to come up with an optimal solution to the original problem.

DP is employed for wide range of optimization problems. It must be noted here that it is not applicable to all kinds of optimization problems because some problems are hard and can not be decomposed. But, it is applicable for solving MDP problems; and it is in the heart of solving reinforcement learning problems [84].

5.3 Problem Formulation

The control problem addressed here is devised for studying dynamic regulatory networks. In the previous chapter, it has been shown that dynamics of these networks can be abstracted as MC. So, to be general and to address all these networks (MCs, BNs, PBNs, and discrete Bayesian networks), the problem is defined over MCs.

5.3.1 Required Models for Control

To control the cell dynamics, a model of the cell is needed. This way, it becomes possible to get insights how the cell evolves or behaves. The behavior of the model under different perturbations or interventions should be accessible before starting control, otherwise appropriate actions can not be taken. Given the model behaviors under a number of interventions are not adequate for controlling. That is, without an objective the model can only be simulated not controlled. Given the objective, control actions can be applied to force the model to be in desired states of the state space. Also important, in practice, is the cost of applying actions. The cost can be given in terms of money, labor, time, etc.

In this study, three models are considered for the control problem: *Gene regulation network model*, *State cost model*, and *State-action cost model*. The first model

clearly defines the network dynamics, i.e., how the system evolves under different environmental conditions. Even though, network dynamics are known, the purpose with control is how to force network to be in desirable states. The second model is aimed to capture this intuition. The third model originates from the real life, as every control has a cost (money, labor, time, etc). In other words, state-action costs sometimes become crucial.

Definition 5.2 (Gene regulation network model). A generic discrete gene regulation network model M is a triple $M =< S, A, T >$, where S is the discrete state-space, A is the discrete action space and T is the transition function or behavior of the model. □

Actually, S defines the possible states that M can be in at any time; A defines all possible actions that can be applied externally to the model; behavior function T defines dynamics of the model over time; behavior (T) of the model can be either deterministic or non-deterministic.

In the deterministic model case, T is a total function from state-action pairs to states, i.e., $T : S \times A \rightarrow S$. In the non-deterministic model case, T is a total function from state-action pairs to the power set of states; $T : S \times A \rightarrow 2^S$. In both cases, it is said that T is Markovian; the behavior depends only on the current state and the current action. Also, the behavior is stationary (does not change over time).

More useful representation of T is as probability distribution of the next states- $T : S \times A \times S \rightarrow [0, 1]$. Components of T are represented by $T^a_{ss'}$ with the interpretation as probability of the next state is s' after taking action a in state s. Since $T^a_{ss'}$ specifies a probability distribution, it must obey the probability axioms: $\sum_{s'} T^a_{ss'} = 1, \forall s \in S, \forall a \in A$ and $T^a_{ss'} \geq 0, \forall s\ s' \in S, \forall a \in A$.

Model M is termed *generic* because it can be regarded as a black-box with inputs and outputs, i.e., the way of its computation is immaterial. To be more concrete, given state s and action a as inputs, it outputs a next state s' according to its probability distribution. Model M is also termed *discrete* mainly because state transitions happen at discrete times; state-space and action-space are discrete as well.

Once model M is available, it can be used for computational purposes. Since sampling from the model corresponds to simulation of the underlying biological phenomena, it has advantage of being sampled for any state-action for unlimited number of times. Needless to say, the quality of a model (degree of closeness to the true model) determines the validity of the results generated using the model. Finally, to reflect the real world case into a more complete modeling process, I argue that it

is important to have a state cost model and state-action cost model in combination with network model M.

Definition 5.3 (State cost model). State cost model L is a tuple $L =< S, C >$, where S is the state-space and C is the cost function. C is a total function from states to non negative real values- $C : S \rightarrow \mathcal{R}^+$. $\square$

The purpose of model L is to numerically quantize the desirability of states: smallest value for the most desirable state. The state cost model is a central issue in control problems since it is the only way that we discriminate between utility of states. Sometimes, it is more useful to define the state cost model as a reward model. For this interpretation, L is written as $L =< S, R >$, where R is a non-negative reward function: high rewards means desirability of the corresponding state.

Definition 5.4 (State-action cost model). State-action cost model K is a triple $K =< S, A, C >$, where S is the state-space, A is the action-space and C is the state-action cost function from state-action pairs to non-negative real values, denoting the cost of applying actions- $C : S \times A \rightarrow \mathcal{R}^+$. $\square$

Biologically, the costs of applying particular actions across states are not uniform most of the time. Hence, model K just exhibits such costs; it shows the costs for actions applied in states.

We can address different computational problems depending on the source of availability of models M, L and K. But in this thesis, these models are assumed to be provided and no restriction is assumed about how these models are acquired. Sources of these models are not unique, yet multiple competing and inconsistent models are the case for a single biological phenomena. Usually, M is obtained by induction from datasets using induction methods, or from prior biological information (or amalgamation of both).

Model L can be gathered from experts implicitly or explicitly. In the explicit way, experts define a value for each state exhaustively. But, as the state space grows larger this method becomes infeasible and implicit methods are employed. In the implicit way, the rules are defined using some features of the states, thus aggregating states. For instance, a rule can be in the form:

if gene $g1$ *is high and gene* $g2$ *is low then the state cost is* 1

It is important to observe that this kind of rules aggregate the states and sometimes result in a conflict because a state may conform to multiple rules. In some

cases, some states may not conform to any of the specified rules; this case is usually handled by a default state cost value.

In case no expert is available, the rules of L can be induced automatically (given the datasets) by the classification method. However, biological knowledge in the useful format should be made accessible for automated methods.

Model K must be provided by experts as there is no kind of dataset that can be used for inducing this kind of model. This is true because the cost model is a subjective combination of multi-criteria, including time, money, labor, etc. However, if there is no expert and when worse comes to worse, model K may be predicted based on existing cost models that share some past similar experience.

Models M, L, and K are defined separately. The main reason is that their sources and objectives are different as discussed. That is, only M is enough for simulating and studying the dynamics of the underlying phenomena. Also, L and K are considered separately as K is not needed from biological point of view. In other words, L can define the objective of the control in the absence of K. But, as already discussed, in some cases the real life constraints are in effect, so suggesting the need for model K.

Given these three models, in the sequel the methods for constructing optimal policies for control purposes are studied in terms of these models.

5.3.2 Finding Optimal Policies

In this study, two approaches are basically considered for the control based on availability of model K. The first approach applies when the three models M, L and K are given; and the second approach is necessary when only M and L are given. For the latter case, the combined model becomes an MDP (1MDP), which can be used to assess the *utility* of states [73]. The latter case corresponds to state-action costs being unknown, neglected, or unimportant. For the former case, the combined model becomes 2 MDPs (2MDP): M plus L and M plus K. It must be emphasized here that the control problem begins after the specified models are given; it is independent of how the models are generated, e.g., automatic, manual, etc.

The control problem is considered as finding policies for each state given the models. The input to the control problem is a combined model (either 1MDP or 2MDP), and the output is a deterministic policy π- $\pi : S \rightarrow A$. From this point of view, the control problem becomes a planning problem: use the models given to generate a recommendation of what to do given the initial state [84].

It is required to have $|A| \geq 2$, because A must contain at least one special action corresponding to *doing-nothing* and at least one control action. In the absence of external input or control action, cells make transitions according to their normal dynamics. The special action *do-nothing* is aimed to capture this. If the only control input to the model is the action *do-nothing*, then another term, namely *monitoring* should be used instead of controlling. Monitoring is necessary to keep discovering flows that could not be realized or predicated during the control process and hence may appear in the system at a latter stage. Discovering such flows during the monitoring stage brings the control stage into the process again; this saves resources, and even lives when applied in medical treatment systems.

Experimenter (Biologist) can decide on using a strategy that combines control and monitor, depending upon the purpose pursued. Since there is no unique such strategy, I argue that it is necessary to develop optimal control policies for every need.

The central issue in the control problem is the duration (or horizon) of controlling and monitoring. Let H be horizon of control and G be horizon of monitoring. Both H and G can be either finite or infinite. Whether they are finite or infinite is completely determined by the domain and the problem posed. To make analogy, for instance if our time unit is minutes, the treatment of cancer can be assumed infinite, but the treatment of constant bleeding is finite horizon (e.g., 20 minutes). Considerations can be combined into the following cases;

- No control : Zero H
- Finite Control (FC) : Finite H and Zero G
- Finite Control-Finite Monitor (FCFM) : Finite H and Finite G
- Finite Control-Infinite Monitor (FCIM) : Finite H and Infinite G
- Infinite Control (IC) : Infinite H

Depending on the duration of control stage (value of H), the control problem is either *no control* ($H = 0$), *finite control* ($H = a\ positive\ integer$) or *infinite control* ($H = \infty$) (see Figure 5.1). There are three instances of finite control, namely FC, FCIM, and FCFM and one instance, namely IC, of infinite control. Finite control case is considered under three different schemes as they differ in how the monitoring is accounted, i.e., FC does not account for it, FCFM accounts only over a finite duration G, and FCIM accounts for it indefinitely. In this study, *no control* case

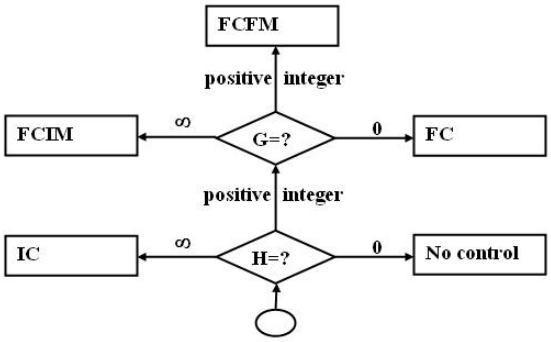


Figure 5.1: The control problem: 5 cases

has been excluded because there can not be any policy for this case. Here, it must be emphasized that the order of control and monitor is fixed: first control then monitoring; this is the natural way things are done.

A common scenario from the medical domain may be articulated as follows: after certain controls are applied, the cell (or cancer patient in the medical domain) is monitored for certain period of time. During monitoring, if everything goes well, no further control is applied; but if something bad is observed, usually the correct way is to apply control again. For example, consider a cancer patient who had been treated for 1 year, and after that monitored for several years. If there is no signs for recurrence, then there is nothing to be done. But, if the patient has some signs of recurrence of the cancer, say after 2 years, the control (medication, radiation, etc) may be decided to start again. As a result, the treatment (or control) problem is in general terms the repeated application of control and monitoring stages. Some specific cases can be obtained from the general case, and optimal controls can be found for such specific cases.

5.4 Finite Control

The assumption here is that for each treatment case (patient), the number of control steps is finite, i.e., doctors do have finite number of treatment options, including gene therapy, chemical therapy, radiation therapy, etc. If the number of control steps is H,

then the problem becomes finding control strategy over time steps $t = 0, 1, \ldots, H-1$. The horizon of monitoring G is zero in this case, so only the state at time $t = H$ is important.

To be general, the optimal policy is assumed state-dependent and non-stationary (may change for a state across time), i.e., the optimal policy is a function of the current state and the remaining time steps. The dependence of the optimal policy on the current state is evident. The dependence over the remaining time steps is also crucial as this is a constraint. As it is going to be shown, the utility of states changes depending on the horizon and the remaining time steps.

In this section, how the optimal policies are driven is shown under the finite control assumption based on two cases: whether state-action costs are available or not.

5.4.1 State-Action Costs Unavailable

The purpose here is deriving an optimal policy for the case. Since state-action costs are not available, only state costs are tried to be minimized stochastically. The stochastic component is model M.

Let $s(t)$ be a random variable denoting the state visited at time t, then $C(s(t))$ is the cost of step t. And let $V_t(s(t))$ be the expected $H - t$ step value function starting at t for state $s(t)$. Given $s(t) = s$, $V_t(s)$ can be written as the expected total cost during the next H time steps starting from state s as given next;

$$V_t(s) = E\big[\sum_{k=t}^{H} C(s(k)) \mid s(t) = s\big] \tag{5.1}$$

Let $\pi = (\pi_0, \pi_1, \ldots, \pi_{H-1})$ be a non-stationary deterministic policy. The value function V_t^π for policy π at time t is defined as the total expected cost for the following H steps acting under the policy π as given next;

$$\begin{aligned}
V_t^\pi(s) = & \quad E\big[C(s(t)) + C(s(t+1)) + \cdots + C(s(H))|s(t) = s\big] \\
= & \quad C(s) + E\big[C(s(t+1)) + C(s(t+2)) + \cdots + C(s(H))|s(t) = s\big] \\
= & \quad C(s) + \textstyle\sum_{s' \in S} T_{ss'}^{\pi_t} \big[C(s') + E\big[C(s(t+2)) + \cdots + C(s(H))|s(t) = s\big]\big] \\
= & \quad C(s) + \textstyle\sum_{s' \in S} T_{ss'}^{\pi_t} V_{t+1}^\pi(s') \qquad (5.2) \\
& \quad t = 0, 1, \ldots H-1
\end{aligned}$$

where $V_H^\pi(s) = C(s)$, for any policy and any state s.

The value function given in Formula (5.2) is easily computable by backing up values backward (using dynamic programming), i.e., first computing $V_H^\pi(\cdot)$, then $V_{H-1}^\pi(\cdot)$, and so on until $V_0^\pi(\cdot)$ is determined.

Action value function $Q_t^\pi(s,a)$ is defined to be the average cost of acting by a in s at time t, and then following the action dictated by a given policy π.

$$\begin{aligned} Q_t^\pi(s,a) = \quad & C(s) + \textstyle\sum_{s' \in S} T_{ss'}^a V_{t+1}^\pi(s') \\ & t = 0, 1, \ldots H-1 \end{aligned} \tag{5.3}$$

where V_{t+1}^π is computed using Formula (5.2).

Instead of evaluating a given policy, it is particularly more interesting to find the policy which gives the minimum expected costs for all states simultaneously. This policy is by definition referred to as *optimal non-stationary policy* and denoted π^* with components $\pi^* = (\pi_0^*, \pi_1^*, \ldots, \pi_{H-1}^*)$. Correspondingly, the optimal policy is the unique optimal value function; and it satisfies the following formula (known as *Bellman optimality equation*) [84];

$$\begin{aligned} V_t^{\pi^*}(s) = \quad & \min_{a \in A} E\big[C(s(t)) + C(s(t+1)) + \cdots + C(s(H)) | s(t) = s\big] \\ = \quad & \min_{a \in A} \big[C(s) + \textstyle\sum_{s' \in S} T_{ss'}^a V_{t+1}^{\pi^*}(s')\big] \\ & t = 0, 1, \ldots H-1 \end{aligned} \tag{5.4}$$

Usually $V_t^{\pi^*}(\cdot)$ is denoted $V_t^*(\cdot)$, since there may be more than one optimal policy; but the optimal value function is unique.

The value function for the optimal policy can be computed in the same way as the value function for a given policy, i.e., by backing up values backward. The only burden is computing the value function for every action in each step and choosing the minimum one.

Optimal action value function for state s and action a, denoted $Q_t^*(s,a)$, is given in Formula (5.5) with the interpretation of applying a in s, and then following the optimal policy.

$$\begin{aligned} Q_t^*(s,a) = \quad & C(s) + \textstyle\sum_{s' \in S} T_{ss'}^a V_{t+1}^*(s') \\ & t = 0, 1, \ldots H-1 \end{aligned} \tag{5.5}$$

Using the optimal action value function, the optimal policy is defined as;

$$\begin{aligned} \pi_t^*(s(t)) = \quad & arg\min_{a\in A} Q_t^*(s(t),a) \\ & t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.6}$$

Discounted Setting

In some cases, discounted settings may be desired to account for uncertainty of delayed costs. However, Equation 5.1 is for non-discounted settings. Under the discounted settings, the state value function is (similar to Formula 5.1) defined as;

$$V_t(s) = E\big[C(s) + \sum_{k=t+1}^{H} \gamma^{k-t} C(s(k)) \mid s(t) = s\big] \tag{5.7}$$

where γ $(0 \leq \gamma \leq 1)$ is discount factor for adjusting the importance of the costs obtained based on time.

The optimal value function in the discounted setting can be derived using arguments similar to those in non-discounted settings; it is given in the following formula.

$$\begin{aligned} V_H^*(s) = \quad & C(s) \\ V_t^*(s) = \quad & \min_{a\in A}\big[C(s) + \gamma \textstyle\sum_{s'} T_{ss'}^a V_{t+1}^*(s')\big] \\ & t = 0, 1, \ldots, H-1 \end{aligned} \tag{5.8}$$

The optimal action cost function and the optimal policy can be defined analogously to non-discounted settings.

5.4.2 State-Action Costs Available

When the state-action costs are available, as discussed before, the problem becomes 2MDP problem. These two MDPs should be somehow combined to find optimal policies. The purpose is trying to reach high utility (low cost) states with the smallest possible total action cost.

Let the state function at time step t be $s(t)$, and the control input be $a(t)$. Then, let $C_t(s(t), a(t))$ be the cost of applying action $a(t)$ in state $s(t)$ at time step t. Note that $s(t)$, $a(t)$ and $C_t(s(t), a(t))$ are all random variables. In model K, it is assumed that the state-action cost function does not depend on time. For this reason and

for brevity, subscript t is dropped and action cost is denoted by $C(s(t), a(t))$. The overall cost function can be written as the expected value of the action cost values over all possible next states, starting from the given initial state $s(0)$ [16].

$$E\big[\sum_{t=0}^{H-1} C(s(t), a(t)) \mid s(0)\big] \tag{5.9}$$

The net effect of applying H steps control is ending up in state $s(H)$. Formula (5.9) does not say anything about which states are desired and which states are not. One solution is using function $C(\cdot)$ from model L for this purpose. Datta *et al* [16] proposes to combine $C(s(t), a(t)), t = 0, 1, \cdots, H-1$ and $C(s(H))$ into a finite horizon formula, which is to be minimized as follows:

$$E\big[\sum_{t=0}^{H-1} C(s(t), a(t)) + C(s(H))|s(0)\big] \tag{5.10}$$

Considering $V_0(s(0))$ as the state value function for the given initial state $s(0)$, Formula (5.10) becomes:

$$V_0(s(0)) = E\big[\sum_{t=0}^{H-1} C(s(t), a(t)) + C(s(H))\big] \tag{5.11}$$

The minimization solution to Formula (5.11) is optimal value function $V_0^*(s(0))$ for time $t = 0$ and can be found using dynamic programming as given next in Formula (5.12).

$$\begin{aligned} V_H^*(s(H)) &= C(s(H)) \\ V_t^*(s(t)) &= \min_{a \in A}\big[C(s(t), a) + \textstyle\sum_{s'} T^a_{s(t)s'} V^*_{t+1}(s')\big] \\ & \qquad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.12}$$

Optimal policy can be computed from the optimal action value function using Formula (5.6), which in turn is:

$$\begin{aligned} Q_t^*(s(t), a) &= C(s(t), a) + \textstyle\sum_{s'} T^a_{s(t)s'} V^*_{t+1}(s') \\ & \qquad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.13}$$

The discounted setting for this case can be easily handled similar to Equation 5.7.

Note that all the components in Formula (5.12) become known when the three models M, L and K are given, and all are defined in the same state and action spaces. That is, $C(s(H))$ is from $C(\cdot)$ of model L, $C(s(t), a(t))$ is from $C(\cdot,\cdot)$ of model K and $T^{a(t)}_{s(t)s(t+1)}$ is exactly T component of model M.

Semi-MDP Approach

Semi-MDPs combine state costs and action execution times into a unique framework. Motivated by this, it is considered that state costs and action costs can be combined into a unique framework using the same idea. Although the problem handled here is not concerned with the action execution times, but the action cost, interpreting action costs as execution times enables casting the problem as Semi-MDPs.

The value function for a non-stationary policy π can be defined as follows:

$$\begin{aligned} V^{\pi}_{t}(s) = \quad & E\big[C(s) + (1-\gamma^{C(s,\pi)})\big[C(s(t+1)) + \cdots \\ + \quad & \big[(1-\gamma^{C(s(H-1),\pi)})C(s(H))\big]\cdots\big] \mid s(t)=s\big] \\ = \quad & C(s) + (1-\gamma^{C(s,\pi)}) \cdot \textstyle\sum_{s'} T^{\pi}_{ss'} \cdot V^{\pi}_{t+1}(s') \\ & t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.14}$$

where γ $(0 \leq \gamma < 1)$ is discount factor and $V^{\pi}_{H}(s) = C(s), \forall s \in S$.

The optimal value function can be defined as a minimization solution to Equation (5.14) as follows:

$$\begin{aligned} V^{*}_{t}(s) = \quad & \min_{a\in A}\big[C(s) + (1-\gamma^{C(s,a)}) \cdot \textstyle\sum_{s'} T^{a}_{ss'} \cdot V^{*}_{t+1}(s')\big] \\ & t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.15}$$

Note that $(1-\gamma^{C(s,a)})$ provides variable discount factor for each state-action pair on the long-term cost. When $\gamma = 0$, we get Equation 5.4 by assuming $0^0 = 0$. So, Equation 5.4 is a special case of Equation 5.15. This is particularly important as when state-costs functions are known but desired to be ignored. In that case what is important is minimizing state costs, and the formula reduces to the intended one given in Equation 5.4. Noteworthy, the main quantity in semi-MDP formulation is state cost; not action cost as opposed to Formula (5.10). The optimal action value functions and optimal policies can be defined similar to Equations (5.5) and (5.6), respectively.

The purpose of Equation (5.14) is to make a good parametric balance between

state costs and state-action costs. This kind of balance is very different than that presented in Equation (5.10). In the latter equation, the combination is linear, and needs re-engineering for possible H values. But, the approach here does not depend on the horizon, and the same cost values can be used with varying H values without re-scaling.

Note that there is a technical problem with Formula 5.14; costs should be positive. However, it is already assumed that cost values are non-negative. So, one can easily replace zero costs with an infinitesimal small cost values. By this way, the technical problem disappears.

5.4.3 Execution of Optimal Policy

The control methods studied so far select a single action for every (state, remaining-time) pair. Note that they are computed off-line and performed on-line. This section briefly presents how they are performed.

Noting the similarities between Formulas (5.4), (5.12) and (5.15), optimal action value functions and optimal policies have similar formulas as well, it can be easily realized that optimal value functions and optimal action value functions can be found using dynamic programming. Since it is straightforward to come up with an optimal policy given the optimal action function, it can be pre-computed before the actual control commences.

Algorithm 5 can be used for acting using π^*, starting from initial state s_0. Since $\pi_t^*(s)$ is computed for every time step $t \in \{0, 1, \ldots, H-1\}$ and for every state $s \in S$, the optimal action selection is a trivial computation: just a look-up table with complexity $O(1)$. The function $Execute(s, a)$, performs action a in s, and returns the next observed state.

Algorithm 5 Acting when optimal policy is computed

Require: $\pi_t^*(s(t)), \forall s \in S$ and $t \in \{0, 1, \ldots, H-1\}$ computed
  $s = s_0$
  for $t = 0$ to $H - 1$ **do**
    $a \leftarrow \pi_t^*(s)$
    $s \leftarrow Execute(s, a)$
  end for

5.5 Finite Control-Infinite Monitoring

In the FC approach, it is strictly assumed that the control step consists of H steps (sometimes at most) and everything ends after these steps (or the state remains

thereafter forever). However, it is sometimes more realistic to model the process having finite control stage first and infinite monitoring stage next, i.e., first apply H steps of control then stop controlling and commence monitoring. Usually this kind of control is used in medicine, first apply medication for a certain period (H) and observe the patient for a certain period (G) of time. During G no medication is applied.

For each action, there is MC to be followed if the respective action is applied. This implies that the MC associated with the $do-nothing$ is to be followed during the entire monitoring period; and this means dynamics do not change because there is no external input to the system; the system evolves without perturbation.

Let the MC to be followed during the monitoring stage be mMC. So, state space S, state cost function $C(\cdot)$ and mMC define a single action MDP corresponding to $do-nothing$. The solution for this MDP can be found using dynamic programming techniques. The value iteration method given in Algorithm 6 is the most frequently employed method for finding the value of every state, i.e., solving the given MDP. In the algorithm, $C_{ss'} = C(s')$ because $C(s, do-nothing) = 0$.

The value of states represents their long term (or infinite) cost. This is reasonable since the length of the monitoring stage can not be determined a priori most of the time. So, the canonical approach is assuming infinite monitoring stage.

As it is shown in the sequel, the FCIM case is very similar to the FC case. Particularly, only the value function of the base case changes; after computing the value functions for all states for the base case (H'th step). Similar methods to the FC case can be used.

5.5.1 Available State-Action Costs

Let the value (long term cost) of the states found by the value iteration be $V_\infty(s), s = 1, 2, \cdots, |S|$. So, the optimal solution to the FCIM setting becomes,

$$\begin{aligned} V_H^*(s) = & \quad V_\infty(s) \\ V_t^*(s) = & \quad \min_{a\in A}\left[C(s,a) + \textstyle\sum_{s'} T_{ss'}^a \cdot V_{t+1}^*(s')\right] \\ & \quad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.16}$$

There is only one difference in the initialization step between Formulas (5.12) and (5.16). Although the difference is small, it is considered very important. This is true because in the latter it is considered that after applying H control steps, the process continues to evolve; however, in Equation (5.12), it is assumed that the

system stops. Concerning human life, stopping criterion corresponds either to death or to full-recovery from the illness. Clearly, these are extreme conditions, and most of the time the system (patient) lives with uncertainty. Consequently, the FCIM approach exploits this fact.

Algorithm 6 Value iteration for 1 action

Require: $C(\cdot)$ and $T^{do-nothing}$
Ensure: $V \approx V^*$
  repeat
    $\triangle \leftarrow 0$
    for all $s \in S$ **do**
      $c \leftarrow V(s)$
      $V(s) \leftarrow C(s) + \gamma \sum_{s'} T_{ss'}^{do-nothing} \cdot V(s')$
      $\triangle \leftarrow \max(\triangle, |c - V(s)|)$
    end for
  until $\triangle < \epsilon$

After finding the optimal value functions, acting is straight forward as it has already been shown in the previous section.

5.5.2 State-Action Costs Unavailable

Let C_H be the total cost of states for applying H-steps control, and let $V_\infty(s)$ be the total discounted cost spent in infinite step monitoring stage. So, the optimal value function is defined as minimizing the expected value of the sum of these two costs for all states:

$$V_0^*(s) = \quad minE\big[C(s) + C(s(t+1)) + \ldots + C(s(H-1)) + V_\infty(s(H)) \mid s(0) = s\big]$$

The solution has the same structure as Formula (5.4), but the case $V_H^*(s)$ should be initialized to the one found by Algorithm 6.

5.6 Finite Control-Finite Monitoring

This case is similar to the FCIM case. The only difference is in the execution of the value iteration algorithm for finding the base case values. That is, the algorithm should be iterated G steps if the monitoring horizon is G, instead of until convergence.

After computing the value functions for the base case, the problem clearly becomes FC problem, and since it has been already shown how to solve them for FC

and FCIM, it is not repeated here. Of course, this does not mean that the methods under FCFM case will give the same results as FC and FCIM methods for the same problem.

5.7 Repeated Application of Finite Control Methods

The structure of the solutions for all the finite control methods (FC, FCIM, FCFM) are similar. Actually, the iterative part is the same, but only the base case changes. These methods are developed for cases where the control strategy specified is applied once or applied in independent iterations. In this section, the control strategy is allowed to be specified in terms of repeated applications of these constituents.

Let FC(H), (FCIM(H)), be the FC (FCIM) problem with control horizon H, similarly let FCFM(H, G) be the FCFM problem with control horizon H and monitoring horizon G. Clearly, given the finite control method and its parameters, i.e., only H for FC and FCIM, and both H and G for FCFM, the optimal policies can be computed as discussed so far. But, consider the case: "find optimal policy, for time steps $t \in \{0, \ldots, H1 - 1\} \cup \{H1 + G1, \ldots, H1 + G1 + H2 - 1\}$, where application of FCIM($H1, G1$) followed by application of FC($H2$)". This can be solved by solving FCIM(H1, G1) for $t \in \{0, \ldots, H1 - 1\}$ and FC(H2) for $t \in \{H1 + G1, \ldots, H1 + G1 + H2 - 1\}$, independently. However, such of solutions are not optimal as they do not exploit the full sequential information provided.

Proposition 5.1. *Optimal policies for sequential application of FC, FCIM, and FCFM can be solved within the FC, FCIM, FCFM framework with restrictions.*

Sketch of the Proof: All possible sequential applications are the following (with appropriate parameters; however, note that nothing can follow FCIM)

- FC($H1$) followed by FC($H2$) is FC($H1 + H2$) with no restrictions
- FC($H1$) followed by FCFM($H2, G2$) is FCFM($H1 + H2, G2$) with no restrictions
- FC($H1$) followed by FCIM($H2$) is FCIM($H1 + H2$) with no restrictions
- FCFM($H1, G1$) followed by FC($H2$) is FC($H1+G1+H2$) with the restriction that the action $do - nothing$ is forced from $t = H1$ to $t = H1 + G1$.
- FCFM($H1, G1$) followed by FCFM($H2, G2$) is FCFM($H1 + G1 + H2, G2$) with the restriction that the action $do - nothing$ is forced from $t = H1$ to $t = H1 + G1$.

- FCFM($H1, G1$) followed by FCIM($H2, G2$) is FCIM($H1 + G1 + H2$) with the restriction that the action $do-nothing$ is forced from $t = H1$ to $t = H1 + G1$. □

From Proposition 5.1, it is not difficult to see that the sequential application of any number (not just 2) of controls can be formulated within the framework of three basic controls. So, it can be concluded that all kinds of finite control problems can be formulated in terms of FC, FCIM and FCFM; thus making the framework complete.

5.8 Infinite Control

In some cases, instead of a finite horizon for control, it is infinite or undetermined. For both cases, the reasonable assumption is finding optimal stationary (same for a state at all times) policies. In the infinite control case the approach adopted in finite control cases can not be used because:

1. infinite sum of cost functions is infinite, and
2. it is impossible to find and store different policies for all time steps as the number of time steps is infinite.

For these reasons, value functions should be found using discounting and the time invariant policy. In the sequel, depending on state-action cost function availability, methods of deriving optimal control policies are given.

5.8.1 State-Action Costs Available

Let C_s^a be the sum of the costs of applying action a in state s and the immediate utility of s: $C_s^a = C(s) + C(s, a)$. This is computable for every state-action pair. Under these settings, it is reasonable to minimize the discounted infinite total cost in order to find optimal policies.

The following formula is used for finding the optimal values for function V.

$$\begin{aligned} V^{\pi^*}(s) = & \quad \min_{a \in A} E\left[\sum_{t=0}^{\infty} \gamma^t c_t \mid s(0) = s\right] \\ = & \quad \min_{a \in A} E\left[c_0 + \gamma \sum_{t=0}^{\infty} \gamma^t c_{t+1} \mid s(0) = s\right] \\ = & \quad \min_{a \in A} E\left[c_0 + \gamma V^{\pi^*}(s(t+1)) \mid s(0) = s\right] \\ = & \quad \min_{a \in A} C_s^a + \gamma \sum_{s'} T_{ss'}^a V^{\pi^*}(s') \qquad (5.17) \\ & \quad \forall s \in S \end{aligned}$$

where c_t is the cost at time step t such that $E[c_t|s(t) = s, a(t) = a] = C_s^a$.

The value iteration process presented in Algorithm 7 can be used to solve Equation 5.17 [84]. After computing the state values, the following deterministic policy is applied and Algorithm 5 is used for optimal action selection with horizon $H = \infty$,

$$\pi^*(s) = arg\min_{a \in A} C_s^a + \gamma \sum_{s'} T_{ss'}^a V^{\pi^*}(s') \tag{5.18}$$

Algorithm 7 Value iteration for multiple actions

Require: $C_s^a, s \in S, a \in A$ and T and γ given
Ensure: $V \approx V^*$
  repeat
    $\triangle \leftarrow 0$
    for all $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \min_{a \in A} C_s^a + \gamma \sum_{s'} T_{ss'}^a V(s')$
      $\triangle \leftarrow \max(\triangle, |v - V(s)|)$
    end for
  until $\triangle < \epsilon$

Algorithm 7 closes the gap between the optimal value function and the current value function in each iteration. As the number of iterations h is infinite, the algorithm computes the optimal value function. However, this may turn up as a very large number for a zero error.

For this reason, optimal value function is computed within an error bound of infinitesimal value, ϵ. The nice property of Algorithm 7 is that the number of iterations can be estimated for any given ϵ. Let $C_{max} = max_{s,a} C_s^a, \forall s \in S, \forall a \in A$, and since $\lim_{h \to \infty} \widehat{V}_h^*(s) = V^*(s), \forall s \in S$, the following equation holds,

$$\begin{aligned} V^* - \widehat{V}_h^* = &\quad C_{max} \tfrac{1}{1-\gamma} - C_{max} \tfrac{1-\gamma^h}{1-\gamma} \\ \leq &\quad \epsilon \end{aligned} \tag{5.19}$$

The desired estimation is obtained by solving Equation 5.19 for h.

$$\widehat{h} = \log_\gamma \epsilon \frac{(1-\gamma)}{C_{max}} \tag{5.20}$$

5.8.2 State-Action Costs Unavailable

The optimality equation is the same as Equation 5.17 with $C_s^a = C(s)$. It can be solved using Algorithm 7 with initial values $V(s) = C(s), \forall s \in S$.

5.9 Multi-Objective Control

The two cost functions in Formula (5.10) are assumed to be given by domain experts (biologists). And it is also assumed in the literature that these two costs are given in a manner that they are additive and reflect their true values in the biological domain. Clearly, defining them requires a good knowledge engineering expertise for every value of H, since it is a parameter. This is true because when H changes, these values should be re-scaled to make them additive in order to reach meaningful policies. To be more concrete when the cost functions are kept the same, if H is too small (say 3), the latter cost has much impact on the score than when H is too large (say 20).

I argue that these two costs are not additive because one is expressed in terms of money, labor, facilities, etc; but the other is expressed based on how desirable the states are. In other words, these two costs are expressed in different units. Clearly, these two kinds of costs can be combined with difficulty, if ever possible. To relax this condition, the state-action cost model can be neglected even if it is available. This approach assumes that every action in every state is applicable with small and uniform costs. However, this approach does not work when state-action costs are inevitable. When both state and state-action cost functions are provided and inevitable, the multi-objective (MO) policy might be the best one. This is because, as it is already claimed, these cost functions are not additive or comparable and different actions optimize each cost functions. In other words, this is an inherent trait of the domain for such cases.

5.9.1 Basics of Multi-Objective Optimization

Let X be a decision space, and $f(x) =< f_1(x), f_2(x), \ldots, f_k(x) >$ be a vector of functions defined over X, and $x \in X$. Then each component of $f(x)$ is called an objective, so totaling to k objectives. The multi-objective problem is posed as;

$$\begin{aligned} minimize\ f(x) = \ & < f_1(x), f_2(x), \ldots, f_k(x) > \\ & such\ that\ x\ \in\ X \end{aligned}$$

where the x values minimizing $f(x)$ are called multi-objective optimal solutions. In case $k = 1$, the problem is single-objective optimization, where the meaning of *minimization* is obviously clear, i.e., the decision alternatives (x values) can be ordered. But when $k \geq 2$, the meaning of *minimize* is ambiguous, i.e. decision

alternatives can not be ordered. In other words, subjective preference structures should be defined [59].

For the multi-objective problems, the minimized solution is not necessarily unique. This is mainly because of the lack of objective minimization structure, otherwise the problem is not multi-objective but single-objective. For instance, consider the problem of selecting the minimum of two vectors (solutions) $< 3, 9 >, < 4, 7 >$. Clearly, without a subjective preference structure the problem can not be solved. But, if the preference structure is the norm of the vector the second is minimum.

5.9.2 Constructing Multi-Objective Solution

In the multi-objective case, for a given state and horizon there might be more than one competing actions. In such case, since there is no objective reason to prefer one of them over others, the best is to present them to the decision maker (Biologist) for a subjective decision. Note that this process can not be automated as an action might lead to bias towards optimizing specific objective. This process can be automated in case a unique action optimizes all the objectives simultaneously.

Let $Q_t^*(s,a) =< Q_t^{1*}(s,a), Q_t^{2*}(s,a), \ldots, Q_t^{k*}(s,a) >$ be optimal value function vector of state s at time step t for action a, where $k \geq 1$ is the number of objectives. Given the computed values for $Q_t^*(s,\cdot)$, the relations between any two different actions is defined next.

Definition 5.5 (Dominated). Given any two different actions $a1, a2 \in A$, action $a1$ is said to be *dominated* by action $a2$ at time t in state s if $Q_t^{i*}(s,a1) \geq Q_t^{i*}(s,a2)$, $\forall i \in 1, 2, \ldots, k$ and at least for one i, $Q_t^{i*}(s,a1) > Q_t^{i*}(s,a2)$ holds.

Definition 5.6 (Indifferent). Given any two different actions $a1, a2 \in A$, actions $a1$ and $a2$ are said to be *indifferent* if $a1$ is not dominated by $a2$ and $a2$ is not dominated by $a1$.

If action $a1$ ($a2$) is dominated by action $a2$ ($a1$) at time t in state s, then clearly $a1$ ($a2$) can not be selected for optimal control, as $a2$ ($a1$) is better than $a1$ ($a2$) by considering all the objectives. Note that the dominance relation is *strict partial order* as it is clearly both asymmetric and transitive. If indifferent relation holds between two different actions, preference of one over the other can not be automated objectively. In short, only the indifferent actions not dominated by other actions (i.e. minimal set of actions) are needed to be present in the alternatives presented to the decision maker.

Assume all the values $Q_t^*(s,a), \forall s \in S, \forall a \in A$ are computed. For any s and t, let actions not dominated and dominated be $ndA_t(s)$ and $\bar{ndA}_t(s) = A \setminus ndA_t(s)$, respectively. Then, all the actions in $\bar{ndA}_t(s)$ can be eliminated from further consideration for action selection at t in s.

After computing $Q_t^*(\cdot,\cdot)$ for all $t \in \{0,1,\ldots,H-1\}$, Algorithm 8 can be used for sequential optimal action selection, where the *Select* function presents to the decision maker a set of non-dominating actions with the value of actions for the state in case $|ndA_t(s)| > 1$. And in case $|ndA_t(s)| = 1$, it automatically selects the respective action. Both s and $Q_t^*(s, ndA_t(s))$ are also included in the *Select* function to help the expert in subjective decision making.

Algorithm 8 Acting when MO value functions are computed

Require: $Q_t^*(s,t), \forall s \in S, \forall a \in A$ and $t \in \{0,1,\ldots,H-1\}$ computed and initial state $s(0)$ given
  $s = s(0)$
  for $t = 0$ to $H-1$ **do**
    $ndA_t(s) = \{a | a \in A, a$ is not dominated in s at $t\}$
    $a \leftarrow Select(ndA_t(s), s, Q_t^*(s, ndA_t(s)))$
    $s \leftarrow Execute(s,a)$
  end for

Since, the actions in the set $ndA_t(s)$ do not dominate and are indifferent to each other. The prior selection probability of any of them can be reasonably assumed equal. This actually originates from being the objective and from the indifferent relation among them. Note that this does not mean that their preference can not be ordered subjectively, but objectively they can not. So, the prior probability of any action $a \in ndA_t(s)$ to be selected to execute is $\frac{1}{|ndA_t(s)|}$.

The central issue is how to compute $Q_t^*(\cdot,\cdot)$. Consider that, the $Q_t^*(\cdot,\cdot)$ values are computed by backing up the values from the next time step $t+1$. This is because the values are assumed to be given for time step H and values for $t = 0,1,\ldots,H-1$ are asked to compute.

In time step t, there are $|ndA_t(s)|$ number of vectors for state s. None of $Q_t^*(s,a), a \in ndA_t(s)$ can determine in isolation the optimal state value (note that the *min* operator is not meaningful). To exploit the fact that the actions in $ndA_t(s)$ have equal selection probability, the optimal state value vector $V_t^*(s)$ can be defined as the component-wise mean of $Q_t^*(s,a), a \in ndA_t(s)$ as follows:

$$V_t^*(s) = \frac{1}{|ndA_t(s)|} < \sum_{a \in ndA_t(s)} Q_t^{1*}(s,a), \sum_{a \in ndA_t(s)} Q_t^{2*}(s,a), \ldots, \sum_{a \in ndA_t(s)} Q_t^{k*}(s,a) > \tag{5.21}$$

The computation of Formula 5.21 is reasonable as $\frac{1}{|ndA_t(s)|}$ acts as a probability and components are values for the cost accrued. Note that in the formula, the components are never compared or added to each other. But, they are considered holistically when the non-dominated actions $ndA_t(s)$ are decided and values are backed up. In other words, the values (and so optimal policy) at time $t-1$ is strongly dependent on the set $ndA_t(\cdot)$, and so on (i.e., the computation is synchronous, e.g., computed first for $t = H-1$, then $t = H-2$, and so on until $t = 0$). So, the components interact for determining optimal policy, but never summed or compared to each other. Actually, this is what we want.

The determination and computation of objectives are clearly application dependent. Given the models M, L, and K, there are two objectives (i.e., $k = 2$), state costs and state-action costs. Let optimal cost vector be $Q_t^*(s,a) =< Q_t^{1*}(s,a), Q_t^{2*}(s,a) >$ and suppose objective 1 is the expected cost for $C(s,a)$ and objective 2 is the expected averaged cost of $C(s)$. Note that there is no explicit or trivial way of deriving $Q_t^*(\cdot,\cdot)$ for a given horizon H for each of the costs. The components are naturally computed as follows:

$$\begin{aligned} Q_H^{1*}(s,a) &= 0 \\ Q_t^{1*}(s,a) &= C_t(s,a) + \textstyle\sum_{s'} T_{ss'}^a \frac{1}{|A_{t+1}(s')|} \sum_{a' \in A_{t+1}(s')} Q_{t+1}^{1*}(s',a') \\ & \qquad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.22}$$

and

$$\begin{aligned} Q_H^{2*}(s,a) &= C(s) \\ Q_t^{2*}(s,a) &= C(s) + \textstyle\sum_{s'} T_{ss'}^a \frac{1}{|A_{t+1}(s')|} \sum_{a' \in A_{t+1}(s')} Q_{t+1}^{2*}(s',a') \\ & \qquad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.23}$$

In both of the formulas above, the auxiliary variable $V_t^*(s)$ given in Formula 5.21 is not shown, instead its explicit form is given. Note that there is no term involving

$C(s)$ in Formula 5.22; and there is no term involving $C(s,a)$ in Formula 5.23. Computations in both formulas are based on backward backup; they should be computed synchronously, i.e., Q_H^{1*} and Q_H^{2*} first, Q_{H-1}^{1*} and Q_{H-1}^{2*} next, and so on. For each s and t, the set of non-dominated actions are determined before passing to $t-1$.

5.10 Running Example

In this section, I show how the control problem can be formulated given a PBN. On the formulated control problem, the behavior of the presented control methods and their effectiveness are demonstrated on a running example. Also, the effectiveness is explored under different problem types (FC, IC, FCIM, and FCFM). All the algorithms are implemented in Matlab.

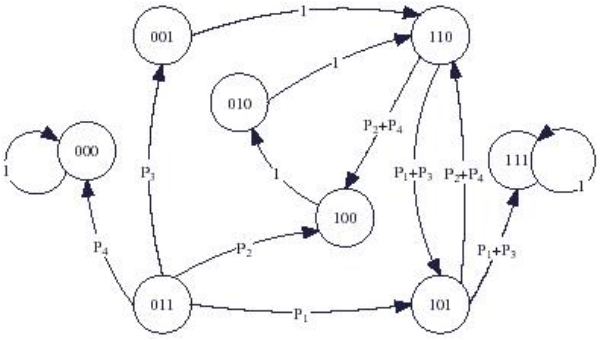


Figure 5.2: State transition diagram for the example (from Shmulevich *et al* [80])

This example is based on the one given in [80], where there are 3 genes, $x1, x2$, and $x3$. There are two predictors for $x1$, one for $x2$ and two for $x3$. So, there are 4 possible selections of predictors for the overall network. Assume these predictors are enumerated as P_1, P_2, P_3 and P_4. The details of how the state transition diagram is generated from the given PBN are omitted here (for details see [80]); the transition diagram is shown in Figure 5.2.

Assume the selection probabilities of the predictors are $P_1 = 0.3$, $P_2 = 0.3$, $P_3 = 0.2$ and $P_4 = 0.2$. After this assignment, the MC that corresponds to normal dynamics of the given 3 genes can easily be computed. So, the MC obtained in the absence of any external control determines the dynamics. Two of the states, namely 000 and 111 are sinks, while others are transient. Biologically, let state 000

represent cancerous state and 111 represent normal state. There are only two states that can lead to cancerous state (state 000 itself and state 011). To match with this discussion, the immediate costs for the states can be assumed as follows;

$$C(s) = \begin{cases} 2, & \text{s=000} \\ 0, & \text{s=111} \\ 1, & \text{otherwise} \end{cases}$$

For control purposes, two kinds of control have been designed, particularly setting $x1$ (the leftmost bit) to either 0 or 1. To match real biological systems, it is assumed that the result of applying controls are non-deterministic; 65% (arbitrarily selected) of the time they have intended result and 35% of the time they have non-intended results with equal probability of resulting in non-intended state. The arbitrary values are selected in such a way that all states are reachable from every state, but with a low probability for non-intended states.

For example, assume that the current state is 001 and we apply the control $x1 = 1$, the intended result is 101 with 65% probability and the other 7 states are resulting states with 0.05% probability each. Under these considerations two MCs are generated; one for $x1$ forced to 0 and the other when $x1$ is forced to 1. So, there are three MCs, one for action 1 (*do-nothing*) and one for action 2 ($x1$ set to 0) and one for action 3 ($x1$ set to 1).

The state action cost function used is;

$$C(s,a) = \begin{cases} 0, & if\ a = 1 \\ 0, & if\ x1 = 0, a = 2 \\ 1, & if\ x1 = 1, a = 2 \\ 1, & if\ x1 = 0, a = 3 \\ 0, & if\ x1 = 1, a = 3 \end{cases}$$

From the action cost function, it is clear that action 1 has no cost irrespective of the states, and the other actions have costs depending on the state (depending only on $x1$). For actions 2 and 3, if $x1$ has the same value as the target action, then the associated cost is 0. Effectively in these cases, only transition probabilities change compared to action 1.

The performance criteria in all of the experiments used is the percentage of being in state 111 and not being in state 000 (difference between these percentages) with hundred percent assumed as being in any of the 8 states. That is high values for state 111 and low values for state 000 are desired. To test the performance, initial states are assumed equiprobable (i.e., 12.5% each). Since state transitions are stochastic, all the results are averaged over 20 runs. Note that the performance criteria selected gives 0 for prior probabilities (i.e., 12.5% for 111 and 12.5% for 000,

so the difference is 0).

5.10.1 State-action cost unavailable

When the state-action costs are not available, the control methods generate policies based only on the biological relevance. So, this case tests the effectiveness of the control methods. Also important is the kind of problem handled (either, FC, IC, FCIM, or FCFM) and the kind of approach employed (either, FC, IC, FCIM, or FCFM). So, to show that for a given kind of problem, the respective solution method is most effective, a series of experiments are done.

Problem is FC

Assume that the actual problem is FC with $H = 3$ (Experimenter applies control for H steps and observes the immediately following state). Here, what is important is the state after 3 steps of control. Clearly, it is accepted as success if the objective is achieved with high percentages after applying 3 steps of control.

FC	IC	FCFM	FCIM	No control
51.5	47.2	41.7	42.7	8.7

The above table shows that all the control methods are better than no control ($do-nothing$, i.e., just applying action 1 irrespective of state and horizon). FCFM result is obtained for control horizon $H = 3$ and monitor horizon $G = 2$ throughout and discount factor is 0.7. The control horizon for FCIM is set to $H = 3$ and discount factor is 0.7 throughout. The FC result is better than all other controls because the problem and the control types match. Although better than the no control case, FCFM and FCIM results are poor compared to FC. This is because they assume that there is a monitoring stage and optimize accordingly. IC result is close to FC, this also makes sense since IC (discount factor is 0.7 throughout) considers the control as lifelong (i.e., there is no monitoring stage and it is true in the experiment case). Also note that no control is better than the score obtained with prior probability. This comes from the dynamics of the MC used; without control it tends to be in 111.

It must be noted that the optimal policies generated by different methods do differ. The policies of each control strategy are presented in Table 5.1. Action 2 is not selected by any of the control methods. This is what we want since it sets $x1$ to 0 and our objective requires $x1 = 1$. The results also show that controlling

Table 5.1: Policies under unavailable state-action costs

	FC			FCFM			FCIM			IC
State	t=0	t=1	t=2	t=0	t=1	t=2	t=0	t=1	t=2	t
000	3	3	3	3	3	3	3	3	3	3
001	3	3	1	3	3	3	3	3	3	3
010	3	3	1	3	3	1	3	3	1	3
011	3	3	3	3	3	3	3	3	3	3
100	3	3	1	3	3	3	3	3	1	3
101	1	1	1	1	1	1	1	1	1	1
110	1	1	1	1	1	1	1	1	1	1
111	1	1	1	1	1	1	1	1	1	1

more than two steps does not help because the policy becomes stationary for all the cases, (the columns $t = 0$ and $t = 1$ are all the same). On the other hand, note that at least one step of control is essential as the column $t = 2$ is different across all the methods and differ from the columns $t = 0$ and $t = 1$. Also note that from state 000 all the methods select action 3, this is reasonable since only actions 2 and 3 can provide exiting from this state; among them action 3 guarantees the exit. Although all the control methods achieve the same basic control, they slightly differ when $t = 2$. This shows that different assumptions on the control problem lead to different policies. Note that the policy found by IC is equal to the column $t = 0$ of others, i.e., IC outputs stationary converged policy.

Problem is IC

Assume that the control problem requires infinite control (Experimenter applies control in each time step). And also assume that FC, FCIM and FCFM are designed as described in Section 5.10.1. Since FC, FCIM and FCFM suggest control only during the first H steps, after H steps, it is assumed that they suggest no control (i.e., action 1). To simulate the IC case, the state at $t = 10$ is sampled for performance evaluation. The results are as follows;

FC	IC	FCFM	FCIM	No control
76	85.2	80.5	80.7	30.5

It is apparent that IC result is the best, and all control methods are superior to the no control case. The results obtained with this setting is very high compared to the one obtained in Section 5.10.1 because the samples are taken at $t = 10$ and

the underlying MC tends to be in 111 as time advances. Note that the policies are exactly the ones presented in Table 5.1 because they have been run with the same parameters.

Comparing the results when the problem is finite (FC) and infinite (IC), it can be recognized that the respective control strategies are superior to others.

Problem is FCIM

Assume that the problem is FCIM (Experimenter applies H steps of control and monitors the system thereafter). To simulate the infinite monitoring, the state at $t = 10$ is sampled. For the IC experiment, the stationary policy is applied during the first 3 steps and then action 1 is applied during the remaining 7 steps. For FCIM, FC, and FCFM, control is applied during the first 3 steps and then action 1 is followed to get the results given next;

FC	IC	FCFM	FCIM	No control
81.2	48	75.5	82	30.5

Again, all the control methods score better than no control. Although FCIM result is slightly larger than FC result, FCIM scores better than all others is inconclusive. But, it can be concluded that it is not worse than others.

Problem is FCFM

Assume that the problem is FCFM (Experimenter applies H steps control and observes G steps of monitoring). Since $H + G = 5$, the methods are evaluated with the resulting state at $t = 5$; and the following results are obtained:

FC	IC	FCFM	FCIM	No control
54.5	52.2	59.2	51.5	24.7

It is conclusive that FCFM scores better than all others; and all the control methods score better than no control.

5.10.2 State-action cost available

Policies of the methods are presented in Table 5.2. Unlike, state-action costs unavailable case, Table 5.2 contains entries where action 2 is selected. This is because when $x1 = 0$, (consider state 000 for instance) applying action 3 guarantees escape,

Table 5.2: Policies under available state-action costs

	FC			FCFM			FCIM			IC
State	t=0	t=1	t=2	t=0	t=1	t=2	t=0	t=1	t=2	t
000	2	2	2	2	2	3	2	2	3	3
001	1	2	1	1	1	1	1	1	1	1
010	1	2	1	1	1	1	1	1	1	1
011	1	2	2	1	2	3	3	3	3	3
100	3	3	1	1	1	3	1	1	1	3
101	1	1	1	1	1	1	1	1	1	1
110	1	1	1	1	1	1	1	1	1	1
111	1	1	1	1	1	1	1	1	1	1

but it has a cost. On the other hand, in this case action 2 has no cost, but non-negative probability of escape. In state 111, none of the methods select action other than 1. This is also reasonable since there is no cost associated to action 1. Note that action 1 is better than action 3, even though action 3 has zero cost. This is because for instance action 3 may lead to 000 with 5% probability from 111. In short, all the methods find basic parts of reasonable actions.

On the other hand, the policies generated differ similar to unavailable state-action costs case. This shows that methods should be chosen based on the experimental strategy planned. As it is already shown, the control strategy matching the experimental strategy scores better than other methods. Conducting similar experiments for this section is not meaningful as state-action costs change the resulting state (i.e., summing together the biological purpose and labor for experimenting, and then only assessing the biological purpose clearly misleads the process).

To use the semi-MDP approach for controlling, it has been already discussed that the state-action costs should be positive. For this reason, replacing 0 valued entries with a small cost value (e.g., 0.1) is required. Under this transformation the policy (with $\gamma = 0.7$) is given in Table 5.3;

Like the other methods, the semi-MDP method gives a reasonable policy satisfying some simple observations under the experimental setting.

As it has been already discussed, under different settings if the state value function and state-action function are carefully designed, the methods presented can be used for control purpose. But, if there is a doubt in these functions or the domain is not engineered well, the multi-objective control should be used.

Table 5.3: Policy found by semi-MDP method

State	t=0	t=1	t=2
000	2	2	2
001	2	2	1
010	2	2	1
011	2	2	2
100	3	3	1
101	1	1	1
110	1	1	1
111	1	1	1

Multi-Objective Control

The MO solution is given in Table 5.4. For some cases, the action presented is singleton (i.e., minimal action set has cardinality 1), which means optimizing both costs at the same time. In some cases, 2-sets are found, when there is no advantage of applying the other action. But, for some cases all the actions are applicable. For any state, the cardinality of MO solution can not be known or anticipated in advance as the problem must be solved totally with the given parameter values.

In case of multiple competing actions, the action can be based on the respective $Q_t^*(\cdot,\cdot)$ values. For example, in the analysis of the Q_t^* values, I observed that first components are almost equal, but second components differ much, or vice versa. Since these value functions do not dominate each other (i.e., it requires strict dominance), these actions are retained. But, the decision maker can easily select the obvious action. In short, there may be a small difference with one component but large with another; this may be a clue for the decision making.

The experiments in Section 5.10.2 forces to select just one action by turning the multi-objective problem into single-objective one. On the other hand, not all the entries in Table 5.4 are singleton sets. This suggests that adding or comparing component cost functions loses power for the sake of the automation of action selection.

5.11 Scaling Up for Large State Spaces

So far it has been assumed that the state space is small. Since the state space increases exponentially with the number of genes, the methods presented in Sections 5.4.1 and 5.4.2 become infeasible or even impossible for true genetic networks.

Table 5.4: Policy found by Multi-Objective control method

State	t=0	t=1	t=2
000	2,3	2,3	1,2,3
001	1,2,3	2,3	1,2,3
010	1,2,3	2,3	1,2,3
011	1,2,3	1,2,3	1,2,3
100	3	1	1,3
101	1	1,3	1,3
110	1	1	1,2,3
111	1	1	1, 3

The most commonly used criteria for large and small state spaces is tabulation of model components. So, if the three models M, L, and K can all be tabulated, then the state-space problem may be considered small. Consider a problem where $|S| = 1000$ and $|A| = 5$, since the tabulation of the T component for model M requires $O(|S|^2|A|)$ space, more than $5,000,000$ entries are required [1]. Though this number is not too large for todays computers, the tabulation of transition probabilities do not make sense when $|S| = 100,000$; even when using efficient data structures like sparse matrices. Even if the tabulation is probable, the time is very important because all the presented methods require sweeps through all the entries in the table. For these reasons, for large state spaces, model components like transition probabilities must be stored implicitly.

Dynamic programming approaches require full backups of value functions. Full backups are expensive with time complexity $O(H|S|^2|A|)$ for H-steps full backup. Furthermore, since transition probabilities are stored implicitly, finding a particular transition probability requires inferencing, which is not a trivial computation. So, full backups are not effective, even though we have all the models.

Scalability is handled using mainly two principles. The first principle is implicitly storing model components and policies instead of explicit tabulation. The policies are computed in such a way that acting and planning are interleaved. The second principle is employing sampling instead of full backups of dynamic programming. This way, near-optimal policies are obtained.

In the sequel, for scaling up to large state spaces, the following two cases are distinguished:

- It is efficient to tabulate all state value or state-action value functions during

[1] A model or a model component having more than few millions of entries is assumed not amenable for tabulation

computation, but tabulation of T of model M is not efficient;

- It is inefficient to tabulate neither state value or state-action value functions nor T of M.

The method employed for both of the cases is simulating model M. For the first case, the value functions for all the states (or state-action pairs) are tabulated. In this case, the models L and K can be given implicitly or stored explicitly. The two approaches for this case are presented, namely *reinforcement learning* and *backward sparse sampling*. Both methods learn approximate value functions for all the states or state-action pairs.

For the second case, in addition to M, models L and K are stored implicitly. Since, the state costs and state-action costs are not tabulated, the inference is required during computation. Two methods (*forward sparse sampling* and *parametric function approximation*) are presented to address the solution to the case. The first method learns approximate value function for a given state, while the second learns approximate value function for all states by generalizing.

All the methods given are formulated under the FC case as the approach for other cases is same as the one already shown.

5.11.1 Reinforcement Learning Approach

Assume that the models are so big that they can not be tabulated, and should be stored implicitly. The only solution then is to simulate the models. Fortunately, *Reinforcement Learning (RL)* methods have been developed for MDP problems, where the models are not fully accessible, but only simulated. In RL, approximate value functions are learned through interaction with the model. By this way, value functions are learned for states that can actually be seen. In other words, value function computations for states that can not be reached (or are reachable with very low probability) from the given initial state can be avoided, thus saving the computation time.

Q-learning is the most commonly employed RL method. It learns action value functions from sample transitions. Assume that upon taking action a in state s, the resulting state is s'; and let the current value of (s, a) be $Q_t(s, a)$, then the value is updated according to the following well-known dynamic programming formula [84].

$$Q_t(s,a) = Q_t(s,a) + \alpha[C(s,a) + \gamma \min_{a'} Q_{t+1}(s',a') - Q_t(s,a)] \tag{5.24}$$

where α is called the learning rate or averaging coefficient, and γ discount factor. After re-arranging Formula (5.24) as $(1-\alpha)Q_t(s,a)+\alpha[C(s,a)+\gamma \min_{a'} Q_{t+1}(s',a')]$ it becomes clear that the new value is an average of the old value and the value of the new state plus the immediate cost. Note that Formula (5.24) has been adapted for the definition given in Equation (5.9). It can be easily modified for the other covered cases.

Algorithm 9 can be used to compute the value function for a given initial state for all actions under the FC approach where the horizon is H.

Algorithm 9 H-step action value computation for a given state using Q-learning

Require: Initial state $s(0)$, T, $C(\cdot,\cdot)$, $C(\cdot)$ and α given
Ensure: $Q_0(s(0),\cdot) \approx Q_0^*(s(0),\cdot)$
  $s_0 \leftarrow s(0)$
  $Q_H(s,a) \leftarrow C(s)$ for every a
  $Q_t(s,a) \leftarrow 0$ for every a, s and $t \in \{0,1,\ldots,H-1\}$
  repeat
    for $t = 0$ to $H-1$ **do**
      Select action a_t using the $Q_t(s_t,\cdot)$
      Simulate execution of a_t for s_t on T and observe resulting state s'_t
      $s_{t+1} \leftarrow s'_t$
    end for
    for $t = H-1$ to 0 **do**
      Update $Q_t(s_t,a_t)$ as given in the Formula (5.24) for transition (s_t,a_t,s'_t)
    end for
    $s_0 \leftarrow s(0)$
  until $Q_0(s(0),\cdot)$ converges

The critical step in Algorithm 9 is how to choose the action given in the current policy function. Here, the ϵ-greedy action selection mechanism can be used, since it satisfies both exploration and exploitation [84]. Another critical step is the termination condition; the process can run for a specified number of iterations depending on the problem size. Since the convergence is only guaranteed for the infinite number of iterations, the resulting value function is always an approximate one. The near-optimal action is then $arg\min_a Q_0(s,a)$ for starting state s and horizon H. After applying the first action, the new state s' is realized, then $arg\min_a Q_1(s',a)$ gives the action to be applied at $t=1$, and so on.

The rationale behind Q-learning is that when large numbers of samples are generated from the model, their distribution almost matches the true transition probabilities. Assuming $|A|$ is constant, the storage requirement of Q-learning is linear in $|S|$. Actually, its space complexity is $O(H|S||A|)$. Consequently, it can be said that Q-learning scales to problems having states in the order of millions in

space (or scales better than full DP backups). Q-learning has another advantage that it computes the value functions for states that are probable to be visited if the initial state is given (and this is the case in our problem). Finally, I argue that all the presented DP algorithms can be easily converted to Q-learning problems and solved accordingly.

5.11.2 Backward Sparse Sampling

In Formula (5.12), the value function for a state is computed using the immediate cost plus the cost of the next states weighted by their transition probabilities. The difficulty with Formula (5.12) is that very small transition probabilities are included in the computation. So, in the worst case, it must be averaged for all states, namely $|S|$ times. Using the idea from RL, if a reasonable number of next states are sampled for a given state and action, the value function computation can be approximated very well. Furthermore, if this number is made constant (independent of $|S|$), the complexity of backing up for a state can be done with complexity $O(1)$. This leads to the following backward sparse sampling formula for the problem given in Formula (5.12),

$$\begin{aligned} \tilde{V}_H^*(s)) = & \quad C(s) \\ \tilde{V}_t^*(s) = & \quad \min_{a \in A} \left[C(s,a) + \tfrac{1}{B} \sum_{b=1}^{B} \tilde{V}_{t+1}^*(s') \right] \\ & \quad t = 0, 1, \cdots, H-1 \end{aligned} \tag{5.25}$$

where s' is generated according to $T_{ss'}^a$ for parameters s and a; and B is the sampling size.

Note that Formula (5.25) is still DP problem, but with reduced complexity. It is different from Q-learning in that there is no problem of selecting policy while learning value functions. But (compared to Q-learning) it learns approximate value functions for states for which visiting probability from starting state is zero.

5.11.3 Forward Sparse Sampling

A variant of the backward sparse sampling is forward sparse sampling. The approximate value function computation is not needed for states other than current state s for decision making. This is because at any time t, the decision is based solely on function $Q_t(s, \cdot)$. Forward sparse sampling just exploits this fact.

Now, assume that the number of samplings for each state is fixed as B for each

action. So, given a state s, the number of next states generated can be at most $|A|B$. If the sampling is repeated for each state generated from the previous iteration, the number of total states generated becomes $O((|A|B)^H)$ at the H'th step. Note that the number of states sampled do not depend on the number of states, but it grows exponentially with H. Here, my argument is that for a reasonable H, the approximate value function for sampled states can be computed by depth-first traversal of the tree. The advantage of depth-first traversal is its use of less memory space; it is linear in H, and independent of $|S|$.

Algorithm 10 Interleaving Acting and Planning

Require: H, B, $s(0)$, T, $C(\cdot,\cdot)$, $C(\cdot)$ given
  $s = s(0)$
  for $t = 0$ to $H - 1$ **do**
    $a \leftarrow Plan(T, s, H - t)$
    $s' \leftarrow Execute(s, a)$
    $s = s'$
  end for

Since the value function is computed only for the current state, the planning and acting should be interleaved. In other words, the acting process presented in Algorithm 5 can not be used; but the one in Algorithm 10 can be used.

The function $Plan(T, s, h)$ (see Algorithm (11)) returns h step best action based on estimates. I adapted the Sparse Sampling Algorithm described in [50]; it is given in Algorithm 11. The $Plan$ has complexity $O((|A|B)^h)$; so the complexity of the algorithm is $O((|A|B)^H)$ for H steps value backup.

Clearly there is a tradeoff between the quality of estimation and run-time. Specifically, large values for B favor for better estimation, but longer run-time.

Like H, the source of B can be biological knowledge provided by biologists. Alternatively, the selection of B can be automated by analyzing the transition matrices for all actions. The method can be based on finding the maximum number of next states possessing the minimum probability selected (e.g., 90%) simultaneously for all (or most, e.g., 90%) states for all actions.

The method can be infeasible for moderate values of H because the complexity is exponential in terms of H. On the other hand, we can reduce the horizon to half or one-third (or so) by considering action sequences as meta actions. Consider action sequences pairs, say $(a1, a2)$, and assume that all the actions are considered in pairs, the runtime complexity becomes $O(C^{H/2}|A|^H))$. Although the complexity is still exponential in H, the runtime greatly diminishes for moderate values of H. In such case, the cost of applying action sequences $(a1, a2)$ from state s is

Algorithm 11 Planning

Function: EstimateV (h, B, T, s)
Let $(\widehat{Q}^*_h(s,a_1), \widehat{Q}^*_h(s,a_2), \ldots, \widehat{Q}^*_h(s,a_{|A|})) = EstimateQ(h,B,T,s)$
Return $arg\min_{a\in A}\{\widehat{Q}^*_h(s,a)\}$

Function: EstimateQ (h, B, T, s)
if $h = 0$ **then**
Return $(C(s), C(s), \ldots, C(s))$
end if
for all $a \in A$ **do**
S_a = Sample $(T(s,a), B)$ {Sample B next states from T at s for a}
$\widehat{Q}^*_h(s,a) = C(s,a) + \frac{1}{B}\sum_{s'\in S_a} EstimateV(h-1,B,T,s')$
end for
Return $(\widehat{Q}^*_h(s,a_1), \widehat{Q}^*_h(s,a_2), \ldots, \widehat{Q}^*_h(s,a_{|A|}))$

Function: $Plan(T, s_0, h)$
Let $(\widehat{Q}^*_h(s_0,a_1), \widehat{Q}^*_h(s_0,a_2), \ldots, \widehat{Q}^*_h(s_0,a_{|A|})) = EstimateQ(h,B,T,s_0)$
Return $arg\min_{a\in A}\{\widehat{Q}^*_h(s_0,a)\}$

$C(s,a1) + \sum_{s'} T^{a1}_{ss'} C(s',a2)$. This is easily computable given models M and K. Its approximation for implicit models is $C(s,a1) + \frac{1}{B}\sum_{j=1}^{B} C(s',a2)$, where s' is drawn according to $T^{a1}_{ss'}$. Similar to state-action costs, state cost is just $C(s) + \sum_{s'} T^{a1}_{ss'} C(s')$. And transition probabilities are computed as: $T^{(a1,a2)}_{ss''} = T^{a1}_{ss'} T^{a2}_{s's''}$.

For discounted MDP problems where the initial state is given, the forward sparse sampling approach can be used. But, the problem is that there is no finite horizon. But, as it is already discussed, the value function computed by Algorithm 7 approximates the true value function as the iteration count increases. Fortunately, Kearns *et al* [50] developed a formula for estimating values of B and H simultaneously within a maximum allowed error ϵ for state values, i.e., $|V^* - \widehat{V}^*_{\widehat{H},\widehat{B}}| \leq \epsilon$ where $\widehat{V}^*_{\widehat{H},\widehat{B}}$ is the result of the discounted Sparse Sampling Algorithm with horizon $\widehat{H}$ and branching factor $\widehat{B}$.

The estimate for H is obtained similar to the one presented in Formula 5.20. B is estimated using Chernoff bounds [2]; it is given as follows (for the proof see [50]);

[2] bounds for random variables for which wrong estimation probability decreases exponentially with the difference between estimates and the true mean

$$\begin{aligned} \widehat{H} &= \lceil \log_\gamma \tfrac{\lambda(1-\gamma)}{C_{max}} \rceil \\ \widehat{B} &= \tfrac{C_{max}^2}{\lambda^2(1-\lambda)^2}\left(2H \log \tfrac{|A|HC_{max}^2}{\lambda^2(1-\lambda)^2} + \log \tfrac{C_{max}}{\lambda}\right) \end{aligned} \tag{5.26}$$

where $\lambda = \epsilon(1-\gamma)^2/4$.

5.11.4 Parametric Function Approximation

The main drawback of the method presented in Section 5.11.3 is that the run time increases exponentially with H; and planning and acting should be interleaved; the other approaches presented so far depend on $|S|$. On the other hand, if the features can be generated from the state information, parametric function approximation methods can be utilized. In our case, fortunately, the state space is defined in terms of features (expression states of genes). To ease notation, s is used for both the state itself and its representation in terms of features.

Assume that the optimal state-action value function for time step t is approximated by $\tilde{Q}_t^*$. Then, the approximation to the optimal state value function for t is $\tilde{V}_t^*(\cdot) = \min_a \tilde{Q}_t^*(\cdot, a)$. Assume $\tilde{Q}_t^*$ is given, then $\tilde{Q}_{t-1}^*$ can be computed; so, (by induction) given $\tilde{Q}_{H-1}^*$, $\tilde{Q}_0^*$ can be computed. In other words $\tilde{V}_t^*(\cdot)$ serves as a negative reinforcement for learning the value function at time step $t-1$. Consequently, the optimal action at time step $t \in \{0, 1, \ldots, H-1\}$ is just $arg\min_a \tilde{Q}_t^*(s, a)$ for any state $s \in S$.

Now consider the case presented in Section 5.4.2; and assume that B samples from model M are drawn for each state s and action a. Then, the following can be defined for the base case:

$$\tilde{Q}_{H-1}^*(s, a) = C(s, a) + \frac{1}{B}\sum_{i=1}^{B} V_H(s_i) \tag{5.27}$$

where s_i is drawn according to $T_{ss_i}^a$ and $V_H(s_i) = C(s_i)$.

Clearly, $\tilde{Q}_{H-1}^*(s, a) = Q_{H-1}^*(s, a), \forall s \in S, a \in A$ as $\lim_{B\to\infty}$. At the same time, for a finite B the formula approximates Q_{H-1}^*. It is easy to compute $\tilde{V}_{H-1}^*(\cdot)$ after $\tilde{Q}_{H-1}^*(\cdot, a)$ is computed for all actions. This way, both $\tilde{Q}_t^*(\cdot, a)$ and $\tilde{V}_t^*(\cdot)$ can be computed for all $t, t \in \{0, 1, \ldots, H-1\}$.

The supervised learning techniques can be used to learn approximate value functions. Particularly, Neural Networks are considered among the best approximation

function techniques and the back propagation algorithm suits the need here.

For learning $\tilde{Q}_t^*(\cdot, a)$, the desired output for state s is $\tilde{V}_{t+1}^*(s')$, if s' is drawn by model M for the input (s, a). This should be repeated for the same input (s, a) for stochastic next state s'. Clearly for learning $\tilde{Q}_t^*(\cdot, a)$, draws must be made for a for a large set of states.

Note that the scenario here is very similar to the tabulated Q-learning already presented above. The only difference is instead of tabulating $Q(\cdot, \cdot)$, we need to represent it by function approximators, which are successfully applied to reinforcement learning problems (Sutton *et al* [84] presents a good overview). Rummery [72] develops connectionist (neural network) learning algorithms for reinforcement learning problems. Particularly, connectionist $Q(\lambda)$ algorithm can be used for the purpose here.

CHAPTER 6

CASE STUDY: SACCHAROMYCES CEREVISIAE

This chapter presents a case study to show how the proposed control methods can be applied to real gene expression data. *Saccharomyces Cerevisiae* (budding yeast) is selected as a case study. The control methods are applied to two kinds of regulatory networks, namely, Markov chain and dynamic Bayesian.

Since every control requires an objective, first of all an objective is set. Next, for this objective, relevant features (genes) are selected to be modeled and a Markov chain model (termed as large) is constructed for the selected genes. Then, control actions and their effects on the model are determined using DEGs analysis. Finally, appropriate control methods are applied.

The original gene set is reduced down and two networks termed as medium and small are obtained. The same procedures are applied to these networks and appropriate control methods are experimented. Also using prior biological knowledge and microarray data, a dynamic Bayesian network is learned. The exact (i.e., finding optimal policy) and approximate (i.e., finding near-optimal policy) methods are experimented and compared on this network.

6.1 Genomics of S.Cerevisiae

Saccharomyces Cerevisiae (S.cerevisiae, a.k.a budding yeast, or yeast for short) is a microscopic unicellular eukaryotic organism with 16 chromosomes. It is one of the first organism completely sequenced, in 1997. Its genetic material consists of 12,052,000 base pairs of nucleotides hosting $\approx$ 6200 Open Reading Frames (ORFs). An ORF is either a gene or a putative gene. The organism is a model organism

for most of the experiments (microarray, biochemical, etc) for several reasons including easy manipulation, rapid-growth, being viable for large set of mutations, commercially cheap, both haploid and diploid states, being non-pathogenic, and easy adaptation to varying conditions.

The diploid cell cycle contains four stages, G1, S, G2 and M (see Figure 6.1). The new-born cell starts growing (G1) and almost ends with bud emergence. After G1 stage, DNA synthesis stage (S) and DNA migration stage (G2) follow. Then, chromosome segregation happens in the mitosis (M) stage. Stage M ends with a new-born daughter cell and cell cycle completes and restarts. Most of the time, the only thing that prevents the exponential rapid growth of yeast culture is lack of nutrients. The cell cycle usually completes in 60-140 minutes depending on external (kind of nutrients, etc) and internal (mutations, etc) conditions. The mother cell dies usually after 20-30 bud formations. The diploid cell cycle is also known as mitotic cell cycle. Haploid cell cycle (a.k.a, meiotic cell cycle) is similar and requires two cells with opposite sex (a and α) for mating, and produces two new daughters.

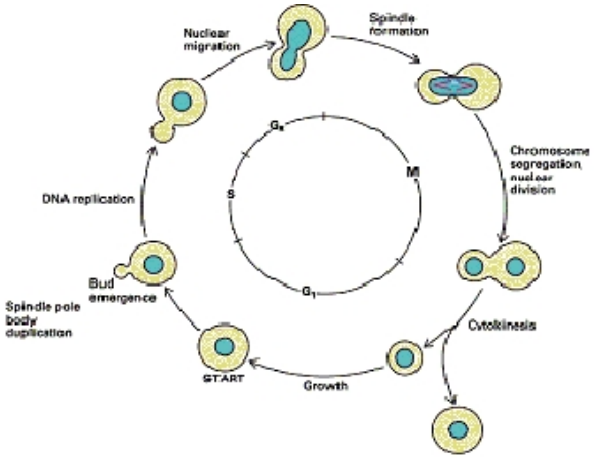


Figure 6.1: Cell cycle stages (from Chen *et al* [12])

Yeast cells are very adaptive. For example, diploid cells undergo *sporulation* (happens during meiosis) in case of nutrient deficiency, and resulting in four encapsulated haploid spores. When exposed to nutrient conditions spores can germinate and mate to diploid cells. Yeast can grow in glucose and galactose mediums. In glucose medium, the organism grows faster and so cell cycle time is shorter. In case of galactose medium, the organism converts first galactose into glucose. The

glucose is fermented to ethanol. If the glucose runs out, the diauxic shift happens, i.e., growth continues aerobically. This suggests that the organism can grow both aerobically and anaerobically.

6.2 Dynamic Modeling of Yeast Cell Cycle

The Stanford Microarray Database (SMD, http://genome-www5.stanford.edu/ MicroArray/SMD/) [77] contains microarray experiments for large number of organisms in various conditions. In general, experimenters publish their data (and their analysis results) in SMD for public use. The database is accessible by web browsers and provides interfaces for data download and analysis. There are several other databases (Whitehead Institute, NCBI, etc.) known worldwide to satisfy same purpose and with similar functionality. In SMD, there are over 40 experiment sets (as of September 2004) present for S.cerevisiae. Each experiment set is done for a specific objective and contains from few to hundreds of experiments.

Spellman *et al* [83] published their yeast cell cycle dataset in SMD. The dataset contains 4 different time series experiments, namely *cdc15, cdc28, elutriation* and *α-factor*. These names also represent different synchronization methods employed in the process. Among them, cdc15 dataset contains 24 samples, and it is the largest. In cdc15 dataset, the samples are taken non-uniformly at following time steps (unit is in minutes): 10, 30, 50, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 270, 290. Recalling that yeast cell cycle takes about 100 minutes, the dataset contains 2-3 cell cycles. The dataset being synchronous makes dynamic modeling possible.

6.2.1 Data Pre-processing

After downloading *cdc15* datasets from SMD, the following pre-processing steps have been applied in order to get a complete continuous dataset. Complete continuous dataset is needed as the methods (discretization) employed here requires complete dataset. Recall that although the gene expression data is continuous it might have missing values, so needs imputation to get complete dataset.

1. For each of the 24 samples, within slide intensity dependent normalization is applied using non-missing values as explained in Section 2.4.1.

2. Samples with more than 20% of missing features are considered bad and are eliminated.

3. Genes with missing value ratio of more than 50% across samples are eliminated.

4. The missing values are imputed using all the experiments in the set using $KNNImpute$ as explained in Section 2.4.2.

In the second step, no samples are eliminated; and in the third step, among the 6389 genes, 501 genes are eliminated. So, the pre-processed complete dataset can be considered as 5888×24 matrix, where rows are gene profiles and columns are samples.

The selection of 20% for missing samples and 50% for missing genes are arbitrary among reasonable values. That is, any rates less than these work, but may cause unnecessary exclusion. On the other hand, rates larger than them are problematic. For instance, 70% for missing samples does not make sense as this suggests that the respective experiment can not be trusted, likewise 80% for missing genes does not make sense, as this shows that there is a systematic problem for the respective gene.

6.2.2 Selecting the Gene Set

Spellman *et al* [83] identified that about 800 genes are cell cycle regulated. This means that 800 genes show varying expression levels during the cell cycle, and about the remaining 5400 genes are not cell cycle regulated, i.e., their expression patterns do not depend on cell cycle. A correct cell cycle model requires all 800 genes are selected for modeling. This is because 5400 remaining genes are considered irrelevant features. But, 800 is too large to be modeled efficiently because the state space is 2^{800}, even in the binary case. In 800 genes, most of the genes exhibit similar expression patterns, i.e., adding all of them makes the model only clumsy. So, these genes must be found and a representative gene should be included in the model. This is because retaining only one of the genes showing almost same expression increases the power of the model as the redundant features are eliminated, i.e., elimination of these genes do not cause loss of information besides helps in simplification of the model.

This way, the number of genes can be reduced down incredibly. Actually, differential equations model of Chen *et al* [12] includes about 40 variables (genes and protein complexes) to model yeast cell cycle. For example, in their model two genes $Clb5$ and $Clb6$ are combined and represented as $Clb5$.

Lee *et al* [60] identifies 106 transcription factors (TFs) for the yeast. They

group these TFs based on their function such as, metabolism, environmental response, cell cycle, etc. Among the 106 TFs, they report 11 TFs are cell cycle related. These are $Ace2, Swi5, Swi4, Swi6, Stb1, Mbp1, Skn7, Ndd1, Fkh1, Fkh2$ and $Mcm1$. Combining these TFs with the genes modeled in [12], 27 genes are obtained. However, 4 of the 27 genes are eliminated by the pre-processing because of missing values. These eliminated genes are $Cdc14$, $Pds1$, $Skn7$ and $Fkh2$. The remaining 23 genes (given in Table 6.1) are included in the model. Indexing the 23 genes from 1 to 23, an expression matrix of 23×24 is obtained.

6.2.3 Markov Chain Modeling of the Selected Gene Set

As it is already pointed out, the cdc15 dataset is not uniformly sampled. To get uniformly sampled data, the dataset is interpolated. The interpolation technique used in this thesis is *cubic splines*, since they are smooth fitters and provide continuity. D'haesseleer [17] uses a similar interpolation method for model fitting.

The other benefit of interpolation is overcoming the less number of samples problem, since any number of samples can be collected by adjusting the sampling interval. But, very frequent sampling is not useful and may mislead as the current and next states become almost identical. On the other hand, very rare sampling has the danger of missing some state changes. That is, cell may change state more than once but it is treated as one. For these reasons, in this study it is kept to be arbitrary but reasonable one, particularly 5 minutes. The exact decision is based on keeping in mind that cell cycle is 100-200 minutes and temporal patterns of key cell cycle genes in the cdc15 dataset. The uniform samples are collected at 5 minutes intervals starting from time 0 to time 290. So, 59 samples are obtained this way for every gene included in the model. As a result, 23×59 expression matrix is obtained.

Interpolated dataset is continuous, but for discrete MC modeling it needs to be discretized. First, the level of discretization should be decided. This is 2 for standard boolean and probabilistic boolean networks. But, this is not a constraint and they can accommodate to any level larger than 2. For discrete network modeling, usually 2 or 3 is used. In this study, ternary discretization is used to express under/over expression and the baseline. Under expression, baseline, and over expression are represented using -1, 0, and 1, respectively. The method used is one standard deviation of mean standard deviations across all original samples (not the interpolated ones). Let, the grand mean be μ, and the average standard deviation be σ, then the following equation is used to determine discretization level of gene g (i.e., x_g).

Table 6.1: Genes selected for modeling

Gene	ORF Name	Gene Function/Description
Cln2	YPL256C	Role in cell cycle START G1 cyclin
Clb2	YPR119W	Involved in mitotic induction; B-type cyclin
Clb5	YPR120C	Role in DNA replication during S phase B-type cyclin
Sic1	YLR079W	Inhibitor of Cdc28p-Clb5 protein kinase complex
Cdc6	YJL194W	Essential ATP-binding protein required for DNA replication
Swi5	YDR146C	Transcription factor that activates transcription of genes expressed in G1 phase and at the G1/M boundary
Cdc20	YGL116W	Cell-cycle regulated activator of anaphase-promoting complex/cyclosome ; directs ubiquitination of mitotic cyclins
Cdh1	YGL003C	Required for Clb2 and Ase1 degradation
Net1	YJL076W	Nucleolar protein involved in exit from mitosis
Tem1	YML064C	Gtp-binding protein of the ras superfamily involved in termination of M-phase GTP-binding protein
Cdc15	YAR019C	Protein kinase of the Mitotic Exit Network that is localized to the spindle pole bodies at late anaphase
Lte1	YAL024C	Gdp/GTP exchange factor required for growth at low temperatures
Bub2	YMR055C	Protein required for cell cycle arrest in response to loss of
Esp1	YGR098C	microtubule function; Shows disrupted cell cycle control
Cln3	YAL040C	Role in cell cycle START; involved in G(sub)1 size control; G1 cyclin
Swi4	YER111C	Involved in cell cycle dependent gene expression; transcription factor
Mbp1	YDL056W	Transcription factor
Swi6	YLR182W	Forms complexes with DNA-binding proteins Swi4p and Mbp1p to regulate transcription at the G1/S transition; Transcription cofactor
Stb1	YNL309W	Protein with a role in regulation of MBF-specific transcription at Start; phosphorylated by Cln-Cdc28p kinases in vitro
Ace2	YLR131C	Transcription factor that activates expression of early G1-specific genes
Ndd1	YOR372C	Nuclear Division Defective 1; arrests prior to nuclear division but after DNA replication
Fkh1	YIL131C	Transcription factor of the forkhead family that regulates the cell cycle and pseudohyphal growth
Mcm1	YMR043W	Involved in cell-type-specific transcription and pheromone response; Transcription factor

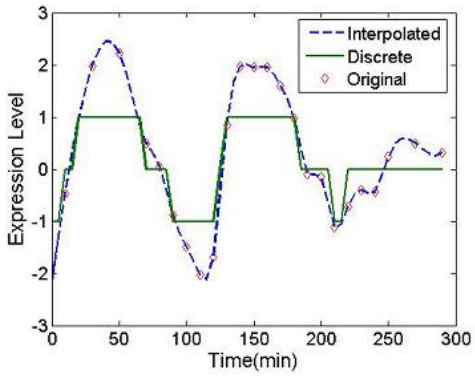


Figure 6.2: Interpolation and Discretization of gene Cln2

$$level(x_g) = \begin{cases} 1, & x_g > \mu + \sigma; \\ -1, & x_g < \mu - \sigma; \\ 0, & \text{otherwise.} \end{cases}$$

Interpolation and discretization for gene $Cln2$ are presented in Figure 6.2. After interpolation and discretization, we get two data matrices, one for interpolated and another for discretized; let them be D_i and D_d, respectively. Note that both D_i and D_d have the same dimension of 23×59.

Determining Predictors

Matrices D_i and D_d are continuous and discrete gene profiles for 23 genes over 59 time points, respectively. Before constructing the MCs, the best predictor genes for every gene should be determined. Using only these genes, we can formulate the dynamic model of the target gene. Let the target gene (the gene we are trying to find its predictors) be g. We can determine the predictors of g using either D_i or D_d.

Kim *et al* [54] proposes using coefficient of determination (CoD) analysis method over D_d. The method is described in [53]. In their method, a perceptron is trained for every gene g for a given predictor set. The values of predictors at time t are used to determine the value of g at time $t + 1$. The error rate of the learned perceptron is used to determine the CoD value. To compare possible predictors, error rates of respective perceptrons are used. That is, the one giving the smallest error rate is

termed as the best predictor.

The method devised by the authors can be used in case D_i is missing, i.e., if only D_d is available. In our case, both matrices are available. For this reason, D_i can be used for determining best predictors; it has been decided to use D_i because it contains much more information than D_d.

Both targets and predictors are continuous valued in D_i. So, to predict the value of gene g at time $t+1$, we can use the values of selected possible predictors at time t. Here, the selection of possible set of predictors matter because there are 2^{23} possible sets. Fortunately, it has been found in biological systems that genes are found only depend on very few number of genes (1, 2, or 3) [61, 8, 16, 54, 12]. For this reason, I only consider the triples of genes as candidate predictors to cover the one and two predictors cases as well.

Let a candidate predictor for gene g be $< g_1, g_2, g_3 >$; and their expression at time t be x_g^t, $< x_{g_1}^t, x_{g_2}^t, x_{g_3}^t >$, respectively. Assuming linear model (to avoid over-fitting and the fact that most relations for S.cerevisiae are reported to be linear [41]), the problem turns into solving the following algebraic equation.

$$\begin{aligned}
x_g^1 = &\quad a_1 \cdot x_{g_1}^0 + a_2 \cdot x_{g_2}^0 + a_3 \cdot x_{g_3}^0 + b \\
\cdots &\qquad\qquad \cdots \\
x_g^{t+1} = &\quad a_1 \cdot x_{g_1}^t + a_2 \cdot x_{g_2}^t + a_3 \cdot x_{g_3}^t + b \\
\cdots &\qquad\qquad \cdots \\
x_g^{59} = &\quad a_1 \cdot x_{g_1}^{58} + a_2 \cdot x_{g_2}^{58} + a_3 \cdot x_{g_3}^{58} + b
\end{aligned}$$

Since this is an over-determined system, the multiple linear regression method is employed to find the coefficients. I use R-square statistics as a measure of CoD. The R-square statistics (CoD) is computed as follows;

$$CoD = \frac{\sum_i (x_g^i - \bar{x}_g)^2}{\sum_i (\hat{x}_g^i - \bar{x}_g)^2}$$

The high values (close to 1) for CoD indicate good fit, while low values (close to 0) indicate poor fit.

For each gene, CoD values are computed for all the 1771 triples (genes are allowed to be within predictors of themselves). Two of them having the highest CoDs are selected as candidate predictors. The results are presented in Table 6.2.

Constructing Markov Chain

Given the predictor sets for every gene, the actual regulation function must be determined. In other words, depending on the value of the predictors, the probability

Table 6.2: Predictor Genes for 23 Genes

Target	Predictor1	CoD1	Predictor	CoD2
Cln2	Cln2 Sic1 Ace2	0.9572	Cln2 Sic1 Swi6	0.9563
Clb2	Clb2 Sic1 Lte1	0.7747	Clb2 Esp1 Cln3	0.7740
Clb5	Cln2 Clb5 Swi5	0.8767	Clb5 Swi5 Cln3	0.8666
Sic1	Cln2 Sic1 Ace2	0.8506	Sic1 Swi6 Ndd1	0.8379
Cdc6	Cln2 Cdc6 Esp1	0.8352	Clb5 Cdc6 Esp1	0.8237
Swi5	Sic1 Swi5 Mcm1	0.8716	Swi5 Bub2 Stb1	0.8691
Cdc20	Cdc20 Lte1 Swi6	0.7397	Cln2 Cdc20 Lte1	0.7327
Cdh1	Cdh1 Cdc15 Mbp1	0.5093	Cdh1 Lte1 Mbp1	0.5084
Net1	Clb2 Net1 Swi6	0.7616	Clb2 Net1 Fkh1	0.7347
Tem1	Cdc6 Tem1 Cd15	0.6065	Tem1 Cdc15 Cln3	0.6051
Cdc15	Cdc15 Cln3 Fkh1	0.7335	Cdc15 Swi6 Fkh1	0.7236
Esp1	Cdc15 Esp1 Bub2	0.8357	Cdc15 Esp1 Swi6	0.8352
Lte1	Esp1 Lte1 Swi4	0.6699	Cdh1 Esp1 Lte1	0.6531
Bub2	Swi5 Tem1 Bub2	0.7111	Cln2 Tem1 Bub2	0.7003
Cln3	Cdc6 Esp1 Cln3	0.8133	Tem1 Esp1 Cln3	0.8090
Swi4	Esp1 Lte1 Swi4	0.6110	Swi5 Cdc20 Swi4	0.5896
Mbp1	Lte1 Bub2 Mbp1	0.6985	Cdc15 Mbp1 Ndd1	0.6981
Swi6	Clb5 Swi4 Swi6	0.6056	Clb2 Cdh1 Swi6	0.6051
Stb1	Bub2 Swi6 Stb1	0.8529	Clb5 Swi6 Stb1	0.8490
Ace2	Clb2 Ace2 Fkh1	0.8040	Clb2 Sic1 Ace2	0.7858
Ndd1	Cdc20 Esp1 Ndd1	0.5856	Cdc20 Net1 Ndd1	0.5853
Fkh1	Ndd1 Fkh1 Mcm1	0.5304	Swi6 Ndd1 Fkh1	0.5268
Mcm1	Swi5 Swi6 Mcm1	0.6059	Cdc15 Swi6 Mcm1	0.5852

of values (-1, 0, and 1) of the target gene should be determined for stochastic modeling. In this section, such context dependent probabilities are computed.

The model has 3^{23} states. Trying to construct and tabulate a MC over this state space is prohibitive. But, the MC over all the state space can be constructed and stored implicitly by exploiting the knowledge that states can be segregated into components of 23 ternary values.

Here, the approach by [54] reviewed in Section 4.6 is adopted. But, here we have two predictors for every gene. So, the key issue is how to fix the conditional probability for a specific gene g given state S. Similar to PBNs, first, one of the predictors is selected probabilistically, and the selected predictor determines the next state based on its own conditional probability. For the cases where D_d does not contain the projected values for both of the predictors in S, another predictor $Prior$ is used. So, there are 3 predictors sets. $Prior$ just gives a probability irrespective of S, i.e., only using the frequency of values of gene g in D_d. Formally, four cases are distinguished:

$$\begin{cases} Prob(P1) = CoD1/(CoD1 + CoD2), & S(P1) \in D_d(P1) \; and \; S(P2) \in D_d(P2) \; ; \\ Prob(P2) = 1 - Prob(P1), Prob(Prior) = 0, & \\ Prob(P1) = 1, Prob(P2) = 0, Prob(Prior) = 0, & only \; S(P1) \in D_d(P1); \\ Prob(P2) = 1, Prob(P1) = 0, Prob(Prior) = 0, & only \; S(P2) \in D_d(P2); \\ Prob(Prior) = 1, Prob(P1) = 0, Prob(P2) = 0, & \text{otherwise.} \end{cases}$$

where $S(P1)$ and $D_d(P1)$ are projected values of predictor1. Except the first case, all other cases are trivial, i.e., the respective predictor determines the next state exclusively. In the first case, however, $Prob(P1)$ and $Prob(P2)$ are treated as weights for respective predictors. That is, predictors are not selected randomly.

The considered MC model is very compact. It has 5 components; 2 predictors, 2 conditional next state functions and 1 prior for every gene. For n genes with k-set predictors, the model size is $O(2 \cdot n \cdot k + 2 \cdot n \cdot 3 \cdot 3^k + 3 \cdot n)$, i.e., linear with n for a constant k.

6.3 Controlling Cell Dynamics

Controlling cell dynamics requires a model (the model M) be constructed and a biological objective set (the model L). The objective defines the cost of being in a given state. Also needed is the number of actions and their associated costs (model K). How to construct model M is shown in the previous section, and how

to construct models L and K is to be shown in this section.

After setting the objective and determining available actions, the solution to the control problem is addressed. First, I start from the large network having 23 genes. Next, I reduce the number of genes to 9 to get a medium size network, and finally to 6 genes to get a small size network. This way, it becomes possible to apply respective algorithms and solve accordingly.

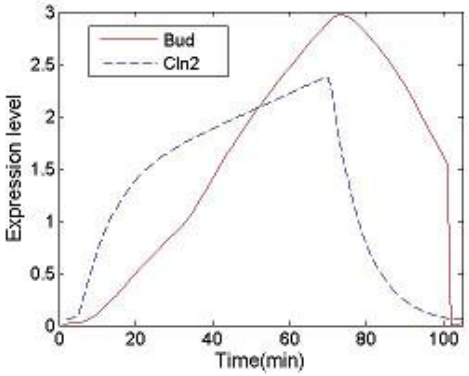
6.3.1 Setting the Objective

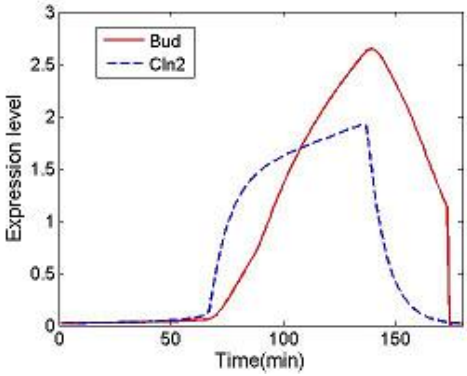
Every control requires an objective to be set. In this study, I set the objective to be extending G1 cycle length. This effectively extends the cell cycle period. The pre-requisite to extending G1 stage or the entire cell cycle period is delaying bud formation. Actually, the objective effectively is reducing the number of cells grown in a fixed time.

The expression level of gene $Cln2$ and the size of the bud are shown in Figure 6.3. Here note that Figure 6.3(a) is for a normal cell (a.k.a. known as Wild Type or WT for short) culture. The cell cycle period is about 100 minutes and bud formation time is about 33 minutes. If the Bud level is larger than 1, it is said that the cell is budded. Figure 6.3(b) shows WT culture response grown in galactose medium (a kind of control). The respective quantities are 180 minutes and 100 minutes. In Figure 6.3(c), the behavior of Bud for the expression level of gene $Cln2$ fixed at 0.06 during the entire cell cycle in glucose medium is simulated. The result show that bud formation time is about 110 minutes and cell cycle period is about 160 minutes. Actually, expression level of 0.06 corresponds to repression of $Cln2$, which extends the bud formation time because its main role is in bud formation. These data are obtained by simulating the Ordinary Differential Equation model for S.cerevisiae of Chen *et al* [12]. So, I set the objective as *down-regulate gene* $Cln2$ using external controls available.

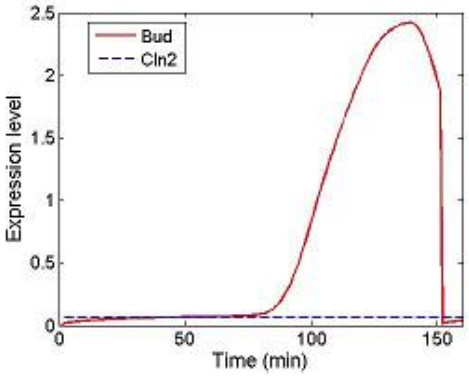
The following state cost function is used to match the biological objective specified above:

$$Cost(S) = \begin{cases} 0, & \text{expression level of Cln2 in state S is -1;} \\ 1, & \text{expression level of Cln2 in state S is 0;} \\ 2, & \text{expression level of Cln2 in state S is 1.} \end{cases}$$





(b) In galactose medium



(c) Constant $Cln2$ level

Figure 6.3: Bud and $Cln2$ levels in three different settings

6.3.2 Selecting External Control Actions

The effects of internal control actions are obvious; they define how the particular gene set values are going to be changed. This approach is taken by Datta *et al* [16] as explained in Chapter 5. On the other hand, following the method proposed in Chapter 4, the internal effects of the external control actions can be determined.

Gasch *et al* [71] conducted a series of experiments for the yeast under varying external conditions, including heat shock, menadione exposure, diamide treatment, DTT exposure, hypo-osmotic shock, nitrogen depletion, amino acid starvation, steady-state growth on alternative carbon sources, etc. The total number of samples in this dataset is 156. This dataset is pre-processed exactly as cdc15 dataset as explained in Section 6.2.1. Due to missing values, 7 of the samples and 47 of the 6418 genes are eliminated. So, the pre-processed matrix has dimensions of 6371×149. This matrix is further normalized so as to match the variance to that of cdc15 dataset; otherwise comparison of samples across two experiments may mislead.

Since DEG analysis requires samples got over the same genes, the common set

Table 6.3: DEGs result for selected conditions

Condition	# up genes	# down genes	DEGs from the model
amino	384	336	Cln3(D), Mbp1(U)
diamide	592	867	Mbp1(U)
heat	496	1	none
hypo-osmotic	133	373	Sic1(D), Mbp1(U), Cln3(D)
menadione	237	1290	Cdc6(D), Cln2(D), Cln3(D), Stb1(D), Mcm1(D)
nitrogen	927	683	none
steady	46	161	Cdc6(D), Cln3(D)

of genes are selected across these two datasets. This gives 5865 genes.

For DEG analysis, first 9 points of cdc15 dataset are selected as reference so as to get samples across the whole cell-cycle. For treatment conditions, the following experimental conditions are selected (abbreviated); amino (5), diamide (8), heat (12), hypo-osmotic (5), menadione (9), nitrogen (9) and steady (8). The number inside parentheses indicate the number of samples of that kind in the dataset. Table 6.3 gives the explored external control conditions and their DEG analysis results. The "U" ("D") next to gene names mean that the gene in such condition is up-regulated (down-regulated). The results are obtained with the minimum confidence of 0.8.

From Table 6.3, some heat and nitrogen conditions have no effect on the selected 23 genes. Other 5 conditions have different effects on the selected genes. The number of affected genes vary across conditions, e.g., 5 for menadione and 1 for diamide. The overall effect of having a condition applied to the model is forcing the respective genes to -1 (1) if it is down (up) regulated. For instance, application of control amino means forcing $Cln3 = -1$, $Mbp1 = 1$ simultaneously, and leaving other genes intact.

Since the heat and nitrogen do not have any effect on the genes in the model, these control actions are eliminated, so leaving 5 actions.

In all of the experiments, what is meant of applying an action is assumed as follows: first forcibly set the values of DEGs genes to respective values and do not alter the other gene values, then make a normal transition under the action *Do Nothing*.

In all of the experiments, the state being in is assumed immaterial for state-action cost. In other words, state-action costs only depend on the action. This is a reasonable assumption since all the actions we consider are external.

6.3.3 Controlling the Large Network

The network with 23 genes is termed as large since it is not efficient to tabulate neither the state-action values nor the state transition probabilities. The former requires $O(3^{23})$ and the latter $O(3^{23} \cdot 3^{23}) = O(3^{46})$ storage space. So, the transition probabilities should be kept implicitly and the near-optimal policy must be computed on line.

The *forward sparse sampling* algorithm presented in Section 5.11.3 can be used to find near-optimal policies for a given initial state using the given cost functions and implicit transition probabilities. The algorithm is implemented and tested on the large network with 5 control actions together with the action *Do Nothing*. The model suggests a control action for a given initial state and horizon.

As it is already considered, the runtime of the algorithm depends on horizon, branching factor and action set size. The horizon and branching factor can be selected by the experimenter. But the utility of every action should be estimated. There are three alternatives for any action on the control objective, 1) not relevant , 2) relevant with adverse effects, 3) relevant with positive effects. Clearly, if a particular action is not relevant with positive effects, then it should be eliminated both biologically and computationally. To illustrate this, assume only one of the actions has positive effects among 5 control actions, and assume branching factor is 10. The runtime with elimination is $O((10 \cdot 2)^H)$; and it is $O((10 \cdot 6)^H)$ without elimination.

To understand the biological relevance, all the state-actions costs should be zero (equivalently, the state-action cost function is ignored). This way, the relevance to goal can be tested. With all action costs set to 0 and with horizon 3, I have evaluated the biological utility of every individual action. The method works as follows, except $Cln2$ the other genes are randomly initialized and $Cln2$ is set to -1, 0, and 1 deterministically. From random initial states, the network is simulated and values of $Cln2$ are tested after H steps. The test is repeated 500 times for each value of $Cln2$ set to -1, 0, and 1. The results (in percents) are shown in Table 6.4 (the controls are shown with initials and Mo stands for monitoring).

The results show that only Menadione and Hyper-osmotic shock can provide relevant and desired effect because the other actions are not better than the monitoring. For this reason, they can be eliminated from the alternatives. But, this does not mean that the eliminated actions can not be beneficial at all. That is, there still might be a small fraction of states that can be beneficial.

Table 6.4: Results of Controlling with individual actions

	M			D			H			A			S			Mo		
	-1	0	1	-1	0	1	-1	0	1	-1	0	1	-1	0	1	-1	0	1
-1	43	42	15	25	48	27	50	33	17	31	42	27	27	42	31	33	40	27
0	40	44	16	17	60	23	38	44	18	20	54	26	16	52	32	16	52	32
1	40	41	19	17	33	50	39	36	25	20	33	47	14	34	52	19	35	46

Table 6.5: Results of considering Menadione and Hyper-osmotic shock together

	-1	0	1
-1	57	30	13
0	55	33	12
1	55	36	09

Shown in Table 6.5 are the results of considering actions Menadione and Hyper-osmotic shock together. The results indicate success over individual considerations.

6.3.4 Controlling the Medium Network

Based on being good predictor for other genes and being predicted by others with high CoD, the number of genes are reduced down to 9. Table 6.6 gives the selected genes and their predictors with CoDs. Markov Chain model is constructed for this set of genes following the same procedure as in the case of large network. The state space size is $3^9 = 19683$, i.e., not too high. So, the state value function can be tabulated. On the other hand, tabulating associated Markov Chain is not efficient since it requires $O(3^{18})$ space [1]. The run time has at least this complexity since all table entries must be filled. That is, even without tabulating, the run time is prohibitive. So, this means that complete dynamic programming backups are not efficient, but the sample backups. This suggests the use of *Backward Sparse sampling* algorithm presented in Section 5.11.2.

Since the set of genes has changed, the effects of the DEGs analysis change on the model as well. For example, action Diamide has no effect on the model as gene $Mbp1$ is not included in the model. The effects of other actions change appropriately.

Table 6.7 compares the number of actions selected over all states for different horizons. In the column where $Cost = 0$, all the costs associated to actions are zero; while for the column $Cost = 1$, the following state-action cost function is used.

[1] Construction of MC of 7-gene network from ternary data takes about 1 hour under Matlab on a Pentium IV 2.0GHz PC running Windows XP with 512Mb of physical memory

Table 6.6: Predictor genes for medium network of 9 genes

Target	Predictor1	CoD1	Predictor	CoD2
Cln2	Cln2 Sic1 Ace2	0.9572	Cln2 Sic1 Cln3	0.9552
Clb5	Cln2 Clb5 Swi5	0.8767	Clb5 Swi5 Cln3	0.8666
Sic1	Cln2 Sic1 Ace2	0.8506	Cln2 Sic1 Swi5	0.8201
Cdc6	Cln2 Cdc6 Esp1	0.8352	Clb5 Cdc6 Esp1	0.8237
Swi5	Sic1 Swi5 Cln3	0.8593	Clb5 Sic1 Swi5	0.8588
Esp1	Clb5 Sic1 Esp1	0.8015	Sic1 Esp1 Stb1	0.8013
Cln3	Cdc6 Esp1 Cln3	0.8133	Cln2 Cln3 Stb1	0.8025
Stb1	Esp1 Cln3 Stb1	0.8253	Cln3 Stb1 Ace2	0.8240
Ace2	Sic1 Cln3 Ace2	0.7791	Clb5 Sic1 Ace2	0.7786

Table 6.7: Profile of actions selected for action costs=0 and action costs=1

	Cost=0			Cost=1		
	H=1	H=2	H=3	H=1	H=2	H=3
Do Nothing	5572	3972	3660	17822	19483	19681
Menadione	6733	4609	4269	1155	65	1
Hypo-osmotic	4062	6152	6209	447	115	1
Amino	1784	2573	2857	128	10	0
Steady	1532	2377	2688	131	10	0

$$C(s,a) = \begin{cases} 0, & \text{a = Do nothing;} \\ 1, & \text{a = Amino,Diamide,Hypo-osmotic,Menadione or Steady.} \end{cases}$$

Table 6.7 shows the importance of adjusting state costs and state-action costs. For example, for $Cost = 1$, horizons with larger than 2 only select actions based on state-actions costs, i.e., state costs are effectively ignored. This suggests design of a good balance. It also shows that the addition of these two cost functions may mislead if not designed properly, as it has been already claimed. Furthermore, the multi-objective control method given in Section 5.9 can be considered for such cases by formulating the two objectives separately as given in Formulas 5.22 and 5.23.

Recall that for large networks, the actions Amino and Steady have been identified as not relevant. But, they become relevant when all the states (i.e., not a random subset) are considered. There seems to be a contradiction, but the following two observations should be noted: 1) For Amino, $Mbp1$ is not within the selected 9 genes, so its effect changes; 2) Occurrence counts of Amino and Steady is less than half compared to Menadione and Hypo-osmotic. Also note that occurrence count

Table 6.8: Predictor genes for small network of 6 genes

Target	Predictor1	CoD1	Predictor	CoD2
Cln2	Cln2 Sic1 Cln3	0.9552	Cln2 Sic1 Swi5	0.9507
Clb5	Cln2 Clb5 Swi5	0.8767	Clb5 Swi5 Cln3	0.8666
Sic1	Cln2 Sic1 Swi5	0.8201	Cln2 Sic1 Cln3	0.7974
Swi5	Sic1 Swi5 Cln3	0.8593	Clb5 Sic1 Swi5	0.8588
Esp1	Clb5 Sic1 Esp1	0.8015	Cln2 Sic1 Esp1	0.8009
Cln3	Cln2 Esp1 Cln3	0.7993	Clb5 Esp1 Cln3	0.7916

of both Amino and Steady is less than *do nothing*, while that of Menadione and Hypo-osmotic is larger. Actually, this also validates the importance of Menadione and Hypo-osmotic towards the goal as it is already claimed for the large network.

6.3.5 Controlling the Small Network

The medium network is further reduced down to 6 genes (Table 6.8), again based on good predictability. This makes tabulation of associated Markov Chains for every action possible. Full dynamic programming backups are possible as well. That is, both time and space complexity become $O(3^{12})$. Since the gene set changes, effects of the actions found by DEGs analysis changes too. Particularly, for example the effects of actions Amino and Steady become the same.

The effects of applying optimal control policy under the state-action costs unavailable condition for FC, IC, FCIM, and FCFM cases are explored. For all of the experiments, the parameter values are $H = 3$, $G = 2$, $\gamma = 0.7$, and the infinite case is simulated at time $t = 7$ and control starts from $t = 0$. The results given are averaged over 50 runs and in percents. As initial states, all the states are used in equal proportion. The column "Mo" indicates no control case.

For the FC problem (see Section 5.10 for what is meant with FC, IC, FCIM, and FCFM problems), the following results are obtained.

$Cln2$	FC	IC	FCIM	FCFM	Mo
-1	47	45	44	46	20
0	40	40	41	40	45
1	13	15	15	14	35

The results indicate that all the control methods score far better than no control. FC seems to score slightly better than other control methods. This easily justify that the problem is FC too.

For the IC problem, the following results are obtained.

$Cln2$	FC	IC	FCIM	FCFM	Mo
-1	20	45	21	20	19
0	44	40	45	45	47
1	36	15	34	35	34

The results indicate that other than IC, all the methods score like not controlling. This clearly means that the MC corresponding to *Do-nothing* is ergodic. In other words, after 5 monitoring steps, all the value functions become almost equal. So, control methods other than IC do not help for this case. This also shows the need for designing control methods for the IC case.

For the FCIM problem, the following results are obtained.

$Cln2$	FC	IC	FCIM	FCFM	Mo
-1	20	20	21	20	19
0	44	45	45	45	47
1	36	35	34	35	34

The results are almost the same, and control strategies do not help here due to ergodicity. From this, for ergodic chains, long step monitoring should be avoided as the effect of initial controls are forgotten.

For the FCFM problem, the following results are obtained.

$Cln2$	FC	IC	FCIM	FCFM	Mo
-1	23	22	24	25	20
0	47	48	45	45	46
1	30	30	31	30	34

The results indicate that all the control methods score slightly better than no control. This is because the MC of *do-nothing* has not reached a steady-state yet, but started to blur values of states.

Like Table 6.7, Table 6.9 shows the importance of balancing the two cost functions. Table 6.9 shows that selection ratios of Menadione and Hypo-osmotic reduce compared to medium-size network. This shows that the power of the model diminishes for small-size networks. Because some interactions can be missed by omitting some of the genes even though the model induction method is perfect. This necessitates developing models capable of handling medium and large size networks.

Table 6.9: Profile of actions selected for action costs=0 and action costs=1

	Cost=0			Cost=1		
	H=1	H=2	H=3	H=1	H=2	H=3
Do Nothing	296	290	311	675	729	729
Menadione	110	170	185	0	0	0
Hypo-osmotic	285	189	146	54	0	0
Amino&Steady	38	80	87	0	0	0

6.4 Controlling Bayesian Network

This section addresses how to control the Bayesian networks using the proposed methods. On a learned Bayesian network, performance of the scaling methods is examined and compared to exact optimal solution.

6.4.1 Network Construction

Figure 6.4 shows the cell cycle subnetwork of S.cerevisiae stored in the KEGG database [1]. This network is also studied by Kim *et al* [55] from the perspective of inducing regulatory networks. The meaning of the arrows in the figure is having *direct influence*. So, the interpretation is very close to Bayesian networks. Except *Fus3* and *Far1*, all other genes are included in Table 6.1.

From Figure 6.4, a subset of the genes given in Figure 6.5 are selected for dynamic Bayesian network construction. I limit the number of genes to be modeled to 7 because a network of 7 genes with ternary data has $3^7 = 2187$ states; it is almost the maximum size that can be solved efficiently with explicit tabulation of MC. So, network of 7 genes can be used for studying all the methods considered, from tabulation of all network components to forward sparse sampling. This way, it becomes possible to assure how good the considered scaling up methods find near-optimal policies.

The objective with the control in Figure 6.5 is forcing $Clb5$ to be down-regulated. This biologically corresponds to delaying *DNA replication* [12]. This is because $Clb5$ is a key gene for starting DNA replication event. The genes in the figure are selected around this objective. Both $Fus3$ and $Far1$ are eliminated based on the fact that given the levels of $Cln1$, $Cln2$, and $Cdc28$, they do not need to be included because of conditional independence. Also, $Cdc28$ is eliminated because it is a key gene involved throughout the cell cycle. This gene is not included in the differential equation model of Kim *et al* [55] under the assumption that it is abundant all the time. In the authors' model, $Cln2$ is included but $Cln1$ is excluded as expression

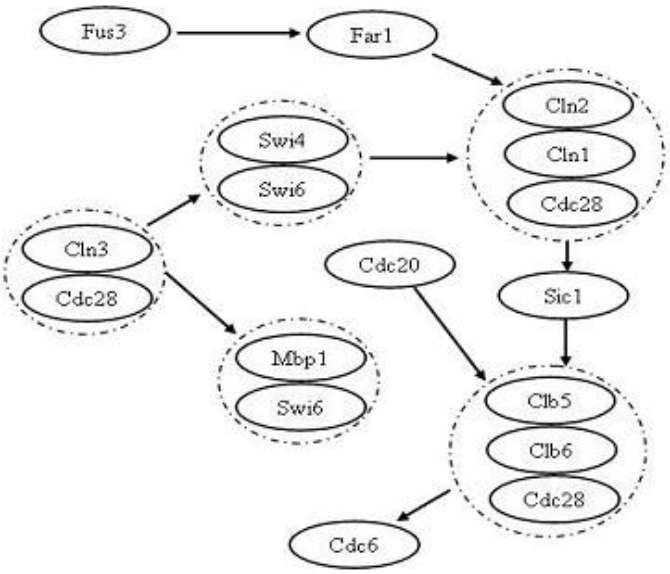


Figure 6.4: KEGG cell cycle network of S.cerevisiae

patterns of $Cln1$ and $Cln2$ are already known to be similar (similar argument applies to $Clb5$ and $Clb6$). So, $Cln2$, $Clb5$ are included and $Cln1$, $Clb6$ are excluded. $Cdc6$ is excluded as it has no influence on $Clb5$. $Swi6$ is excluded as well because it needs to make complexes with $Swi4$ and $Mbp1$ to have influence on $Cln2$ and $Clb5$.

For the remaining 7 genes, the network structure is assumed as given in Figure 6.4. But, the self arrows are added, as the temporal expression of genes is dependent on their expression levels; this is exploited in [55] as well. So, the structure of the resulting dynamic network is obtained as given in Figure 6.5.

After fixing the structure, the parameters are learned under the multinominal distribution with Dirichlet priors. The dataset used for learning is the discretized dataset considered in Section 6.2. After learning the parameters, the transition probability matrix (MC) is constructed from conditional probability tables. So, this chain is stochastically equivalent to the learned network. To study the effects of controls, 2 actions (other than *Do-nothing*), namely *Steady* and *Hypoosmotic* are selected. The MCs corresponding to these actions are computed as follows. The transition probability for every state is replaced by the that of the state when the respective action is taken.

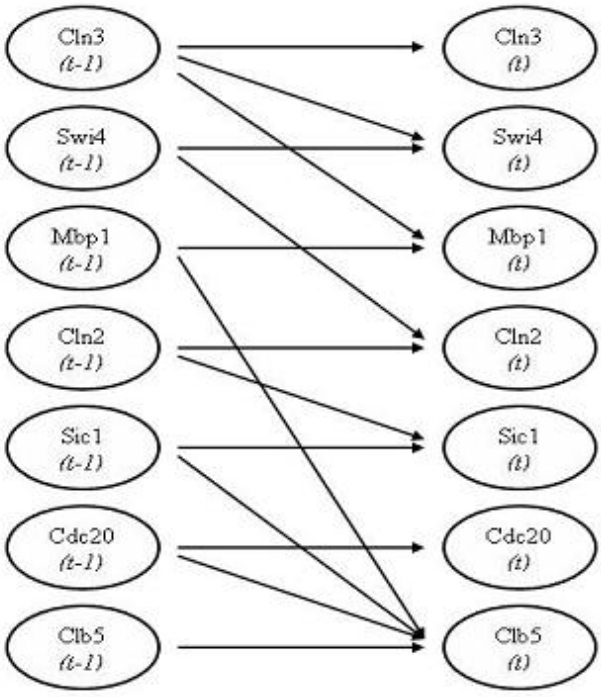


Figure 6.5: Dynamic Bayesian network studied

Table 6.10: Results of Methods for the Bayesian Network

$Clb5=$	No cont.	FC	Back SS, B=10	Back SS, B=5	Forw SS	Q-learning
-1	25	38	37	36	35	34
0	42	38	38	38	36	31
1	33	24	25	26	29	35

6.4.2 Experiments

Four algorithms, FC, Backward sparse sampling, Forward sparse sampling and Q-learning (Figure 9) are run on the control problem constructed next. The state action costs are assumed zero and the cost of a state is 0 when $Clb5 = -1$, 1 when $Clb5 = 0$, and 2 when $Clb5 = 1$. The horizon of control is set to 3 for all methods. The sampling size B is set to 10 and 5 for the Backward sparse sampling algorithm. The branching factor B for Forward sparse sampling is set to 5, and the near optimal policy is computed by running Algorithm 11 for every state and every horizon (i.e., 1,2, and 3). For Q-learning, the parameters are as follows $\alpha = 0.4$, and $\gamma = 0.7$.

Table 6.10 gives the results obtained. The results give the percent of states (according to $Clb5$ level) up to the end of the control period (i.e., the states after the third time step). All of the 2187 states are selected as initial state, and the

results are averaged over 10 runs.

The results clearly show that all the control methods are better than the no control case. Among the 3 control methods, FC is better as it generates optimal policy; but Backward sparse sampling and Forward sparse sampling methods find near-optimal policies. The results also indicate that Backward and Forward sparse sampling algorithms have close performance to FC. This suggests their applicability (i.e. their reliability) for large problems where FC is not applicable. Although better than the no control case, Q-learning exhibits worse performance compared to the other 3 control methods.

6.5 Discussion

In Chapter 5, I have shown the importance of handling various control and monitoring strategies for different settings (FC, IC, etc) on a running example. Also the importance of the multi-objective optimization has been demonstrated. On the other hand, the case study in this chapter shows the importance of a good engineering (i.e., determining the cost and state-cost functions) of the domain before applying control methods. This is true because when actions are associated with high costs, the optimal control strategies do not select any control action, but *Do nothing*. This clearly shows a good balance should be taken before applying controls for finding policies. This suggests that multi-objective control is really needed for such cases, as this control method removes all dominated actions and allows biologists build their preference structure.

The case study also shows the importance of handling FC, IC, FCIM, and FCFM cases separately; this is shown on the small network. All the results indicate that control methods are really useful and score far beyond the no controlling.

Also shown is the importance of the number of genes selected for modeling. For example, the transition from medium to small network shows that effects of some actions may mislead or may become irrelevant.

For model building, MCs are used instead of PBNs. This is mainly because PBNs are not context dependent, i.e., selection probabilities of predictors are independent of the state. PBNs are also experimented for model building, but due to the best-fit extension it has been found that there are usually many predictors for a gene with zero error. So selecting predictors becomes problematic. This is why we select the predictors on continuous dataset. Initially, best linear predictor is decided to be selected, but it is already given in the text that there are other predictors having

score very close to the best one. For this reason, two best predictors are selected. The gene itself is allowed to predict its next state because in real biological systems, most of the genes auto-regulate themselves. Predictor size is set to 3; which is usually the rule of thumb and have biological background.

Internal effects of external actions are determined using DEGs analysis instead of hypothetical interventions. I consider that determining the internal effects of external controls using DEGs analysis is a progress towards developing true control strategies over a given model. Given the almost correct model, the effects of different conditions and treatments can be simulated and used for discoveries. To simulate models under various conditions, the microarray datasets containing up to several samples available on the Internet (i.e., not requiring a special purpose large scale experiments) can be used.

Biological relevance and effects of the control strategies proposed directly depend on the correctness of the model and appropriate engineering. Finally, it must be noted that the control methods presented can be used together with other discrete modeling methodologies including PBNs and Bayesian. In other words, MC modeling step can be replaced by these modeling approaches.

A dynamic Bayesian network is also provided to show that the control methods can be applied on these networks as well. And, the other objective is to show that scaling methods find near-optimal policies close to exact solutions (i.e., optimal policies). The results show that approximate methods proposed for large state spaces can be applied to problems for which exact methods are inapplicable.

CHAPTER 7

SUMMARY, CONCLUSIONS & FUTURE RESEARCH DIRECTIONS

Within the scope of this thesis work, the problems of identifying differentially expressed genes from gene expression data have been studied. In this context, two sophisticated methods (PaGE and q-values) are considered. The focal point in these considerations is to improve the power. To increase the power of PaGE, a method based on estimating proportion of differentially expressed genes is proposed. The lower and upper cutoffs are computed separately and independently in PaGE, it is considered that simultaneous and dependent computation of these cutoffs are also possible. This has the advantage of non-symmetric rejection regions with explorative data analysis notion. A dynamic programming algorithm is presented for efficient computation of confidences. Also, the q-values are extended to single-sided tests where the interpretation becomes more clear, i.e., up-regulation and down-regulation are considered separately.

Improving the power is particularly important for cases where the confidence in differentially expressed genes is desired to be very high (say > 0.8), the number of differentially expressed genes obtained may be very low. This may mislead that some treatments or environmental conditions do not make any difference on the model. To alleviate this problem, methods for improving DEGs analysis within the same confidence level are proposed. This way, effects of treatments on the model will not be missed with higher probability. The proposed methods are applied to synthetic and real gene expression data successfully.

For standard boolean networks, necessary number of experiments for time series expression data for constant in-degree is proven to be $\Omega(\sqrt{n} \cdot log(n))$. This number is $\Omega(log(n))$ for random (non-time series) experiments. It is considered that this is important as time series experiments are not random, but dependent as the name

imply; it is the normal case that actually occur in the real life. The argument used in the proof is information theoretic lower bound.

For inducing MCs directly from expression data, using multiple predictors has been proposed and experimented in the selected case study. It is shown that multiple predictors are really needed as the predictive powers of the best two predictors are shown to be very close to each other. This approach is identified as akin to transition from boolean networks to probabilistic boolean networks.

To control the regulatory networks, the effects of actions on the model should be determined. In the literature, the effects of certain interventions on the genetic networks are studied. They are either random or explorative, i.e., what happens if a particular gene is forced to be over-expressed. In this work, I have suggested a novel connection between differentially expressed gene analysis and effects of interventions. Particularly, effects of certain environmental conditions and treatments on the model are automatically extracted using DEGs analysis results. The approach is particularly important as it is automatic, cheaper and makes general purpose and publicly available datasets considerable for control purposes.

A principled approach for controlling discrete genetic networks are developed. Controlling genetic networks is crucial as inducing models only helps to get insight into the mechanism in the cells. On the other hand, the state can be forcibly changed externally. Since not all the states of the network are equally desirable, the controlling becomes a central issue. So, controlling genetic networks in the direction of specific purpose is a challenging task. This challenge is taken in this work and control methods for various cases are developed. It must be emphasized here that the work presented in this thesis is one of the pioneering in this direction.

The control methods presented here are general that they can be used together with MCs, boolean networks, probabilistic boolean networks, and discrete bayesian networks. The control scenarios are identified as follows: FC, IC, FCFM, and FCIM. The control methods for each case are developed. From this point of view, the control methods presented are general as well. The multi-objective control method presented here is a novel one and I argue and have already demonstrated that it can be used when the domain is not engineered very well. The utility of the methods, their effectiveness, and need for handling different control scenarios are illustrated on a running example. The other direction is generality of the methods, scaling from small to large networks. It is also shown that the framework for finite control cases is complete for sequential application of finite control schemes.

A case study is given to show all the steps of the analysis starting from raw

expression data and following the sequence: model building, setting the objective, exploring the effects of the treatments and environmental stresses, and ending with evaluation of the results. On the case study, the control methods proposed for medium and large networks (small networks as well) are applied and their effectiveness are shown. Also, the control problem is shown to be best solved with the matching approach.

A dynamic Bayesian network is constructed from a prior biological knowledge and microarray dataset. The control problem is formulated and solved under exact and approximate solutions. The experimental results show that the proposed scaling up methods can be applied to large scale problems.

7.1 Future Research Directions

The work presented in this thesis may be expanded in several directions as outlined next.

Clustering Using DEGs patterns

Although differentially expressed genes exhibit significant changes between two conditions, their significance levels are not the same. That is, significance level of genes change within the same set of significant genes. The *pattern generation* problem further clusters significant genes based on ordered significance levels.

In this work, generating patterns is not studied (patterns quantize to what extent genes show differential expression). This clearly suggests a way of discretizing replicated measurements. Given a number of conditions (say 10), we can easily find their pairwise differential expression patterns. In total this gives discrete and relatively high number of dataset size (for 45 for 10 genes). To date, usually the average values of replicated experiments are computed and clustering algorithms are applied on this dataset (10 for 10 conditions). Finding pairwise expression patterns and collecting them together for clustering may help as the sample number is increased; and by quantization original noise in the data may be avoided. Also, the determined dataset can be used for discrete regulation network induction.

Here, it is assumed that data for $k, (k > 2)$ number of conditions are available. For any ordering (but fixed) of these conditions pairwise pattern matrix is generated, where each row corresponds to a gene and each column corresponds to a condition pair. I define similarity between any two genes based on that pattern profile. The

possible similarity metrics are Pearson correlation, manhattan, euclidean and mahalanobis distance measures. With these metrics, some known clustering algorithms (e.g., k-means, hierarchical, etc) can be explored. If k is very large (meaning the cell state is under many conditions) the generated matrix contains information from almost every condition possible. So, clustering gene behaviors under all conditions is our motivation here. I believe that this method is more realistic than clustering under a particular condition.

Data Mining from Model

To date, most of the modeling research is concentrated on building an interaction models of genes. As it is already shown the induced model can be used for controlling. But, I believe that the induced model can be used for other purposes as well. For instance, consider given utmost correct model, any cardinality of data can be sampled from the model. So, the generated data (in large volume) can be used for a number of analyses. One such analysis is association rule mining. I propose to use association rule mining to generate association rules over the data simulated from the model.

The motivation is that since the original dataset (the dataset used to induce model) is scarce, we cannot apply association rule mining. But, after getting the large volume of simulated data, it can be applied. The potential benefit of this approach is, uncovering some dependencies that can not be found by inference or eye-examination of the (particularly large size) model.

Controlling Continuous Models

The presented controlling approach is for discrete state-space networks. This is mainly because of using discrete MCs and MDPs. It seems that the approach can be extended for continuous state-space networks. In this case, for representing state values and state-action values, parametric function approximators can be used. Since the model assumes that state costs and state-action costs are user-defined, any function defined can be used. The functions can range from very simple (such as if expression level of a particular gene is below some threshold use this cost and otherwise use another cost, etc) to very complex functions.

Note that the discrete time is assumed to continue. So, continuous bayesian networks can be used this way for control purposes. Also recall that, since our control methods do not impose any form (like look-up table) on state values and state-action values, they can be used with parametric approximators without change

in the formulas.

Using Eigen-genes to Scale up for Controlling

One of the principled way of dimension reduction is using principal components, since this effectively corresponds to reducing the number of variables to be used with the extent that all the variables become orthogonal. To date, principles components are used for clustering and other analysis. For dimension reduction, usually subsets of genes are selected as shown in the case study. The principle components can not be directly used here as the objective can not be defined in terms of eigen-genes (principal components of genes). If a method capable of transforming original objective into eigen-gene space can be developed, the methods for small network can be used.

BIBLIOGRAPHY

[1] Kyoto encylopedia of genes and genomes database. *http://www.genome.ad.jp/kegg*, Last accessed: October 12, 2004.

[2] Alizadeh A.A. and et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403, 2000.

[3] Osman Abul, Reda Alhajj, and Faruk Polat. Extracting differentially expressed genes from a dataset of microarray experiments. In *(ACM SAC'04)*, Nycosia, Cyprus, March 2004.

[4] Osman Abul, Reda Alhajj, and Faruk Polat. Importance of monitoring in developing optimal control policies for probabilistic boolean genetic networks. In *Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU)*, Perugia, Italy, July 2004.

[5] Osman Abul, Reda Alhajj, and Faruk Polat. Markov decision processes based optimal control policies for probabilistic boolean networks. In *(IEEE) Bioinformatics and Bioengineering (BIBE)*, Taichung, Taiwan, May 2004.

[6] Osman Abul, Reda Alhajj, Faruk Polat, and Ken Barker. Finding differentially expressed genes for pattern generation. *Bioinformatics*, In press, 2004.

[7] Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model. *Theoretical Computer Science*, 298:235–251, 2003.

[8] Tatsuya Akutsu, Satoru Kuhara, and Satoru Miyano. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. In *Pacific Symposium on Biocomputing*, volume 4, pages 17–28, 1999.

[9] Tatsuya Akutsu, Satoru Miyano, and Satoru Kuhara. Algorithms for 'identifying boolean networks and related biological networks based on matrix multiplication and fingerprint function. In *Research in Computational Molecular Biology (RECOMB)*, pages 8–14, 2000.

[10] D.P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Engleewood Cliffs, NJ, 1987.

[11] Alvis Brazma, Helen Parkinson, Thomas Schlitt, and Mohammadreza Shojatalab. A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays. *http://www.ebi.ac.uk.*

[12] Katherine C. Chen, Laurence Calzone, Attila Csikasz-Nagy, Frederick R. Cross, Bela Novak, and John J. Tyson. Integrative analysis of cell cycle control in budding yeast. *Molecular Biology of the Cell*, 15:3841–3862, 2004.

[13] Ting Chen, Hongyu L. He, and George M. Church. Modelling gene expression with differential equations. In *Pacific Symposium on Biocomputing (PSB)*, pages 29–40, 1999.

[14] Zhengxin Chen. *Data Mining and Uncertain Reasoning: An Integrated Approach.* John Wiley Sons, Inc., 2001.

[15] Raymond J. Cho, Michael J. Campbell, Elizabeth A. Winzeler, Lars Steinmetz, Andrew Conway, Lisa Wodicka, Tyra G. Wolsberg, Andrei E. Gabrielian, David Landsman, David J. Lockhart, and Ronald W. Davis. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell*, 2:65–73, 1998.

[16] Aniruddha Datta, Ashish Choudhary, Michael L. Bittner, and Edward R.Dougherty. External control in markovian genetic regulatory networks. *Machine Learning*, 52(2):169–191, 2003.

[17] Patrik D'haesseleer. Reconstructing gene networks from large scale gene expression data. *PhD Thesis, The University of New Mexico*, 2000.

[18] Patrik D'haesseleer, Shoudan Liang, and Roland Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–726, 2000.

[19] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[20] Chris Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. In *Proceedings of the Computational Systems Bioinformatics (CSB)*, pages 523–529, 2003.

[21] Margaret H. Dunham. *Data Mining Techniques and Algorithms*. Prentice-Hall, 2000.

[22] Manduchi E., Grant G.R., McKenzei S.E., Overton G.C., Surrey S., and Stoeckert C.J. Generation of patterns from gene expression data by assigning confidence to differentially expressed genes. *Bioinformatics*, 16:685–698, 2000.

[23] Michael B. Eisen and Patrick O. Brown. DNA arrays for analysis of gene expression. *Methods in Enzymology*, 303:179–205, 1999.

[24] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. Genetics*, 95:14863–14868, December 1998.

[25] Nir Friedman, Moises Goldszmidt, and Abraham Wyner. Data analysis with bayesian networks: A bootstrap approach. In *UAI'99*, 1999.

[26] Nir Friedman, Michal Linial, Iftach Nachman, and Dana Peer. Using bayesian networks to analyze expression data. *Journal of Computational Biology*, 7:601–620, 2000.

[27] Markus F.Templin, Dieter Stoll, Monika Schrenk, Petra C. Traub, Christian F. Vohringer, and Thomas O. Joos. Protein microarray technology. *Drug Discovery Today*, 7(15):815–822, 2002.

[28] Churchill G.A. and Kerr M.K. Bootstrapping cluster analysis: Assessing the reliability of conclusions from microarray experiments. *Pnas Genetics*, 98:8961–8965, 2001.

[29] Dudoit S. Ge Y. and Speed P.S. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Technical Report #633, University of California, Berkeley*, 2003.

[30] Carlos Gershenson. Classification of random boolean networks. *Artificial Life*, VIII:1–8, 2002.

[31] Grant G.R., Manduchi E., and Stoeckert C.J. Using non-parametric methods in the context of multiple testing to identify differentially expressed genes. In *CAMDA*, 2000.

[32] Alexander J. Hartemink, David K. Gifford, Tommi S. Jaakkola, and Richard A. Young. Combining location and expression data for principled discovery of genetic regulatory network models. In *PSB'02*, pages 437–449, 2002.

[33] Alexander John Hartemink. Principle computational methods for the validation and discovery of genetic regulatory networks. *PhD Thesis, MIT*, 2001.

[34] Sampsa Hautaniemi, Olli Yli-Harja, and Jaakko Astola. Analysis and visualization of gene expression microarray data in human cancer using self-organizing maps. *Machine Learning*, 52:45–66, 2003.

[35] David Heckerman. Tutorial on learning with bayesian networks. *Learning in Graphical Models. MIT Press.*, 1999.

[36] David Heckerman, Christopher Meek, and Gregory Cooper. A bayesian approach to causal discovery. *Technical Report MSR-TR-97-05, Microsoft Research*, 1997.

[37] Timothy R. Hughes and Matthew J. Marton. Functional discovery via a compendium of expression profiles. *Cell*, 102:109–126, 2000.

[38] Lawrence Hunter, editor. *Artificial Intelligence and Molecular Biology.* MIT Press, 1993.

[39] Dozmorov I. and Centola M. An associative analysis of gene expression array data. *Bioinformatics*, 19, 2003.

[40] Hedenfalk I. and et. al. Gene-expression profiles in hereditary breast cancer. *Journal of Medicine*, 344(8), 2001.

[41] Seiya Imoto, Takao Goto, and Satoru Miyano. Estimation of genetic networks and functional structures between genes by using bayesian networks and nonparametric regression. In *PSB'02*, pages 175–186, 2002.

[42] Seiya Imoto, Tomoyuki Higuchi, and Takao Goto Kousuke Tashiro. Combining microarrays and biological knowledge for estimating gene networks via bayesian networks. In *Proceedings of the Computational Systems Bioinformatics*, 2003.

[43] Storey J., Taylor J.E., and Siegmund D. Strong control, conservative point estimation and simultaneous conservative consistency of false discovery rates: a unified approach. *J.R. Statistics*, 66:187–205, 2004.

[44] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3), September 1999.

[45] Storey J.D. False discovery rates: Theory and applications to DNA microarrays. *PhD Thesis, Department of Statistics, Stanford University*, 2002.

[46] Storey J.D. and Tibshirani R. Statistical significance for genome-wide experiments. *Manuscript*, 2003.

[47] Daxin Jiang and Aidong Zhang. Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2004, to appear.

[48] Claverie J.M. Computational methods for the identification of differential and coordinated gene expression. *Human Molecular Genetics*, 8, 1999.

[49] Hidde De Jong. Modelling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9(1):67–103, 2002.

[50] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2):193–208, 2001.

[51] Andrew D. Keller, Michel Schummer, Lee Hood, and Walter L. Ruzzo. Bayesian classification of DNA array expression data. *Technical Report UW-CSE-2000-08-01, University of Washington*, 2000.

[52] Rishi Khan, Yujing Zeng, Javier Garcia-Frias, and Guang Gao. A bayesian modeling framework for genetic regulation. In *Proceedings of the IEEE Computer Society Bioinformatics Conference*, 2002.

[53] Seungchan Kim, Edward R. Dougherty, Michael Bittner, Yidong Chen, Krishnamoorty Sivakumar, Paul Meltzer, and Jeffry M. Trent. General nonlinear framework for the analysis of gene interaction via multivariate expression arrays. *Journal of Biomedical Optics*, 5(4):411–424, October 2000.

[54] Seungchan Kim, Huai Li, Edward R. Dougherty, Nanwei Cao, Yidong Chen, Michael Bittner, and Edward B. Suh. Can markov chain models mimic biological regulation. *Journal of Biological Systems*, 10(4):337–357, 2002.

[55] Sunyong Kim, Seiya Imoto, and Satoru Miyano. Dynamic bayesian network and nonparametric regression for nonlinear modeling of gene networks from

time series gene expression data. In *Int. W. on Computational Methods in Systems Biology (CMSB)*, 2003.

[56] John R. Koza, William Mydlowec, Guido Lanzo, Jessen Yu, and Martin A. Keane. Reverse engineering of metabolic pathways from observed data using genetic programming. In *PSB'01*, pages 434–445, 2001.

[57] Lukasz Kurgan and Krzysztof J. Cios. CAIM discretization algorithm. *IEEE TKDE*, 2004.

[58] Harri Lahdesmaki, Ilya Shmulevich, and Olli Yli-Harja. On learning gene regulatory networks under the boolean network model. *Machine Learning*, 52:147–167, 2003.

[59] Marco Laumanns. Analysis and apllication of evolutionary multiobjective optimization algorithms. *PhD Thesis, Swiss Federal Institute of Technology, Zurich*, 2003.

[60] Tong Ihn Lee, Nicola J. Rinaldi, and Francois Robert. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298:799–804, October 2002.

[61] Shoudan Liang, Stefanie Fuhrman, and Roland Somogyi. REVEAL: A general reverse engineering algorithm for inference of genetic network architectures. In *Pacific Symposium on Biocomputing*, volume 3, pages 18–29, 1998.

[62] Huan Liu, Farhad Hussain, Chew Lim Tam, and Manoranjan Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6:393–423, 2002.

[63] Philip M. Long and Vinsensius Berlian Vega. Boosting and microarray data. *Machine Learning*, 52:31–44, 2003.

[64] Schena M., D. Shalon, R. Heller, A. Chai, P.O. Brown, and R.W. Davis. Parallel human genome analysis: Microarray-based expression monitoring of 1000 genes. In *Proc. Natl Acad. Sci.*, pages 10614–10619, 1996.

[65] Hiroshi Matsuno, Atsushi Doi, Masao Nagasaki, and Satoru Miyano. Hybrid petri net representation of gene regulatory network. In *PSB'02*, pages 341–352, 2000.

[66] Tom M. Mitchel. *Machine Learning*. McGRAW-Hill, 1997.

[67] Kevin Murphy. Dynamic bayesian networks: Representation, inference and learning. *PhD Thesis, University of California, Berkeley*, 2002.

[68] Kevin Murphy and Saira Mian. Modelling gene expression data using dynamic bayesian networks. *Technical Report, CS Division, University of California, Berkeley*, 1999.

[69] Lucila Ohno-Machado, Staal Vinterboo, and Griffin Weber. Classification of gene expression data using fuzzy logic. *Journal of Intelligent and Fuzzy Systems*, 12(1):19–24, 2002.

[70] Dana Peer, Aviv Regev, Gal Elian, and Nir Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17:S215–S224, 2001.

[71] Audrey P.Gasch, Paul T. Spellman, Camilla M. Kao, Orna Carmel-Harel, Michael B. Eisen, Gisela Storz, David Botstein, and Patrick O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*, 11:4241–4257, 2000.

[72] Gavin Adrian Rummery. Problem solving with reinforcement learning. *PhD thesis, Cambridge University*, 1995.

[73] Stuart J. Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[74] Dudoit S., Yee Hwa Yang, Matt Callow, and Terry Speed. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Technical Report #578, University of California, Berkeley*, 2000.

[75] Kauffman S.A. The origins of order: Self organization and selection in evolution. *Oxford University Press, New York*, 1993.

[76] Guy Shani, Ronen I. Brafman, and David Heckerman. An MDP-based recommender system. In *UAI'02*, 2002.

[77] Gavin Sherlock, Tina Hernandez-Boussard, Andrew Kasarskis, Gail Binkley, John C. Matese, Selina S. Dwight, Miroslava Kaloper, Shuai Weng, Heng Jin, Catherine A. Ball, Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, David Botstein, and J. Michael Cherry. The stanford microarray database. *Nucleic Acids Research*, 29(1):152–155, 2001.

[78] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim, and Wei Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

[79] Ilya Shmulevich, Edward R. Dougherty, and Wei Zhang. Control of stationary behavior in probabilistic boolean networks by means of structural intervention. *Biological Systems*, 10:431–446, 2002.

[80] Ilya Shmulevich, Edward R. Dougherty, and Wei Zhang. Gene perturbation and intervention in probabilistic boolean networks. *Bioinformatics*, 18(10):1319–1331, 2002.

[81] Ilya Shmulevich, Olli Yli-Harja, and Jaakko Astola. Inference of genetic regulatory networks under the best-fit extension paradigm. *Computational and Statistical Approaches to Genomics, Kluwer Academic Publishers*, 2002.

[82] Adrian Silvescu and Vasant Honavar. Temporal boolean network models of genetic networks and their inference from gene expression time series. *Complex Systems*, 13:54–70, 2001.

[83] Paul T. Spellman, Gavin Sherlock, Michael Q. Zhang, Viswanath R. Iyer, Kirk Anders, Michael B. Eisen, Patrick O. Brown, David Botstein, and Bruce Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.

[84] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* MIT Press, 1998.

[85] Speed T., Dudoit S., and Fridlyand J. Comparison of discrimination methods for the classification of tumors using gene expression data. *Technical Report #576, University of California, Berkeley*, 2000.

[86] Jesper Tegner, Stephen Yeung, Jeff Hasty, and James Collins. Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling. *PNAS*, 100(10):5944–5949, 2003.

[87] Tianhai Tian and Kevin Burrage. Stochastic neural network models gene regulatory networks. In *Proc. of the 2003 IEEE Congress on Evolutionary Computation*, 2003.

[88] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[89] Eugene P. van Someron, L.F.A. Wessels, and M.J.T. Reinders. Genetic network models: A comparative study. In *Proceedings of SPIE*, 2001.

[90] Hines W. W., Montgomery D.C., Goldsman D.M., and Borror C.M. *Probability and Statistics in Engineering*. Wiley and Sons, 2003.

[91] D.C. Weaver, C.T. Workman, and G.D. Stormo. Modeling regulatory networks with weight matrices. In *PSB'99*, 1999.

[92] Ann S. Wilson, Bridget G. Hobbs, Terence P. Speed, and P.Elizabeth Rakoczy. The microarray : potential applications for ophthalmic research. *Molecular Vision*, 8:259–270, 2002.

[93] Peter J. Woolf and Yixin Wang. A fuzzy logic approach to analyzing gene expression data. *Physiol Genomics*, 3:9–15, 2000.

[94] Yee Hwa Yang, Sandrine Dudoit, Percy Luu, and Terence P. Speed. Normalization for cDNA microarray data. *Nucleid Acids Research*, 30(4), 2002.

[95] C. Yoo, V. Thorsson, and G.F. Cooper. Discovery of cuasal relationships in a gene regulation pathway from a mixture of experimental and observational DNA microarray data. In *PSB'02*, pages 498–509, 2002.

VITA

Osman Abul was born in Kadınhanı/Konya on June 15, 1974. He completed his primary schools in Toprakkale/Osmaniye and high school in Çanakkale (at Çanakkale Lisesi). He received his B.S. degree in Computer Science and Engineering from the Hacettepe University in June 1996 and M.S. degree in Computer Engineering from the Middle East Technical University in August 1999. He worked as a research/teaching assistant in the Department of Computer Engineering of Hacettepe University from August 1996 to August 1997. Between August 1997 and June 2001 he worked as a software engineer in the Microwave and System Technologies division of Aselsan inc, Ankara. He completed his compulsory military service between August 2001 and November 2002. Between December 2002 and December 2003, he was a visiting PhD student in the Computer Science Department of University of Calgary, Canada, where he also served as a research associate. Since January 2004, he has been affiliated with Aselsan, serving as a software engineer.