APPLICATION OF LOGIC SYNTHESIS TOWARD THE INFERENCE AND CONTROL OF GENE REGULATORY NETWORKS

A Dissertation

by

PEY-CHANG KENT LIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Sunil P. Khatri
Committee Members,	Edward Dougherty
	Paul Gratz
	Tiffani Williams
	Gabor Balazsi
Head of Department,	Chanan Singh

August 2013

Major Subject: Electrical Engineering

ABSTRACT

In the quest to understand cell behavior and cure genetic diseases such as cancer, the fundamental approach being taken is undergoing a gradual change. It is becoming more acceptable to view these diseases as an engineering problem, and systems engineering approaches are being deployed to tackle genetic diseases. In this light, we believe that logic synthesis techniques can play a very important role. Several techniques from the field of logic synthesis can be adapted to assist in the arguably huge effort of modeling cell behavior, inferring biological networks, and controlling genetic diseases. Genes interact with other genes in a Gene Regulatory Network (GRN) and can be modeled as a Boolean Network (BN) or equivalently as a Finite State Machine (FSM). As the expression of genes determine cell behavior, important problems include (i) inferring the GRN from observed gene expression data from biological measurements, and (ii) using the inferred GRN to explain how genetic diseases occur and determine the ”best” therapy towards treatment of disease.

We report results on the application of logic synthesis techniques that we have developed to address both these problems. In the first technique, we present Boolean Satisfiability (SAT) based approaches to infer the predictor (logical support) of each gene that regulates melanoma, using gene expression data from patients who are suffering from the disease. From the output of such a tool, biologists can construct targeted experiments to understand the logic functions that regulate a particular target gene. Our second technique builds upon the first, in which we use a logic synthesis technique, implemented using SAT, to determine gene regulating functions for predictors and gene expression data. This technique determines a BN (or family of BNs) to describe the GRN and is validated on a synthetic network and the p53 network. The first two techniques assume binary valued gene expression data. In the third technique, we utilize continuous (analog) expression data, and present an algorithm to infer and rank predictors using modified Zhegalkin polynomials. We demonstrate our method to rank predictors for genes in the mutated mammalian and melanoma networks. The final technique assumes that the GRN is known, and uses weighted partial Max-SAT (WPMS) towards cancer therapy. In this technique, the GRN is assumed to be known. Cancer is modeled using a stuck-at fault model, and ATPG techniques are used to characterize genes leading to cancer and select drugs to treat cancer. To steer the GRN state towards a desirable healthy state, the optimal selection of drugs is formulated using WPMS. Our techniques can be used to find a set of drugs with the least side-effects, and is demonstrated in the context of growth factor pathways for colon cancer.

To my family

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

I-A. Genomics

Recently, there have been many advances in biology towards our understanding of the human genome. Improvements in DNA/RNA sequencing have allowed rapid and inexpensive sequencing of a person's genome and improvements in microarray technology have allowed biologists and clinicians to rapidly measure tens of thousands of gene expressions at once. These advances have brought new interest to the field of genomics, which aims to study genes as a collective system in a Gene Regulatory Network (GRN), rather than to study genes individually. Genomics is important to biology and medicine as both cellular control and its failure – disease – is a result of the activity of many genes interacting simultaneously.

Towards the treatment of genetic diseases, genomics has three main goals.

1. Understand how cells operate and the way the cellular system fails
2. Identify key genes for specific diseases
3. Use models to guide drug development and therapy for such diseases

There has been recent work in the genomics area from various researchers in the signal processing, computational biology, and data-mining communities, in addition to the work of biologists and medical practitioners. We recognize that genomics, or the system of gene interactions, can be modeled as a Finite State Machine (FSM) in logic-speak, and thus is amenable to many powerful logic synthesis techniques. The motivation for our research is to determine how logic synthesis can be used in inferring and controlling the GRN, and to increase the interest among the logic and CAD community towards the study of genomics.

I-B. Cell Biology

In this section, we present an engineering-centric view of the biological organism, with an overview of some of the relevant terminology and domain information.

I-B.1. Genome

In an organism, the basic unit of life is the cell. Practically all cell function is carried out by large molecules called *proteins*. There exist many types of proteins, and they provide most of the cell structure and cell function. Some examples of proteins are enzymes to promote chemical reactions, signaling molecules for communication across cells, and molecules with moving parts [1, 2]. Proteins can have complex shapes, allowing for many functions. Each protein is made up of a chain of amino acids as determined by its corresponding gene, and the shape is determined by its amino acid sequence [1, 2]. The unique shape of a protein allows it to chemically bind to other molecules, including other proteins, that match its specific shape.

The genetic information of each living organism is encoded in DNA (Deoxyribonucleic acid). DNA is a molecule consisting of a sequence of 4 nucleotide bases: adenine, guanine, cytosine, and thymine (often shortened to the letters A, G, C, and T respectively) [3]. The actual sequence of the bases is the property that encodes the genetic information. The structure of DNA consists of two strands, each providing a copy of the sequence. The two strands run in parallel and are connected to each other through base pairing, wherein each base on one strand bonds with only one type of base on the other strand. There are two types of base pairs, A-T and G-C. An example of the DNA structure is shown in Figure I.1.

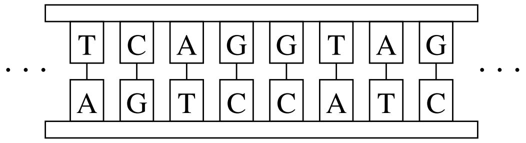


Fig. I.1. DNA structure

The DNA can be visualized as a string of characters, where each character is one of the 4 nucelotide bases. *Genes* are short stretches (chunks) of *DNA* (Figure I.2) that produce functional molecules proteins and RNA. The linear sequence of bases in a gene spells out the sequence of amino acids in a protein.

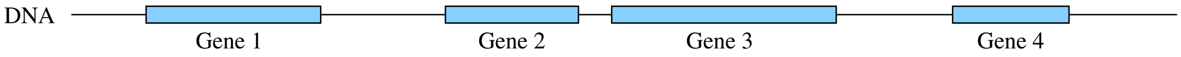


Fig. I.2. Genes consist of short stretches of DNA

When taken as a whole, the complete set of information in an organism's DNA is called the genome. Individual "instances" of the same species have small variations in their genome (which result in variations in characteristics of the human being for instance). The entire genomes of several model organisms have been *sequenced*, yielding the entire DNA sequence for those organisms. For instance, the human genome has been sequenced and has been found to consist of approximately 3.2 billion base pairs in all and around 30,000 genes.

While DNA is comprised of 4 bases, proteins are an amino acid chain with 20 possible amino acids. The process of mapping a 4-letter alphabet (DNA) to 20-letter alphabet (amino acid) takes place with the help of *RNA* in a process called transcription and translation (Figure I.3).

When a protein is needed by the cell, the nucleotide sequence of the gene is first copied to another type of nucleic acid, RNA, which is similar to DNA but with the 4 nucleotide bases: A, G, C, U. The RNA strand then serves as a template for protein synthesis. A specific molecule called polymerase latches onto the start site of the gene and slides along the DNA, synthesizing the complementary RNA at the same time. This process of copying gene DNA into RNA strands is referred to as *transcription*. When a gene is being transcribed, it is said to be *expressed*, or turned ON. If no transcription is taking place, then the gene is said to be not expressed, or turned OFF.

After the RNA strand is produced, the RNA nucleotide sequence has to be decoded to produce the appropriate protein. This *translation* process takes place with the use of other functional molecules called ribosomes among others, which read the RNA strand and bind the complementary amino acids to form a chain. The resulting amino acid chain folds to create the final protein.

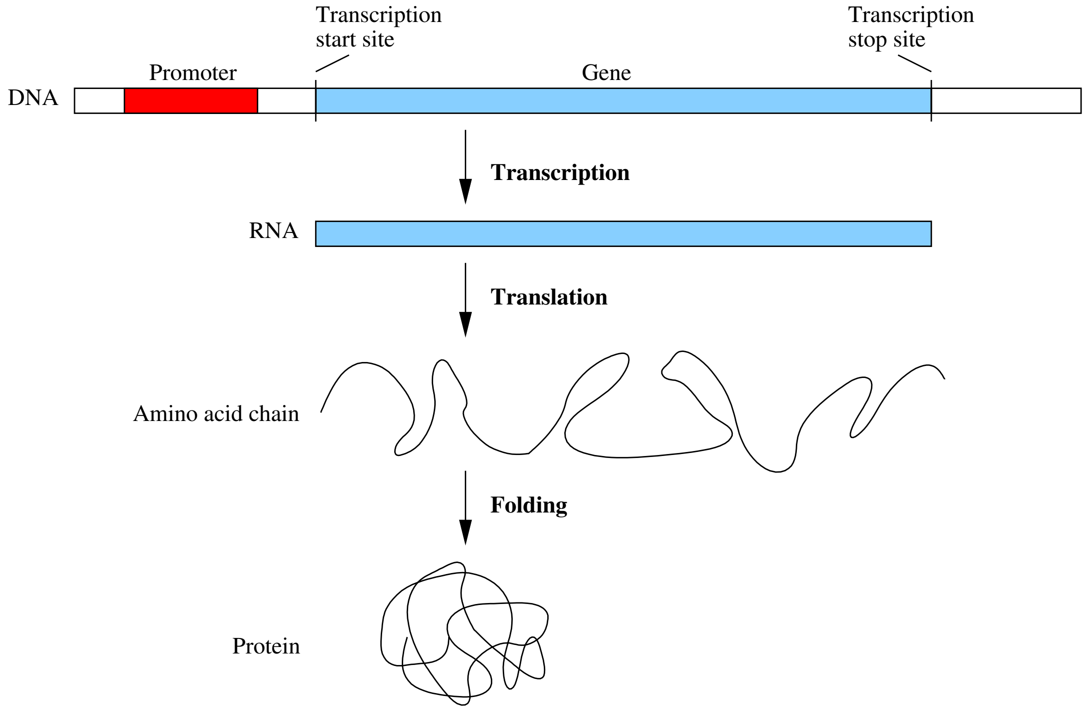


Fig. I.3. Transcription and translation of gene to RNA and protein

I-B.2. Gene Expression Regulation

While the genome of an organism encodes all functional molecules that are needed to make and maintain its cells, not every gene needs to be expressed all the time. A cell can regulate its genes and use its genes selectively, switching genes ON and OFF to produce different proteins depending on the situation. Or in the case of multicellular organism, all cells have the same genome and different genes can be expressed to create large variety of cell types (i.e. skin cells, muscle cells, colon cells, etc.). One example to demonstrate that the alteration of expression of single gene can trigger development of a different cell is the study involving fruit flies and gene *Ey* [4], which is crucial for eye development. In this study, *Ey* is expressed early in development (using artificial means) in cells that normally go on to form legs. As a result, in these flies eyes developed in the middle of legs. Another example of single gene expression affecting cell function is the β-globin gene, which produces one of the hemo protein. Mutations in the β-globin gene [5] cause the protein to have the wrong amino acid sequence and hence, different physical dimension. When this protein binds with the other hemo protein groups, the resulting hemoglobin does not have the correct shape to transport oxygen, leading to the disease sickle cell anemia. Both these examples show how erroneous gene expression can affect normal cell operation and disease.

Protein production can be controlled at different points throughout the transcription, translation, and protein binding processes. Of interest to gene regulation and genomics is the first type, transcription control. Each gene has a start site that indicates where transcription will start. Upstream of the start site on the DNA is the promoter region (regulatory DNA sequences as shown on Figure I.3) which are needed by the cell to switch the gene ON or OFF [6]. These regulatory DNA sequences must be bound by gene regulatory proteins which uniquely recognize these sequences. The gene regulatory proteins, when bounded, can either suppress or enhance transcription. Those proteins which turn OFF genes are called *repressors*, which proteins that turn ON genes are *activators*. For example, when the a repressor binds to the gene regulatory sequence, the polymerase molecule cannot attach to the starting site, thus transcription cannot begin and the gene expression is turned off. As shown in Figure I.4, gene $G1$ produces protein $P1$, which is a repressor for gene $G2$. So if $G1$ is expressed ($P1$ is present), then gene $G2$ cannot produce protein $P2$. Otherwise, if gene $G1$ is not expressed, then gene $G2$ can produce protein $P2$. In this manner, a complex gene expression network can be formed.

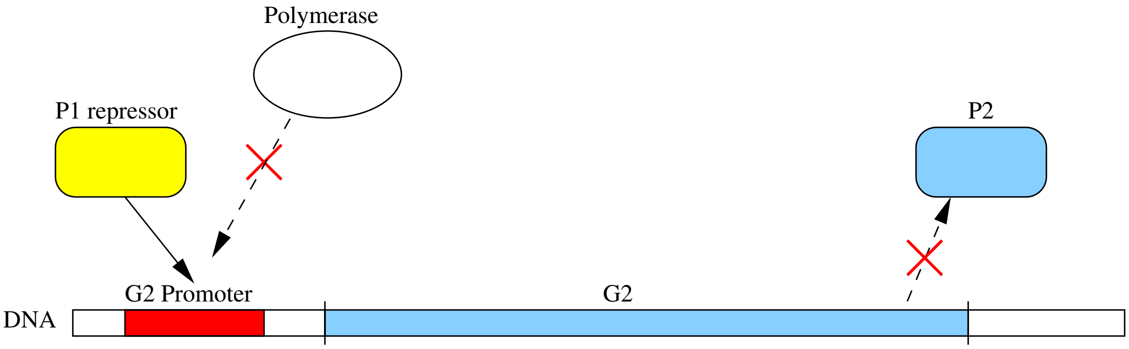


Fig. I.4. Gene expression repression example

I-B.3. Gene Expression Measurement

As described in the previous section, cell function and control results from the interaction of genes and its products: RNA and proteins. All three components, DNA (and genes), RNA, and proteins, are involved in gene expression regulation and have high level of interaction. This interaction allows a significant amount of information about the gene activity to be available in each of the components. In industry and research, the focus is measurement at the RNA level, from which the gene expression value can be inferred. In particular, high-throughput applications such as *expression microarrays* have been recently developed which allow for measurement of tens of thousands of RNA simultaneously.

The expression microarray system is comprised of both biochemical and optical imaging processes. A microarray is a slide plate with an array of thousands of different *probes* attached to the surface in a grid pattern. Each probe is a single-stranded DNA corresponding to a unique gene. The general steps (shown in Figure I.5) begin with extracting RNA from cells, converting the RNA to single stranded cDNA (complementary DNA), attaching fluorescent labels (markers) to the cDNAs, allowing the cDNAs to attach (bind) to their complementary probes on the slide, washing the slide of any unbounded molecules, and then detecting the fluorescence of the attached cDNA on the slide.

The principle of the microarray is that if a specific gene is expressed, then the corresponding RNA is produced. The RNA is converted to cDNA and the fluorescent-marked cDNA will attach to its complementary probe on the microarray. Those fluorescent-marked cDNA that attach to probes will fluoresce (emit light when excited by a laser) and the intensity of the fluorescence can be recorded as a digital image (for each probe on the microarray).

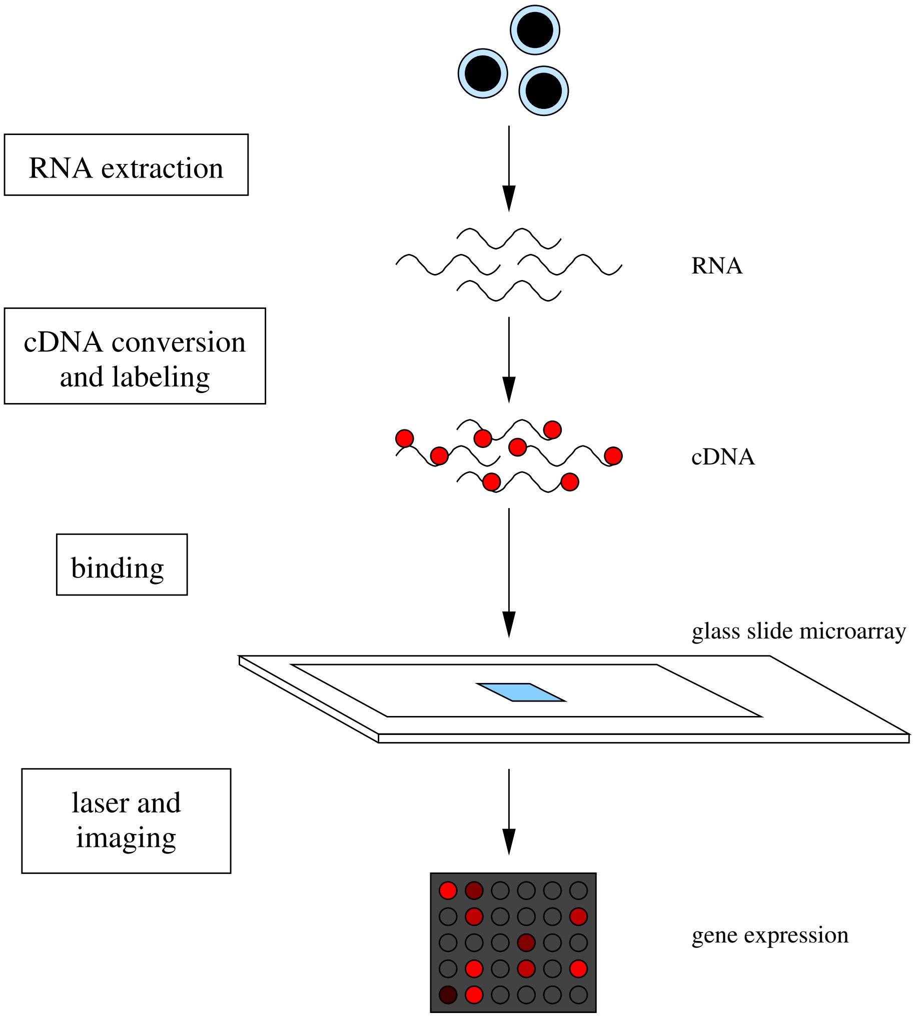


Fig. I.5. Microarray process flow for measuring gene expression

Through analysis of the digital image, the intensities of fluorescence reflect RNA levels, and in turn gene expression levels. The measurements of gene expression from microarrays can be expressed in the form of ratios or as raw intensity values, which can be further processed with statistical software [7] to obtained normalized or binary expression values. The development of microarrays and other measuring technologies have allowed for snapshot measurements of the entire genome, driving research to focus on gene regulation in the complete network, rather than just gene pair interactions.

I-C. Gene Regulation Networks

A main focus of genomics is the understanding of the manner in which cells execute and control the number of operations required for normal cellular function, and the ways in which cell systems fail, causing disease. While classical approaches in molecular biology have identified specific processes and interactions in the cell, they have not been able to produce an overall formalism for cell operation. Many cell processes, functions, and diseases are a result of highly complex and multivariate gene interaction, necessitating a system or network view of the genome.

The gene regulatory network (GRN) is one systematic approach to characterize the cell behavior through gene-to-gene interaction among a set of genes (i.e. how the expression of a subset of genes affects the expression of another gene in the set). Several GRN models have been developed, but all models have the same properties, in that they all represent systems which characterize an interaction among a group of components as a whole, and they all model a dynamical, time-varying physical process.

In particular, a GRN model describes the 1) topology (connectivity structure) of the genes, and 2) the regulating functions of the genes. Both these aspects together determine the dynamical behavior of system. With an accurate GRN model, an analysis of the topology and regulation functions can provide deep insight in the long-term behavior of the system, and identify how the system can fail and lead to disease. Several models have been proposed for the GRN such as Markov Chains [8, 9], Differential equations [10, 11], Boolean Networks (BNs) [12, 13], Continuous Networks [14], and Stochastic Gene Networks [15]. Our research focuses on Boolean networks due to its significant adoption by the research community. A benefit of using BNs is that they lend themselves to analysis using logic synthesis techniques. Boolean networks are described in detail in the following subsection.

In addition to gene regulatory networks, RNA and protein regulatory networks are also studied in genomics. RNA regulatory networks [16, 17] describe RNA interactions such as splicing. After transcription, the RNA may undergo splicing where portions of the RNA are kept, while other portions are removed. Some RNA have alternative splicing, which allows RNA to produce different proteins. On the other hand, protein regulatory networks [18] describe protein-protein interactions such as protein binding (for example, hemo protein groups) or altering protein activity. Protein networks are also important to study as most proteins perform various cell functions through interactions with other proteins.

I-C.1. Boolean Network

We utilize the *Boolean Network* (BN) model that was proposed by Kauffman in 1969 [12]. In a Boolean Network, the expression activity of a gene is represented as a binary value, where 1 indicates the gene is ON (expressed) and producing gene-products, while 0 indicates it is OFF (not expressed). Such a model cannot capture the continuous and stochastic biochemical properties of protein and RNA production. However, it has been observed that genes can typically be modeled as ON or OFF in any particular biochemical pathway [19].

A Boolean network is formally defined as a set of nodes $\{x_1, x_2, \ldots, x_n\}$ with Boolean functions $\{g_1, g_2, \ldots, g_n\}$. In the context of genomics, each node x_i is a gene, and each gene is associated with a logic function g_i. The value of a gene is a binary variable, $x_i \in \{0, 1\}$, and is updated at the next time point $t+1$ according to its associated function $g_i()$ and the value of the genes $(x_1, x_2, \ldots, x_n)$ at the current time point t. The state of a gene x_i represents the expression of the gene, where $x_i = 1$ indicates expressed, and $x_i = 0$ indicates not expressed. In the BN, all genes are assumed to updated synchronously, at each time step.

In general, each function $g_i()$ depends on a subset of genes $s_i \subseteq (x_1, x_2, \ldots, x_n)$. In this sense, this subset of genes determine or ”predict” the expression of a target gene x_i. The subset of genes s_i is referred to as a *predictor* for gene i. In essence, a predictor describes which genes directly interact with each other. The complete set of all predictors in the GRN (for each of the genes in the GRN) is the *predictor set*. The complete set of functions for each gene in the GRN in turn determine the complete dynamic behavior of the GRN. The predictor set of the GRN determines the structure or topology of the GRN.

Naturally, the Boolean network describes a dynamic system. The expression values of all the genes $(x_1, x_2, \ldots, x_n)$ at a particular time t is the state in the network at t. At the next time point $t+1$, the network transitions to a new state as determined by the functions and expressions of the genes at the current time step. For a BN with n genes, there are 2^n total states, and the behavior of the BN can be described in a state transition table (truth table) or a state transition diagram. The long-term behavior of the BN is such that absent any external input and given any starting state, the network repeatedly visits a fixed sequence of state(s), forming a cycle. The states in such a cycle are called *attractor states*, while the cycle is called an *attractor cycle*. Since the BN is deterministic, after it has reached an attractor state, it will stay in the attractor cycle absent any external perturbation.

We present a small example of a Boolean network. In this example, there are 4 genes (x_1, x_2, x_3, x_4) with the corresponding functions listed in Table I.1.

Gene	Regulating Function
x_1	$g_1 = (x_3 \oplus x_4)$
x_2	$g_2 = \overline{x_4}$
x_3	$g_3 = \overline{(x_1 x_2 x_4)}$
x_4	$g_4 = x_1 + x_2 + x_3$

Table I.1. Boolean Regulating Functions for Example 4-Gene Network

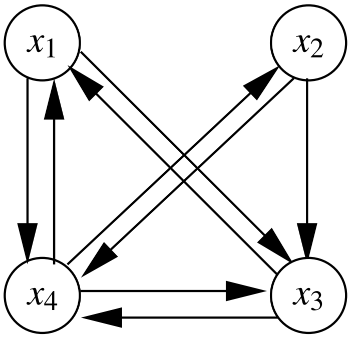


Fig. I.6. Example BN topology (each node represents a gene)

From the functions in Table I.1, we can determine the topology (Figure I.6) and predictors of the BN. For example, the expression of gene x_1 is predicted by genes x_3 and x_4. Similarly, gene x_2 is predicted by gene x_4, and so on. The state space is $[x_1, x_2, x_3, x_4]$ with $2^4 = 16$ total states in the BN.

If at any time t, the BN has a state x^t, then we can apply the gene regulating function on x^t (current state) to obtain x^{t+1} (next state). For example if the current state of the BN is $< x_1, x_2, x_3, x_4 >= 1010$, then in the next time instant, $x_1^{t+1} = (x_3^t \oplus x_4^t) = (1 \oplus 0) = 1$, $x_2^{t+1} = \overline{x_4^t} = \overline{0} = 1$, $x_3^{t+1} = \overline{(x_1^t x_2^t x_4^t)} = \overline{(1 \cdot 0 \cdot 0)} = 1$, and $x_4^{t+1} = x_1^t + x_2^t + x_3^t = 1 + 0 + 1 = 1$. In this way, by enumerating over all the values of $< x_1, x_2, x_3, x_4 >$ in the state space, we obtain $< x_1, x_2, x_3, x_4 >^{t+1}$ and can populate the state transition table (Table I.2) and state transition diagram (Figure I.7). From these we find the BN has two attractor cycles: a 3 state attractor cycle $(1111) \rightarrow (0001) \rightarrow (1010)$ and a 1 state (singleton) attractor cycle (0011)

Current state				Next state			
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
0	0	0	0	0	1	1	0
0	0	0	1	1	0	1	0
0	0	1	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	0	0	1	1	1
0	1	0	1	1	0	1	1
0	1	1	0	1	1	1	1
0	1	1	1	0	0	1	1
1	0	0	0	0	1	1	1
1	0	0	1	1	0	1	1
1	0	1	0	1	1	1	1
1	0	1	1	0	0	1	1
1	1	0	0	0	1	1	1
1	1	0	1	1	0	0	1
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	1

Table I.2. Example 4-Gene State Transition Table

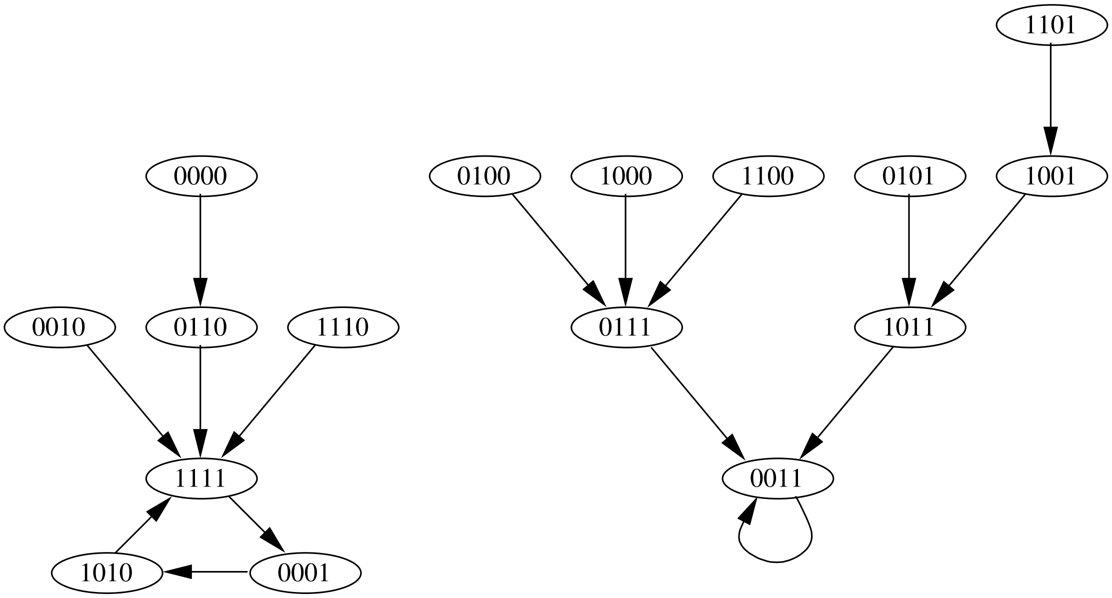


Fig. I.7. Example 4-gene state transition diagram

In logic-speak, the Boolean network introduced by Kauffman is equivalent to a Finite State Machine (FSM). In an FSM, the machine can be in one state at a time (its *current state*), which is stored in memory elements such as registers or flip-flops. The FSM can change to another state (the *next state*), according to combinational logic function which depends on its current state and inputs. These combinatorial logic functions are exemplified by the gene regulating functions in Table I.1. Figure I.8 shows a block diagram of a typical FSM. It can be seen that memory elements storing state at a particular time are the same as the gene expression x_i^t, and the combinational (next state) logic is the same as the gene logic function g_i, which operates on x^t to produce x_i^{t+1} (i.e. $g_i(x^t) = x_i^{t+1}$).

Similarly, the predictor of a gene is equivalent to the logical support of (gene regulating) function. By modeling the GRN as a BN (FSM), this allows a rich set of logic synthesis algorithms to be brought to bear to the problem of GRN inference and control.

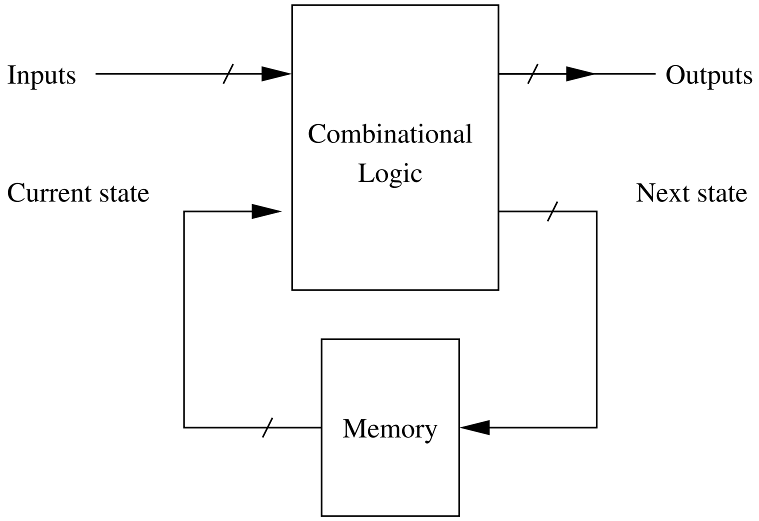


Fig. I.8. Block diagram of finite state machine

I-D. Genomics Overview

Before introducing some key ideas in logic synthesis, we first present an overall view of genomics and its key issues. In genomics, an accurate representation of the GRN is necessary for understanding how genetic diseases occur and how we can treat these diseases. Over the lifetime of an organism, mutations can occur in the genome, causing changes in its normal biological behavior. Mutations are caused several environmental factors (radiation, drugs, etc.). Once a genome is mutated, the GRN behavior may deviate from that of a healthy organism. Cancer and gene-related diseases are often the result of a failure in the gene signaling mechanisms, leading to incorrect gene regulation and its associated functions. With a GRN-based view of the biological functioning of the organism, we can potentially target specific gene(s) for drugs, and hence modify their genetic expression, thereby treating the disease. This is a promising way to treat genetic diseases, and can yield the possibility of ”personalized medicine” - targeted and specific disease prevention and treatment based on an individual’s genetic information [20, 21].

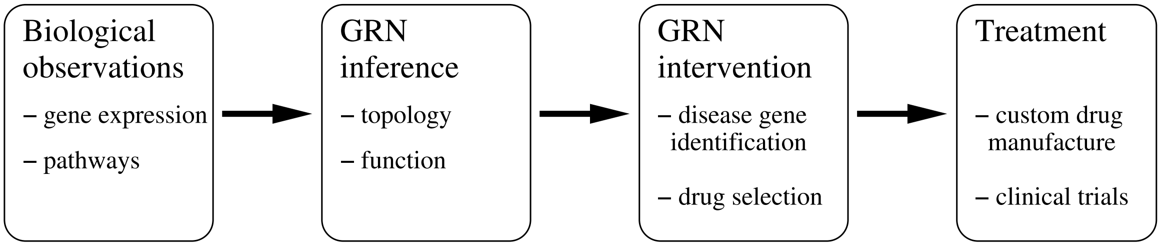


Fig. I.9. Overview of genomics research areas

We can divide genomics into four main areas, biological observations, GRN inference, GRN intervention, and treatment, as outlined in Figured I.9. Biological observations refer to measurement of the GRN including gene expression measurements, pathways, sequencing, biopsy, and so on. GRN inference aims to infer a model of the GRN from observations of the biological system, for example, inferring gene predictors or topology, or gene regulating function from binary valued or continuous gene expression measurements. The GRN must describe both the topology and interaction of the genes in the network. The topology refers to the connectivity structure of the genes (predictors and predictor set). The interaction describes the dynamical behavior or regulating function of all genes. GRN intervention analyzes the inferred GRN model to determine which genes can fail, leading to disease, and how to intervene in the GRN to treat disease. Treatment applies the intervention results to drive new drug development and direct clinical studies. The research presented in this thesis focuses on GRN inference and GRN intervention.

The task of inferring a GRN is an arduous one. Because of the complex interaction of the genes, it is hard if not impossible to construct a single biological experiment that will yield the complete GRN. Instead, several steps are employed. First, from biological measurements such as expression microarrays, biologists statistically observe that a certain subset of genes G are involved in the growth and spread of a genetic disease. Multiple samples of the gene expression of the genes in G (for several diseased and healthy individuals) are taken for comparison. These can also be in the form of *time course* data (where expression of the genes in G are taken for the same individual, over a sufficiently long duration). Time course data is generally not readily available, however. From the gene expression data, logic techniques can be utilized to i) find the support or predictors for each gene $g_i \in G$, and ii) infer the function of the GRN. To validate the GRN obtained in this manner, biologist can perform targeted experiments to verify specific gene interactions within the GRN. Often, pathways (or portions) of the GRN are known, from targeted experiments that have already been conducted by biologists in the past. By curating the results of several such (often independently conducted) experiments, some GRNs have been inferred with reasonable confidence. This information can be used to verify results, or used as additional inputs to constrain the state space of our logic synthesis methods.

Once the GRN for a genetic disease is known, one main area of interest is to understand how genetic disease arises from the GRN and how to intervene in the GRN to treat diseases. In the case of genetic diseases, genes are mutated or damaged leading to signaling failure in the GRN. The problem then becomes how to identify which genes are responsible for a particular disease and how to design drugs to correct the behavior of these genes. If the specific effect of candidate drugs on particular genes is known, another problem of interest is to find the best set of drugs which correct the GRN behavior of a diseased organism. Both problems can be cast as instances of another logic synthesis method called automatic test pattern generation (ATPG).

I-E. Logic Synthesis

The Boolean Network [12] model for GRNs is a finite state machine (FSM), which provides the motivation for applying established and efficient techniques and algorithms from the field of logic synthesis. In computer circuit design, logic synthesis is the process of converting a high-level specification into an optimized logic gate representation. Logic synthesis techniques can be further split into methods, which include simplification of the logic, mapping the logic to specific technology or libraries, timing optimization, and testing or verification of the design. Logic synthesis is an essential step in the process of digital integrated circuit (IC) design, given the ever increasing complexity of modern digital ICs. For example, state of the art micro-processors comprise millions of gates, and their design is made possible only with the help of computer-aided design (CAD) tools [1], which include logic synthesis tools as well.

Logic synthesis techniques can take gene expression observations as inputs and infer and construct the GRN as a Boolean network (logic circuit), which represents the $g_i(x)$ functions of all the genes of the GRN. The resulting circuit can be simulated using logic synthesis tools to query the long term behavior of the GRN and analyze its attractor cycles. This thesis employs several logic synthesis techniques, all of which are based on Boolean Satisfiability (SAT). While SAT is an NP-complete problem, many Boolean logic problems translate naturally to SAT. Furthermore, SAT is heavily used in logic synthesis algorithms and in the EDA industry, and as such there are many efficient SAT solvers that are available for public download. In the remainder of this chapter, we first present an overview of logic synthesis and SAT, and later describe the operation of SAT solvers.

I-E.1. Logic Functions and Representation

Definition I.1: Suppose that $B = \{0,1\}$. The **Boolean *n*-cube** B^n is a representation of the n-dimensional hyperspace constructed by combining n instances of B.

Some examples of Boolean n-cubes are shown below and in Figure I.10.

[1] Computer-Aided Design (CAD) tools for IC design are also often referred to as Electronic Design Automation (EDA) tools

$$B^1 = B = \{0,1\}$$

$$B^2 = \{0,1\}X\{0,1\} = \{00,01,10,11\}$$

$$B^3 = \{0,1\}X\{0,1\}X\{0,1\} = \{000,001,010,011,100,101,110,111\}$$

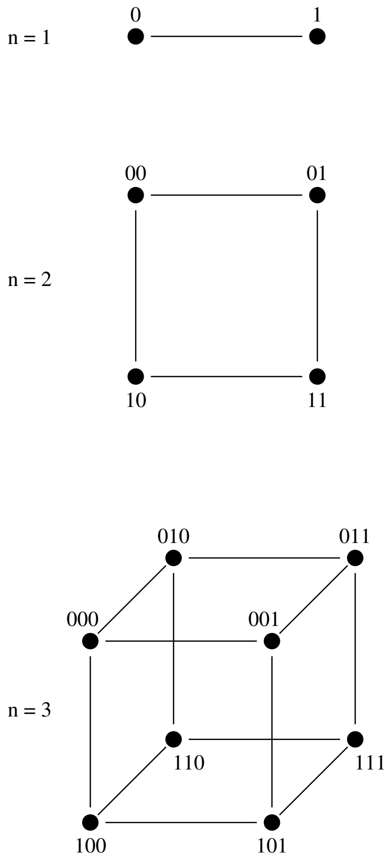


Fig. I.10. Boolean n-cube B^n

Definition I.2: A **Boolean function** f is a mapping $f(x_1, x_2, \ldots, x_n) : B^n \to B$. In other words, f maps each vertex of the Boolean n-cube (B^n) to 0 or 1.

There are several types of representations for Boolean functions. Some common representation used in logic synthesis include the Boolean n-cube (where each vertex is mapped to a 0 or 1 value), the truth table, the Karnaugh map, and the Boolean formula.

Definition I.3: A **truth table** of a function $f(x) : B^n \rightarrow B$ is a table of the values of each the 2^n vertices of B^n. Each vertex of the Boolean n-cube appears in a row of the truth table. The truth table comprises of columns (one for each input variable) and a column for the output. The input columns of the the truth table represent the vertex of B^n. If the output column is a '1', the row is referred to as a *minterm*, and if the output column is a '0', the row is called a *maxterm*. An example of truth table for a 2-input (x, y) AND function is shown in Table I.3, where only the vertex $\{1, 1\}$ is assigned to a '1', value and all other vertices are assigned '0'.

x	y	f_{AND}	
0	0	0	maxterm
0	1	0	maxterm
1	0	0	maxterm
1	1	1	minterm

Table I.3. Truth Table for AND Function f_{AND} with 2 Input Variables

Definition I.4: A **Karnaugh map** (K-map) is a graphical representation of the truth table of a function. In a K-map, each square represents a vertex v of B^n, and the contents of any vertex is the value. An example of a K-map is shown in Figure I.11. The K-map in this figure is that of the AND function with two input variables.

		x	
		0	1
y	0	0	0
	1	0	1

Fig. I.11. K-map for AND Function

Definition I.5: A **literal** or a **literal function** is a binary variable x or its negation $\overline{x}$. For example, $x_1, x_2, \ldots$ are variables, while $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots$ are literals.

Definition I.6: A **cube** or a **product** is a conjunction (AND) of literals. For example, the cube xy is a conjunction of two literals x and y. In the example $x_1 x_2 \overline{x_3}$, this cube is a conjunction of the literals x_1, x_2, and $\overline{x_3}$.

Definition I.7: A **clause** is a disjunction (logical OR) containing literals. For example, $(\overline{x_1} + x_3)$ is a clause with two literals $\overline{x_1}$ and x_3. Another example is $(x + \overline{y} + \overline{z})$ which is a clause with three literals x, $\overline{y}$, and $\overline{z}$.

Definition I.8: A **Sum of Products** (SOP) expression is a canonical representation of a function f, which consists of a disjunction (OR) of minterms. For example, the SOP of a 2-input AND gate is $f_{AND} = xy$.

Definition I.9: A **Product of Sums** (POS) expression is a canonical representation of a function f, which consists of a conjunction (AND) of maxterms. The POS is the dual of the SOP. For example, the POS of a 2-input AND gate is $f_{AND} = (x + y) \cdot (x + \overline{y}) \cdot (\overline{x} + y)$.

Definition I.10: A **Conjunctive Normal Form (CNF)** expression S consists of a conjunction (AND) of m clauses $c_1 \ldots c_m$. Each clause c_i consists of disjunction (OR) of k_i literals. POS is a form of CNF.

Definition I.11: A **Boolean formula** is a formula that represents a function. The formula is defined as catenations of:

- parentheses ( )

- literals $(x, y, \overline{x}, \overline{y})$
- Boolean operators ("$+$") OR, ("$\cdot$") AND
- complementation (example: $\overline{x+y}$)

Following are three examples of Boolean formulas.

$$f = x_1 \cdot \overline{x_2} + \overline{x_1} + x_2$$

$$g = (x_1 + x_2) \cdot (\overline{x_1} + \overline{x_2})$$

$$h = \overline{a} \cdot (\overline{b} + \overline{c})$$

Often, the AND operator "$\cdot$" is replaced by catenation, for example $a \cdot b$ is replaced by ab.

For any Boolean function, there are an infinite number of Boolean formulas. Note that in the previous examples of Boolean formulae above, both f and g are formulas that represent the same Boolean function. The goal of logic synthesis is to find the "best" Boolean formula for a function. In chip design, the utility function for logic synthesis may include minimizing number of gates (which may be selected from a standard gate (or cell) library) or minimizing the estimated circuit's delay or power.

I-E.2. Boolean Satisfiability

Boolean satisfiability (SAT) is an NP-complete decision problem. Given a Boolean formula in CNF form, SAT determines if there is an assignment of the variables that will satisfy the formula (make the formula evaluate true). Many algorithms in logic synthesis and electronic design automation (EDA) can be cast as an instance of Boolean satisfiability. Some examples where SAT is used in logic synthesis and EDA include functional equivalence checking, automatic test pattern generation (ATPG), and logic optimization. In addition, the industry and research community have developed several efficient and scalable SAT engines. In this thesis, we take advantage of these advancements in SAT solvers, and apply them towards problems in genomics. We first define basic terms in Boolean satisfiability.

Definition I.12: **Boolean satisfiability (SAT)**. Given a Boolean formula S (on a set of binary variables X) expressed in CNF, the objective of SAT is to identify an assignment of the binary variables in X that satisfies S, if such an assignment exists. If no such assignment exists, S is concluded to be *unsatisfiable* (UNSAT).

In order to satisfy the formula S (i.e. make it evaluate to *true*), each clause of S must have at least one literal evaluate to true. Satisfying S is equivalent to satisfying all $c_i \in S$. In general, there may exist many satisfying assignments for the formula in question.

For example, consider the formula:

$$S(a,b,c) = (\overline{a}+\overline{b}) \cdot (a+b+c)$$

This formula consists of 3 variables, 2 clauses, and 5 literals. To determine if the formula is satisfiable, we attempt to satisfy all the individual clauses. If all clauses are satisfied, then the formula S is also satisfied. We observe that the first clause $(\overline{a}+\overline{b})$ is satisfied (or evaluates to 1) if either $a = 0$ or $b = 0$. In that case, $\overline{a} = 1$ or $\overline{b} = 1$ respectively, which satisfies the first clause. If $a = 0$ and $b = 0$, the second clause $(a+b+c)$ is satisfied only if $c = 1$.

Because all its clauses are satisfied, we conclude that S is satisfiable, and a *satisfying assignment* is $(a,b,c) = (0,0,1)$ or $\overline{a}\overline{b}c$. Note, the cube (product) $\overline{a}\overline{b}c$ is logically the same as stating that $a = 0, b = 0$, and $c = 1$.

An extension of the SAT problem, in which the goal is to find *all* satisfying assignments is called *All-SAT*.

Definition I.13: **All-SAT**. Given a Boolean formula S (on a set of binary variables X) expressed in CNF, the objective of All-SAT is to find all assignments of the binary variables in X that satisfies S, if such an assignment exists.

One simple algorithm for All-SAT is to perform SAT on the formula S, express the satisfying assignment as a cube k, complement k to get a clause c, add c as a new clause of the formula S, and perform SAT again repeatedly until an UNSAT result is obtained. The inclusion of c in S ensures that the same cube k cannot be found as a satisfying assignment again. The process continues until no new solutions can be found.

To demonstrate All-SAT, we refer back to the previous example, $S(a,b,c) = (\overline{a}+\overline{b}) \cdot (a+b+c)$ where we found a satisfying cube $k = \overline{a}\overline{b}c$. We perform an All-SAT by first taking the satisfying cube and complementing it (using DeMorgan's law) to form a new clause c to be added to S:

$$c = \overline{k} = \overline{(\overline{a}\overline{b}c)} = (a+b+\overline{c})$$

The new clause c is then appended to the original formula to obtain $S = S \cdot c$:

$$S = (\overline{a}+\overline{b}) \cdot (a+b+c) \cdot (a+b+\overline{c})$$

Note that the new clause c is unsatisfiable using the variables from k, ensuring that the next iteration of SAT will find a different satisfying cube. The new CNF S is solved by SAT again to obtain a new satisfying cube, for example $a\overline{b}c$. The steps are repeated until no new satisfying assignments are found.

Another useful extension of SAT is *Weighted partial Max-SAT* (WPMS) which aims to satisfy a subset of the clauses. In WPMS, each clause in the CNF is identified as a *hard clause* or *soft clause*. Each soft clause is associated with a weight. The problem then is to identify an assignment that satisfies *all* hard clauses while maximizing the total weight of the satisfied soft clauses.

Definition I.14: **Weighted partial Max-SAT** (WPMS). Given a Boolean formula S (on a set of binary variables X) expressed in CNF, where each clause is identified *hard clause* or *soft clause* , the objective of WPMS is to identify an assignment of the binary variables in X that satisfies all hard clauses in S and maximizes the total weigh of all satisfied soft clauses, if such an assignment exists.

To give an example of WPMS, consider the following CNF:

$$S(x,y,z) = (\overline{x}+\overline{y})\cdot(\overline{x}+\overline{z})\cdot(\overline{y}+\overline{z})\cdot(x)\cdot(y)\cdot(z)$$

In this CNF, we are given that the first three clauses $(\overline{x}+\overline{y}),(\overline{x}+\overline{z}),(\overline{y}+\overline{z})$ are hard clauses, while the last three clauses are soft clauses $(x),(y),(z)$. Furthermore, the soft clauses $(x),(y),(z)$ are assigned weights of 3, 7, and 6 respectively.

For WPMS, an assignment of (x,y,z) must satisfy all the hard clauses and maximize the total weight of the satisfied soft clauses. In this example, each soft clause contains a single positive literal, and to satisfy any one of the soft clauses, its corresponding variable needs to be set to its positive literal. However, no more than one of the three variables can be assigned to their postive litera, while the other two variables must be assigned to their negative literals in order to satisfy the hard clauses. Of the soft clauses, (y) has the highest weight, so the satisfying solution with maximum weight is $(x,y,z) = (0,1,0)$ or $\overline{x}y\overline{z}$, with weight = 7.

I-E.3. SAT Solvers

While several high-performance algorithms exist for solving SAT, the most popular in logic synthesis and electronic design are conflict driven methods based on the Davis-Putnam-Logemann-Loveland or DPLL algorithm [22].

The DPLL algorithm is a complete[2] search process for finding a satisfying assignment by implicitly pruning of the exponentially sized search space. This algorithm searches for a satisfying assignment through repeated *branching* and *decision* steps. In the branching step, an unassigned variable x is selected in the CNF formula S. The decision step sets x to 1 or 0, resulting in the CNF formulas S_x (which is S with x replaced by 1) or $S_{\overline{x}}$ (which is S, with x replaced by 0). The branching and decision steps are recursively repeated for all remaining unassigned variables in S until a solution is found, or the search space is exhausted.

The DPLL algorithm performs a depth first traversal of the state space. During the search, a partial variable assignment list p is recorded. If S_p contains an empty clause (0), for example $S_p = (\ldots) \cdot (0) \cdot (\ldots)$, then S_p is unsatisfiable. In this case, the DPLL algorithm *backtracks* and then branches or decides on a different variable or value. If all variables have been assigned in p and there are no empty clauses in S_p, then S is satisfiable and p is the satisfying assignment.

For example, consider the CNF formula $S(a,b,c) = (\overline{a} + \overline{b}) \cdot (a + b + c)$, where the variable a is the first to be selected for branching. Also, let us assume that a is set to its negative literal (a replaced by 0), and the assignment list is updated to $p = \overline{a}$. We evaluate $S_{\overline{a}}$ by setting $a = 0$ and simplifying the formula. We recall that a clause with a literal evaluating to 1 means the clause is satisfied, and can be removed from the CNF.

$$S = (\overline{a} + \overline{b}) \cdot (a + b + c)$$

$$S_{\overline{a}} = (\overline{0} + \overline{b}) \cdot (0 + b + c) = (b + c)$$

We continue the example by selecting variable b as the next variable for branching.

[2] A *complete* or *exact* algorithm is one that is guaranteed to find a solution if a solution exists.

Let us assume that b is set to its negative literal, and the assignment list is updated to $p = \overline{a}\overline{b}$. We evaluate $S_{\overline{a}\overline{b}}$ by setting $b = 0$ and simplifying the formula.

$$S_{\overline{a}\overline{b}} = (0 + c) = (c)$$

The last variable c is selected. Again, let us assume that c is set to its negative literal, and $p = \overline{a}\overline{b}\overline{c}$. We evaluate $S_{\overline{a}\overline{b}\overline{c}}$.

$$S_{\overline{a}\overline{b}\overline{c}} = (0)$$

In this case, the partial assignment $p = \overline{a}\overline{b}\overline{c}$ causes the formula evaluates to 0, and the DPLL algorithm backtracks. Let us assume that the algorithm now sets c to its positive literal ($c = 1$) and partial assignment is updated to $p = \overline{a}\overline{b}c$. We evaluate $S_{\overline{a}\overline{b}c}$.

$$S_{\overline{a}\overline{b}c} = (1)$$

At this point, all clauses have been satisfied and all variables have been assigned. We conclude that S is satisfiable and $p = \overline{a}\overline{b}c$ is a satisfying assignment.

Modern SAT solvers [23, 24, 25, 26] augment DPLL with techniques such as variable selection heuristics, clause learning, and watched literals to greatly improve SAT solving efficiency. In the following, we briefly describe each of these techniques.

Variable selection heuristics vary widely between SAT solvers. The next variable to branch on has a key role to play in determining solver efficiency. Common strategies include random selection of a variable, maximum occurrence of variable in clauses of minimum size, most frequent variable in unsatisfied clauses, or choose variables based on weights from conflicts.

Watched literals [25] is an efficient method to identify variables for assignment that are required to satisfy S_i. For example, if a clause consists of one unassigned literal and all other literals are set to 0 value, then the unassigned literal must be set to 1 value for the clause to be satisfied. In this situation, the unassigned variable set to 1 value is an implication. Any time a decision (variable branch or value set) is made in DPLL, this generates new implications.

In [25], the method selects (watches) two literals for each clause not yet satisfied in the partial assignment. The watched literals can be set to 1 or unassigned. With this method, satisfiability of a clause can be tested by checking whether one of the watched literals is 1, significantly reducing computation and memory requirements for SAT solving. So long as both watched variables are not set to 0 value, the clause is not implied.

For example, during the DPLL process if a variable x is set 0, all clauses with watched literal x must find another literal to watch as this implies that the other watched literal must be set to 1 to be satisfied. Furthermore, for any clauses that are satisfied when $x = 0$, $\bar{x}$ is set as a watched literal. If no other literals are available to watch, the algorithm must backtrack, and the results of which can be used in clause learning.

Clause learning [25, 26] is a technique which improves efficiency of DPLL by avoiding redundant computation on assignments that are unsatisfiable. The technique keeps tracks of clauses that become empty, causing a conflict in the algorithm. The CNF leading to a conflict is analyzed through its structure and implications to create a conflict clause to learn. The DPLL algorithm then backtracks and the conflict clause is included in the CNF.

In [26], each decision in the DPLL algorithm is recorded with the time (or time step) of the decision. For example, variable x_1 was set to 1 value at time 6, or variable x_9 was set to 0 value at time 1. All decisions up to the current assignment can be shown on an implication graph, for example Figure I.12.

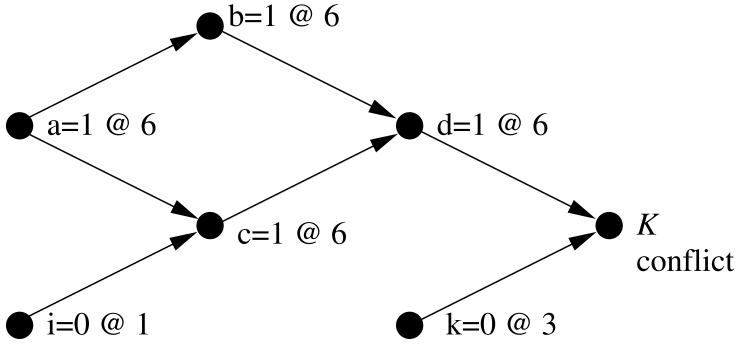


Fig. I.12. Implication graph for current assignment

When a conflict occurs at a decision K, the conflicting assignment A can be determined by traversing the graph backwards from K. Only the assignments at previous assignments of K are a sufficient condition for the conflict. For example, from Figure I.12, the conflicting assignment is $A = \{a = 1@6, i = 0@1, k = 0@3\}$. The conflict assignment induces the conflict clause $C = (\overline{a}ik)$. The conflicting assignment and induced conflict clause enables further implications which improves the search and backtracking in the SAT solver engine.

I-F. Chapter Summary

As we deepen our understanding of how cells operate and how genetic diseases occur, we realize that many cell components are involved in cell function and that a systems engineering approach is required. In this chapter, we described the key issues in genomics and provided an introductory background to the genome and gene regulatory network. Our work proposes to model the GRN as a Boolean network (FSM), from which we can use logic synthesis techniques and Boolean Satisfiability to tackle several genomics problems. We take advantage of improvements in modern SAT solvers to provide efficient and powerful tools for research. The following chapters present our methods and application to biological networks. In Chapter II, we present a method for inferring the gene predictor set [27, 28] from gene expression data. Following in Chapter III, we use the predictor set and gene expression data to determine gene function [29] to define a family of BNs. Chapter IV presents a method for inferring and ranking gene predictors from continuous gene expression data using modified Zhegalkin functions. Lastly, in Chapter V, we use ATPG techniques on the BN [30, 31] to identify genes leading to cancer and to determine drug selection for cancer therapy.

CHAPTER II

PREDICTOR SET INFERENCE USING SAT[1]

The inference of gene predictors in the gene regulatory network (GRN) has become an important research area in the genomics and medical disciplines. Accurate predictors are necessary for constructing the GRN model and to enable targeted biological experiments that attempt to validate or control the regulation process. In this chapter, we implement a SAT-based algorithm to determine the gene predictor set from steady state gene expression data (attractor states). Using the attractor states as input, the states are ordered into attractor cycles. For each attractor cycle ordering, all possible predictors are enumerated and a conjunctive normal form (CNF) expression is generated which encodes these predictors and their biological constraints. Each CNF is solved using a SAT solver to find candidate predictor sets. Statistical analysis of the resulting predictor sets selects the most likely predictor set of the GRN, corresponding to the attractor data. We demonstrate our algorithm [27, 28] on attractor state data from a melanoma study [32] and present our predictor set results.

II-A. Background

With increasing availability of gene expression data, the focus in computational biology has shifted to the understanding of gene regulation and its inter-relation with the biological system. The use of genome information has given rise to the possibility of ”personalized medicine” – targeted and specific disease prevention and treatment based on individual gene information [20, 21]. The urgent applications to cancer and gene-related diseases calls for

[1] Part of the data reported in this chapter is reprinted with permission from ”Inference of Gene Predictor Set Using Boolean Satisfiability” by Pey-Chang Kent Lin, Sunil P. Khatri. *IEEE International Workshop on Genomic Signal Processing and Statistics (GENSIPS) 2010*, Nov. 2010, pp. 1-4, Copyright 2010 by IEEE

the genomics field to significantly improve the algorithms used for accurate inference of the gene regulatory network (GRN).

In an organism, the genome is a highly complex control system wherein proteins and RNA produced by genes and their products interact with and regulate the activity of other genes [33]. A *predictor* for a target gene g_i is the collection of genes directly participating in the regulation of gene g_i. As such, the predictor does not consider the type of regulation (repression versus activation), and is analogous to the *support* of a function in logic synthesis. Each gene has a single predictor (which is a collection of genes) and the *predictor set* is the set consisting of predictors of each gene in the GRN.

There are several observations that impact the formulation of our GRN model and predictor inference algorithm. First, the activity level (i.e. activation or repression) of all genes at a particular time t represents the *state* of the GRN at that time t. From our knowledge of biological systems, we observe that over time, cellular processes converge to sequences of stable *attractor* states. Some of these attractor states represent normal cellular phenomena in biology (i.e. cell cycle and division), while other attractor states are consistent with disease (i.e. metastasis of cancer). Second, the GRN is often inferred by observing microarray-based experimental data though which the activity level of genes is measured. Both observations of gene activity (or state) can be used to infer the gene regulation network. The disadvantage of using microarray data is that such studies do not involve controlled time-series experimental data. Hence the measurements are assumed to arise from cyclic sequences of gene expressions (attractor states) in steady state. Such a sequence is referred to as an *attractor cycle*. The GRN is then inferred from this data, using methods traditionally based on probabilistic transition models [34, 35].

As previously mentioned, it is necessary to determine the predictor set in order to reconstruct the GRN. However, there may exist many possible predictors for any gene, based on the attractor cycle data. Furthermore, only certain combinations of predictors may form a valid predictor set, due to biological constraints. The issue addressed in this paper is how to efficiently and deterministically select the predictors that form the predictor set. We have implemented a Boolean satisfiability (SAT) based algorithm for the inference of gene predictor sets. Satisfiability is a decision problem of determining whether the variables in a Boolean formula (expressed in Conjunctive Normal Form or CNF) can be assigned to make the formula evaluate to *true*. Although SAT is NP-complete, many SAT solvers have been developed to quickly and efficiently solve large SAT problems. Our algorithm takes advantage of a recent SAT solver to find the predictor set.

The basic outline of our SAT-based algorithm for predictor set inference is described briefly below. First, all possible orderings of attractor states are enumerated, yielding all possible attractor cycles. For each ordering, we enumerate all predictors that are logically valid, and create a CNF expression which encodes all these predictors and biological constraints (such as cardinality bounds on the predictors). A SAT solver is then used to find the valid candidate predictor sets. After this process is done iteratively for all attractor cycle (orderings), statistical analysis provides the most likely predictor set. Note that this paper does not claim to extract the GRN. Using the predictor set inferred by this paper, we plan to infer the GRN in a subsequent research effort.

The key contributions of this chapter are:

- We develop a Boolean Satisfiability based approach to realize the gene predictor set from attractor state data.
- We modify an existing SAT-solver (MiniSat [23]) for efficient all-SAT computation, and further optimize the decision engine of MiniSat for improved predictor set inference.
- On gene expression data from a melanoma study [32], we apply our SAT-based algorithm and present the predictor set, including the predictor for the cancer gene

WNT5a.

- Our approach can be used to find the predictor set for any gene related disease, provided attractor state data is available. The predictor set information obtained from our algorithm can be used by biologists to fine tune their gene expression experiments.

II-B. Previous Work

In the context of predictor set inference, [36, 37] use dynamic Bayesian networks and probabilistic Boolean networks (PBNs). The GRN is then inferred from this data, using methods traditionally based on probabilistic transition models [34, 35] The method proposed considers gene prediction using multinomial probit regression with Bayesian variable selection. Genes are selected which satisfy multiple regression equations, of which the strongest genes are used to construct the predictor set. The target gene is predicted based on the strongest genes, using the coefficient of determination to measure predictor accuracy.

Another method proposed by [38] also assumes a PBN model. A partial state transition table is constructed based on available attractor state data. From this state transition table, predictors with 3 or less regulating genes are selected for each target gene. All unknown values in the table are randomly set. The Boolean network is simulated for several iterations using different starting states, observing whether the states eventually transition to an attractor cycle. If the simulation successfully transitions to an attractor cycle, the selected predictors are considered as a valid predictor set. This process is repeated, to build a collection of Boolean Networks which are combined to form a Probabilistic Boolean Network (PBN).

Our larger goal is to find a small number of *deterministic* GRNs, rather than a PBN. Towards this, we need to first find ways to accurately find the predictor set. This is the focus of this chapter. Philosophically, our aim is to invest effort into accurate predictor set determination, so that the results can be used to find high quality deterministic GRNs.

II-C. Background

This section describes our background and problem definition for inference of predictor sets using SAT. We begin with some GRN definitions and then explain some of the biological constraints that will be used in our formation for the the next section.

Definition II.1: A **predictor** $f_i = \{g_j, g_k, \cdots\}$ lists the set $\{g_j, g_k, \cdots\}$ of genes which regulate the activity of gene g_i.

Definition II.2: The **predictor set** is the complete set of predictors $\{f_1, f_2, \cdots, f_n\}$ for the GRN with n genes $g_1, g_2, \cdots, g_n$.

Based on the gene products of one or more genes in a set f_i, a gene g_i can become repressed or activated. in this case f_i is said to be predictor of gene g_i. A predictor for target gene g_i is the collection of genes directly participating in the regulation of gene g_i. As such, the predictor does not consider the type of regulation. Each gene has a single predictor and the predictor set is the set consisting of predictors of each gene.

Note, we can relate these terms to logic synthesis: the predictor is identical to the logical support of a logic node, while the predictor set is akin to the circuit netlist. The Boolean network GRN then is the complete logic circuit including function for each node.

Definition II.3: Given a starting state, within a finite number of steps, the network will transition as determined by the gene functions into a cycle of states, called an **attractor cycle**. States in an attractor cycle are called **attractor states**. The attractor cycle represents the long term behavior of the network and absent perturbation, a network that has transitioned to an attractor will continue to cycle thereafter.

There are several observations that impact the formulation of our GRN model and predictor inference algorithm. First, the activity level (i.e. activation or repression) of all genes at a particular time t represents the *state* of the GRN at that time t. From our knowledge of biological systems, we observe that over time, cellular processes converge to sequences of stable *attractor* states. Some of these attractor states represent normal cellular phenomena in biology (i.e. cell cycle and division), while other attractor states are consistent with disease (i.e. metastasis of cancer).

Second, the GRN is often inferred by observing microarray-based experimental data though which the activity level of genes is measured. Both observations of gene activity (or state) can be used to infer the gene regulation network. The disadvantage of using microarray data is that such studies do not involve controlled time-series experimental data. Hence the measurements are assumed to arise from cyclic sequences of gene expressions (attractor states) in steady state. Such a sequence is referred to as an *attractor cycle*.

II-D. Problem Formulation and SAT Construction

Given gene expression data (a set of unordered attractor states) as input, we would like to determine the best predictor set. We first present an outline of our SAT-based algorithm, and then explain the steps through a simple example.

The algorithm has three main steps.

1. **SAT Construction for Predictor Set:** In this step, attractor states are ordered into attractor cycles in all possible ways. For each possible ordering of attractor states into attractor cycles, all possible predictors are found and a CNF is generated encoding valid predictor sets.

2. **All-SAT:** Each attractor ordering from step 1 generates a CNF which is solved for All-SAT. All satisfying cubes are recorded, where each satisfying cube corresponds to a predictor set. The first two steps are repeated for all attractor cycle orderings.

3. **Predictor Set Selection:** Statistical analysis on the All-SAT results determines the most frequent (likely) predictor set for the GRN. This step is explained in Section II-E.

To illustrate the SAT-based algorithm, we apply it to a simple example with three genes (g_1, g_2, g_3) and gene expression data with three lines $(010, 110, 111)$. The present state of these genes is represented by the variables $< x_1, x_2, x_3 >$ and the next state is represented by the variables $< y_1, y_2, y_3 >$. We assume each line was measured in steady state and therefore is an attractor state.

We order (or arrange) the attractor states into attractor cycles for which there are six possibilities for our example. One ordering is with each attractor state transitioning to itself with a self-edge, resulting in three singleton attractor cycles. Two possible orderings result when all three attractor states form a single attractor cycle of length three. The last three possible orderings have two attractor cycles, one cycle with length two and the other cycle of length one. We focus our example on an ordering with two attractor cycles, as shown in Table II.1.

II-D.1. Partial State Transition Table

For each valid attractor cycle ordering, a *partial state transition table* is constructed, containing the attractor states. Table II.1 shows the partial state transition table for the example attractor cycle ordering. To find all valid predictors of a gene, each next state column is checked against all combinations of the current (present) state columns. For example, let us explore gene g_2 and g_3 as a predictor for gene g_1. For gene g_1, the next state bit is y_1, while for gene g_2 and g_3, the current (present) state bits are x_2 and x_3. In the first two rows of TableII.1, $< x_2, x_3 >= 10$. However, in row 1, $y_1 = 1$, while in row 2, $y_1 = 0$, which forms a contradiction (since the same input cannot result in different outputs). Therefore, gene g_1 cannot be predicted by genes g_2 and g_3.

Now, consider genes g_1 and g_3 as a predictor for gene g_1. There is no contradiction, and the combination is logically valid. Thus one possible predictor for gene g_1 is $f_1 = \{x_1, x_3\}$. All valid predictors with P (user-defined) or less inputs are exhaustively searched and recorded for CNF formulation (which is done in the next step). In our example, gene g_1 has 2 possible predictors $\{x_1, x_3\}$, $\{x_1, x_2, x_3\}$ which we label v_1^1, v_2^1 respectively. We assume that a gene cannot self-regulate, so $\{x_1\}$ by itself is not a valid predictor.

Current state			**Next state**		
x_1	x_2	x_3	y_1	y_2	y_3
0	1	0	1	1	0
1	1	0	0	1	0
1	1	1	1	1	1

Table II.1. Example 3-Gene Partial State Transition Table

II-D.2. SAT Formulation and GRN Constraints

After all predictors are found for each gene, we generate the SAT formula which encodes logically valid predictor sets. The j^{th} predictor for gene i is assigned a variable v_j^i. Gene g_1 in our example will have two predictor variables $v_1^1 \equiv \{x_1, x_3\}$, $v_2^1 \equiv \{x_1, x_2, x_3\}$. Gene g_2 and g_3 will have their own corresponding predictor variables $v_1^2 \equiv \{x_1, x_2\}$, $v_2^2 \equiv \{x_1, x_3\}$, $v_3^2 \equiv \{x_2, x_3\}$, $v_4^2 \equiv \{x_1, x_2, x_3\}$ and $v_1^3 \equiv \{x_1, x_3\}$, $v_2^3 \equiv \{x_2, x_3\}$, $v_3^3 \equiv \{x_1, x_2, x_3\}$ respectively. There are three constraints that we incorporate while constructing the CNF that encodes valid predictor sets. The conjunction of these constraints forms our final CNF.

1. The first constraint (S_1) is that all genes in the GRN must have a predictor. In other words, we assume that all genes are highly correlated and are ”participating” in the GRN. For gene i, all of its associated predictor variables are written in a single clause

$$c_i^1 = (v_1^i + \cdots + v_j^i)$$

In our example, for g_1, $c_1^1 = (v_1^1 + v_2^1)$. For g_2 and g_3, we have $c_2^1 = (v_1^2 + v_2^2 + v_3^2 + v_4^2)$ and $c_3^1 = (v_1^3 + v_2^3 + v_3^3)$ respectively.

To satisfy any c_i^1 clause, at least one predictor in the clause must be chosen. To ensure that at least one predictor is chosen for all genes, we write the conjunction of all c_i^1 clauses as S_1 (Equation 2.1).

$$S_1 = c_1^1 \cdot c_2^1 \cdot c_3^1 \tag{2.1}$$

2. The second constraint (S_2) specifies that for each gene, exactly one predictor is chosen. The assumption is that a gene cannot have multiple predictors. To formulate the clauses c_i^2 for gene i, smaller clauses are formed from all pairs of combinations of its predictors $v_{1\cdots j}^i$. In each of these clauses of pairs of variables, both predictor variables are complemented.

$$c_1^2 = (\overline{v_1^1} + \overline{v_2^1})$$

$$c_2^2 = (\overline{v_1^2} + \overline{v_2^2}) \cdot (\overline{v_1^2} + \overline{v_3^2}) \cdot (\overline{v_1^2} + \overline{v_4^2}) \cdot (\overline{v_2^2} + \overline{v_3^2}) \cdot (\overline{v_2^2} + \overline{v_4^2}) \cdot (\overline{v_3^2} + \overline{v_4^2})$$

$$c_3^2 = (\overline{v_1^3} + \overline{v_2^3}) \cdot (\overline{v_1^3} + \overline{v_3^3}) \cdot (\overline{v_2^3} + \overline{v_3^3})$$

Any selection of two or more predictors for gene i will result in the clauses of c_i^2 becoming unsatisfiable. The c_i^1 clause ensures that at least one predictor will be chosen for gene i, and c_i^2 forces the selection of exactly one predictor for gene i. The conjunction of all c_i^2 clauses forms the constraint S_2 (Equation 2.2), which forces SAT to choose only one predictor per gene.

$$S_2 = c_1^2 \cdot c_2^2 \cdot c_3^2 \tag{2.2}$$

3. The last constraint (S_3) requires that each gene must be used as a predictor for at least one other gene in the predictor set. A gene that is not used in any predictor does not perform any regulation function and could be removed from the GRN. S_3 ensures that this does not occur. To ensure that gene g_i is used in at least one predictor, we form clauses c_i^3 which include all predictors that use gene g_i as input. To specify that gene g_i must be used, we also include a single variable clause (x_i) to c_i^3. For gene g_1, g_2, and g_3, we create the following clauses c_1^3, c_2^3, and $c_1^3 3$ respectively:

$$c_1^3 = (x_1) \cdot (\overline{x_1} + v_1^1 + v_2^1 + v_1^2 + v_2^2 + v_4^2 + v_1^3 + v_3^3)$$

$$c_2^3 = (x_2) \cdot (\overline{x_2} + v_2^1 + v_1^2 + v_3^2 + v_4^2 + v_2^3 + v_3^3)$$

$$c_3^3 = (x_3) \cdot (\overline{x_3} + v_1^1 + v_2^1 + v_2^2 + v_3^2 + v_4^2 + v_1^3 + v_2^3 + v_3^3)$$

To satisfy these clauses, x_i and at least one other predictor variable in the second clause of c_i^3 must be selected. S_3 is a conjunction of all the c^3 clauses (Equation 2.3).

$$S_3 = c_1^3 \cdot c_2^3 \cdot c_3^3 \tag{2.3}$$

The final SAT formula S as a conjunction of the S_i formulas (Equation 2.4).

$$S = S_1 \cdot S_2 \cdot S_3 \tag{2.4}$$

II-D.3. All-SAT

The SAT solver performs an All-SAT on S. The satisfying cubes (each cube encodes a candidate predictor set) from the All-SAT output are collected. The process is repeated for the remaining attractor cycle orderings. From the results, we find the most likely predictors based on the frequency of occurrence of the predictors across all orderings. Three methods are used to analyze the statistical results, which will be described in the next section.

In general, the above algorithm can be applied to input data for N genes and A attractor states. The total number of attractor state orderings is $A!$. For each ordering, there can be up to $O(N^3)$ predictors per gene. The SAT search space per ordering is on the order of $O(2^{N^3})$, resulting in overall complexity of $O(A!2^{N^3})$. Typically, the number of attractor states A recorded through gene expression measurements is small. As such, $A!$ is thus much smaller than $2^{(N^3)}$, so the runtime complexity is dominated by the All-SAT operation. For pragmatic reasons, our algorithm stops each All-SAT after T minutes (or C cubes), where T or C is defined by the user.

II-E. Experimental Results

To evaluate our SAT-based algorithm for inferring gene predictors, the algorithm was tested on gene-expression data from a melanoma study done by Bittner and Weeraratna [32]. In the melanoma study, it was observed that an abundance of RNA (expression) for gene $WNT5A$ was associated with a high metastasis of melanoma. The study measured 587 genes with 31 gene expression patterns (lines). Seven genes are believed to be closely knit: $PIRIN, S100P, RET1, MART1, HADHB, STC2$, and $WNT5A$. There are 18 distinct patterns, which were reduced to seven using Hamming-distance of one, in Table II.2. These seven lines form the attractor states which are the input to our algorithm.

	$PIRIN$	$S100P$	$RET1$	$MART1$	$HADHB$	$STC2$	$WNT5A$
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
BAD	0	0	0	0	0	1	1
	0	0	1	1	1	1	1
	1	0	1	0	0	0	1
GOOD	0	1	0	0	0	0	0
	0	1	1	1	0	0	0
	1	0	1	1	1	1	0
	1	1	0	1	1	0	0

Table II.2. Attractors for **Melanoma** Network

For the experiments, we assume two additional specifications. First, we divide attractor states into good and bad states, based on the presence of $WNT5A$. We allow good attractor states to cycle only to other good attractor states, and bad attractor states can only cycle to other bad attractor states. Second, we limit the maximum attractor cycle length L to 3, and the maximum number of predictor inputs P to 3, because long attractor cycles and large predictor inputs are highly complex and less likely to occur in biological systems [39, 33].

Our algorithm utilizes a modified open-source and highly efficient exact SAT-solver called MiniSAT v1.14 [40, 23]. All-SAT operations were limited to a 30 minute time-out. On average, each All-SAT run yielded 10K satisfying cubes in this duration. Our algorithm was implemented and run on a Pentium 4 Linux machine with 4GB RAM. MiniSat [23], was originally designed to find a single satisfying assignment. We modified MiniSat to perform All-SAT as MiniSat normally only returns one SAT result. We further modified

MiniSat to always randomly select decision variables during the solving process to increase the activity of all variables.

The unaltered MiniSAT uses a heuristic for selecting the next decision variables. However, this heuristic results in many of the same variables being chosen over iterative runs of MiniSat. To increase the activity of all variables, we change the random variable frequency of MiniSat to 100% (from 2% in the unaltered MiniSat code). This forces MiniSAT to always choose a random variable on every variable-branch decision. A random variable frequency of $f\%$ means that MiniSat selects the next variable randomly $f\%$ of the time.

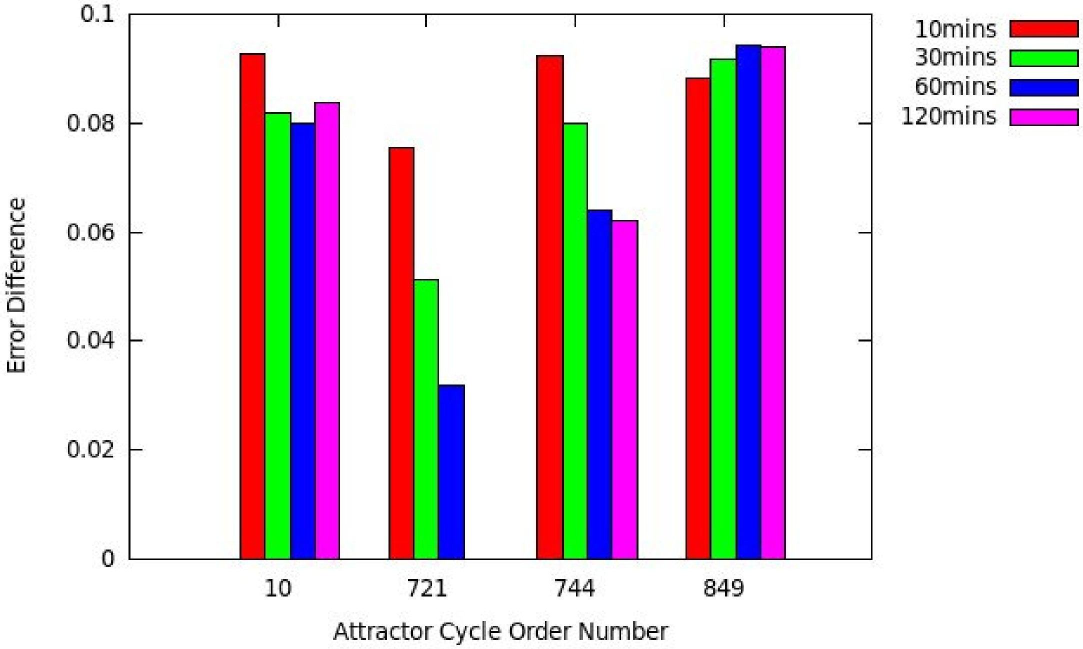


Fig. II.1. Average predictor error difference on melanoma attractor data using MiniSat without modification

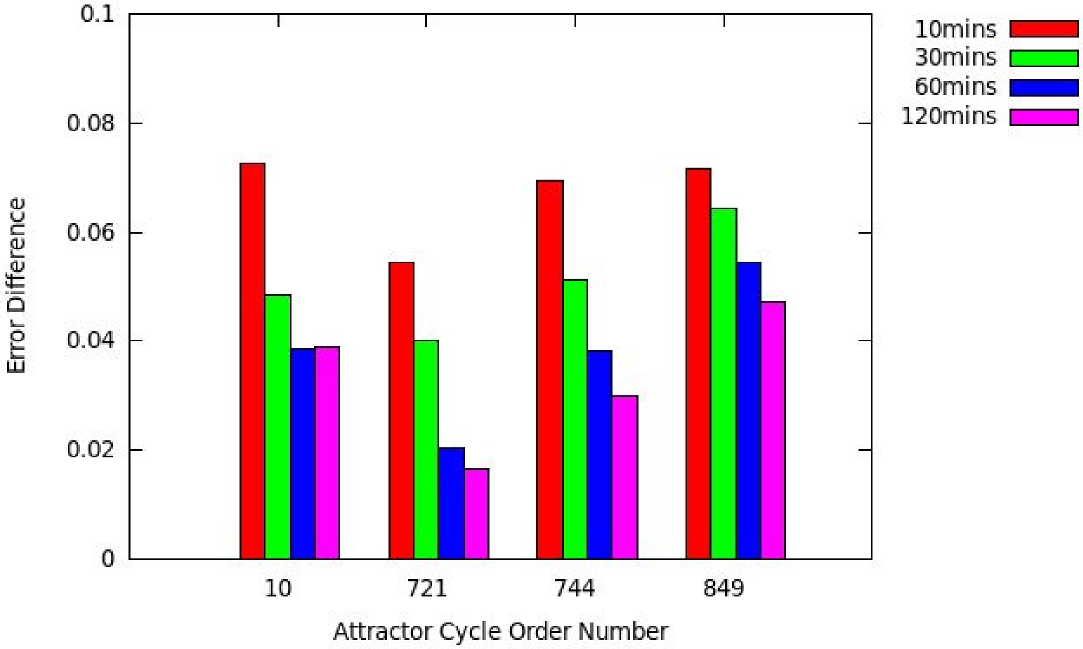


Fig. II.2. Average predictor error difference on melanoma attractor data using MiniSat with random variable selection modification

To validate the quality of predictor selection using our modified All-SAT, our algorithm was run on four selected attractor cycle orderings (labeled 10, 721, 744, and 849) using melanoma data from [32]. The All-SAT operation was allowed to run for 12 hours (which approximates a complete All-SAT). In the case of attractor cycle order 721, all cubes were found. In Figures II.2 and II.1, we compare the average difference in all the predictors' occurrence frequency in the complete All-SAT result with the results obtained with shorter All-SAT runtimes (10, 30, 60, and 120 minutes). Figure II.2 shows the average error difference of all predictors' frequency for the four orderings, using MiniSat with the random variable selection modification (100% random variable frequency), while Figure II.1 shows the same results without random variable selection (2% random variable frequency). Across the four orderings analyzed, the average error difference of all predictors' occurrence frequency (shown in Figures II.2 and II.1) is significantly lower using the random variable selection modification than without. Furthermore, the average error difference decreases with increasing runtime when using random variable selection. From this experiment, we determine that 30 minutes with random variable selection was sufficient to achieve an average of $\leq 5\%$ difference in the predictors' occurrence frequency compared to the full All-SAT results.

The following presents our results after collection of All-SAT results from all valid attractor cycle orderings. In Figure II.3, we display a histogram of all logically valid predictors and their frequency of occurrence, across all attractor orderings. In the sequel, a predictor label of 2367 means that gene g_2 is predicted by genes g_3, g_6, and g_7. From this chart, we can observe that certain predictors occur with significantly higher frequency than others. For example with gene g_1, the predictor $\{x_3, x_5, x_7\}$ (*PIRIN* predicted by *RET*1, *HADHB*, *WNT*5*A*) occurs with much higher frequency than all other predictors for gene g_1. This indicates that this predictor is most likely to be present in the final predictor set. From this data, we propose three methods (A, B, AB) for selecting the predictor set.

II-E.1. Method A

In **method A**, a predictor histogram is created as in Figure II.3. From the histogram, for each gene g_i, we find its predictor p^i_j such that p^i_j is the most frequently occurring predictor of gene g_i and the *resolution ratio* R_i of this predictor (defined as the ratio of the occurrence frequency of p^i_j to the occurrence frequency of the next most frequently occurring predictor of gene g_i) is maximum. Among all genes, we choose the one with the highest resolution ratio, and select its most frequently occurring predictor as its final predictor. After selecting this final predictor, we regenerate the histogram, discarding any candidate predictor sets that do not contain the final predictor(s) that have been selected in previous steps. The process repeats until all genes have a single final predictor. The set of final predictors of all genes forms the predictor set. The advantage of method A is that at every iteration, we select real predictors that have a high overall occurrence in the solution. However the method may have problems selecting final predictors if the resolution ratio is low (i.e. when the frequencies of occurrence of the predictors are nearly identical).

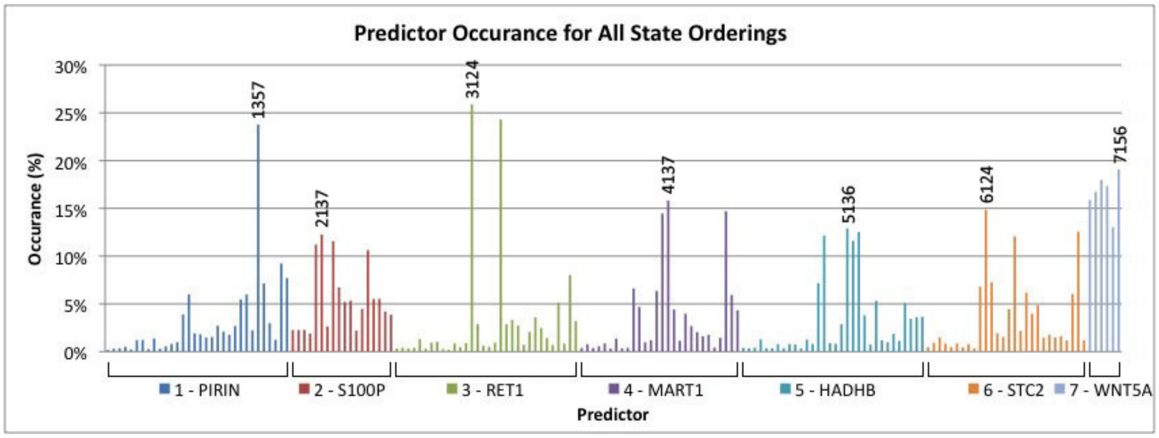


Fig. II.3. Method A: Predictor occurrence for all valid attractor cycle orderings (first iteration: no predictor selected)

II-E.2. Method B

As an alternative, **method B** is proposed, to determine for each gene i, how likely it is that gene g_i will predict the other genes in the GRN. In other words, we ask what is the occurrence frequency of x_i in the predictors of f_j. Table II.3 shows in entry (i, j) how frequently a gene g_i is used to predict a gene g_j. This table is populated by summing the occurrence frequency of all predictors of g_j that have gene g_i as one of their inputs. As such, any entry can be $\geq$1, and is a measure of the usefulness of g_i as a predictor for g_j. The predictor of g_j is determined by finding, for each column j of Table II.3, the three largest entries and adding their values. Suppose we call this sum s_j (the resolution score of column j). We compute the resolution score for all columns and select the final predictor for the column with the highest resolution score. This final predictor is formed by listing the 3 input genes that correspond to the 3 entries that were used to compute the highest resolution score. Similar to method A, we reiterate the process by regenerating the table after discarding all predictor sets that do not contain predictors that were selected in previous steps. Method B has the advantage of being more robust when no *single* predictor has a significantly higher occurrence frequency than others. However, there is no guarantee that the predictor selected by method B is a valid predictor. If this happens, we select the column with the next highest resolution score.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
x_1		0.59	0.68	0.57	0.69	0.60	1.00
x_2	0.24		0.41	0.29	0.33	0.49	0.51
x_3	0.65	0.48		0.76	0.58	0.56	0.17
x_4	0.39	0.40	0.78		0.54	0.44	0.29
x_5	0.56	0.30	0.27	0.44		0.39	0.36
x_6	0.42	0.54	0.52	0.41	0.44		0.67
x_7	0.64	0.63	0.24	0.48	0.32	0.45	

Table II.3. Method B: Gene Occurrence for All Predictors (First Iteration)

II-E.3. Method AB

In our experiments, we also use a hybrid **method AB** which works in the following manner. Both methods A and B are used to select their best predictor. If both methods produce the same predictor f_i, we select this predictor as a final predictor. If not, we list the best predictors for each gene, for both methods. If multiple predictors match for both methods, we choose the final predictor as the one with the highest weighted sum of the resolution ratio and resolution score. The resolution ratio is weighted by 0.3 and the resolution score is weighted by 0.7. The weighting factor for the resolution ratio is lower since the resolution ratio values of any gene are often close to 1. In such a situation, we would like to favor method B. If no predictor is produced by the previous step, we look at the top five predictors of method A for each gene and calculate the weighted sum of their resolution ratio and resolution score. The predictor with the highest weighted sum is selected as the final predictor. The process is reiterated, regenerating the histogram and table at each step, discarding any predictor sets that do not contain any of the previously selected final predictors. With this combined approach, we are able to select predictors with a higher degree of confidence and robustness.

We process our All-SAT data from melanoma attractor data of [32] using methods A, B, and AB. Results are shown in Table II.4 and shows what predictor was selected for each gene and the accompanying resolution ratio, resolution score, or weighted sum.

		$PIRIN$	$S100P$	$RET1$	$MART1$	$HADHB$	$STC2$	$WNT5A$
		x_1	x_2	x_3	x_4	x_5	x_6	x_7
A	Predictor set	1357	2137	3146	4357	5124	6124	7124
	Resolution ratio	2.57	1.41	1.34	1.30	1.41	1.66	1.31
B	Predictor set	1357	2137	3146	4137	5134	6137	7126
	Resolution score	1.78	1.77	1.84	1.97	1.99	1.98	2.56
AB	Predictor set	1357	2367	3146	4137	5137	6357	7124
	Weighted sum	2.06	1.57	1.75	1.61	1.45	1.39	1.88

Table II.4. Melanoma Network Predictor Set Selection

From the results, we can draw several conclusions:

- The iterative steps in regenerating the histogram (or table) retain only cubes (predictor sets) that contain previously selected final predictors. Hence the final predictor set from *each* method is a valid satisfying cube of the SAT formula S.
- The final predictor set is present in a select number of attractor cycle orderings. For

example, the final predictor set selected by methods A, B, and AB are found in respectively 8, 4, and 6 attractor cycle orderings out of the total 5040 possible orderings. Hence the algorithm will enable us to generate a few deterministic GRNs.

- Some predictors are common among the predictor sets between the three methods. For example, all three methods select $f_1 = \{x_3, x_5, x_7\}$ ($PIRIN$ predicted by $RET1, HADHB, WNT5A$) as well as $f_3 = \{x_1, x_4, x_6\}$. We can conclude this predictor is highly likely to be a final predictor in the GRN. Also, a majority of the predictors selected by the three method share common input genes. For example, the predictor selected by all methods for gene x_2 ($S100P$) contain 2 common genes $\{x_3, x_7\}$ ($RET1, WNT5A$), indicating these 2 genes are likely to be contained in the final predictor of f_2. Similarly f_7 has two common genes x_1 and x_2 for all methods.
- Using the above results, biologists can target their research on gene regulation and control, focusing on the gene relationships determined by the predictor set results.

II-F. Chapter Summary

Determining the predictor set for a gene regulatory network is important in many applications, particularly inference and control of the GRN which we discuss in subsequent chapters. In this research, we formulate gene predictor set inference as an instance of Boolean satisfiability. In our approach, we determine all possible orderings of attractor state data, generate the CNF encapsulating predictor and biological constraints, and apply a highly-efficient and modified SAT solver to find candidate predictor sets. The SAT results are analyzed using three selection methods to produce the final predictor set. We have tested our algorithm on attractor state data from a melanoma study, and determined the predictor sets for this GRN.

The results of this research, however, only reveals the predictor set (topology) of the

GRN. Our next step is to determine the gene regulating function (logic) of the genes in the GRN to fully define the BN. In the next chapter, we describe a logic synthesis method for determine gene functions for a GRN using a SAT-based logic synthesis approach.

CHAPTER III

DETERMINING GENE FUNCTION IN BOOLEAN NETWORKS USING SAT[1]

There are many instances where the circuit topology of the GRN is known, but the logic function of each node in this topology is not. In addition, a number *N* of measurements on the gene expression of the GRN are given or are known. Using this information, this chapter will derive SAT based algorithms which yield the logic of every node in the GRN so that the *N* gene expression measurements and topology are satisfied. If *N* is too small, then a multitude of GRNs may satisfy the observed behavior, yielding a reduced certainty in the final result due to lack of data. We will also study the behavior of the number of satisfying GRNs with respect to the number of observations *N*.

III-A. Background

In the cell, genes interact and communicate using a complex interconnected network called the gene regulatory network (GRN) [33]. The GRN and gene expression defines cell function and behavior. An accurate model of the GRN is necessary for understanding cell behavior, for learning how genetic diseases arise and for developing intervention strategies to treat such diseases.

In many situations, biologists can produce gene predictor sets or connectivity graphs, denoting which genes act upon or regulate each other. While gene predictor sets or connectivity graphs show how genes are interconnected, they do not provide any information about the regulating function of the genes. Predictor sets are generally useful, but without information about the regulating function, they cannot be used to simulate the dynamic in-

[1] Part of the data reported in this chapter is reprinted with permission from ”Determining Gene Function in Boolean Networks using Boolean Satisfiability” by Pey-Chang Kent Lin, Sunil P. Khatri. *IEEE International Workshop on Genomic Signal Processing and Statistics (GENSIPS) 2012*, Dec. 2012, pp. 1-4, Copyright 2012 by IEEE

teraction between genes which is crucial for the intervention and control of the GRN. Thus, a major goal for the genomics and the medical field is to determine the regulating function of the genes in the GRN.

At the same time, biologists may have prior knowledge of the GRN, gained through observations of gene expression or pathway information. These observations provide insight into the GRN state and, in turn, the gene regulation function. Such observations can be complete or partial, and may curated from different sources or databases.

The problem then is how to determine the gene function and create a complete and functional GRN model that matches the predictor set and gene expression observations. Assuming a Boolean network model [12] for the GRN, the gene expression for a single gene is a binary value (expressed or not expressed) and the gene regulation function is represented as a Boolean logic function. For each time step, all genes are assumed to update at the same time, and the value of all genes at a particular time step represents a state in the GRN at that time step. In this discussion, we present an efficient approach to assign a logic function to each gene, given a predictor set, such that the gene expression observation are matched.

In this work [29], we leverage mathematical tools from the field of logic synthesis in digital circuit design. Logic synthesis techniques have been recently applied to genomics in [41, 42, 27, 30, 28]. In order to determine gene function, [41] uses Karnaugh maps (K-maps) to explicitly generate two-level logic functions based on pathway information for a Boolean network.

In this chapter, we develop a general Boolean satisfiability (SAT) based implicit method to select logic functions and generate BNs that match a predictor set and gene expression observations. In our method, all possible logic functions are implicitly explored for each gene, based on the predictor set, with a multiplexer (MUX) selecting one of these functions for each gene. The resulting logical circuit is duplicated, and each copy is assigned one of the observed gene expression observations. All copies (now represented as a SAT formula) are then solved in parallel by linking the MUX selectors of each copy, and using a SAT-solver to select a function for each gene, and generate a BN that satisfies all input observations. Where more than one valid selection exists, our method generates a family of satisfying Boolean networks.

III-B. Previous Work

One commonly used representation scheme for GRNs is the Boolean network (BN) [12]. In this model, genes are binary valued (expressed or not expressed), and can act on (regulate) other genes. Regulation is represented using Boolean functions. In the BN, each gene takes its inputs (the values of its regulatory genes) and produces a new output value according to its Boolean function. All genes in the BN update synchronously, and the values of the genes at a given point of time represent the state of the BN at that time. In reality, gene expression is continuous; however a discrete model like BN is preferred because many genes exhibit switch-like behavior [19] and the discrete model simplifies analysis. Furthermore, the Boolean network is inherently a logic circuit, and a vast number of techniques from the field of logic synthesis can be applied to BN.

One such logic synthesis technique is the Kaurnaugh Map (K-map) [43] which was used in [41] to assign logic function to genes and generate Boolean networks from a priori gene pathway information. The K-map is an explicit method to represent and to simplify a Boolean function. Pathway information is used to create partially filled K-Maps which describe the update functions (or the next state functions) for each of the genes in an incompletely specified manner. Minimization of the functions in each K-map yields a family of Boolean networks. One issue with this approach is that logical conflicts can arise between different update functions obtained in this manner from the pathway information. The paper attempts to resolve these conflicts by perturbing the pathway information, possibly leading to a vastly different network.

One characteristic of assigning logic to the update function given a predictor set, is that in a predictor set, the gene connections or ”wiring” is fixed. Hence, the problem is how to determine the logic function of each gene, to obtain the GRN. Similar situations arise in digital design (an example is the wire planning problem in which wires or communication channels are placed before logic synthesis [44]). One method to approach this problem uses SPFDs (sets of pairs of functions to be distinguished) [45] which expresses the functional flexibility of nodes in a Boolean network. In [46, 47], SPFDs have been used to optimize Boolean networks. One drawback is that SPFDs are usually implemented using Boolean Decision Diagrams (BDDs), which do not scale well due to an exponential memory usage for some classes of logic functions.

Other logic synthesis techniques [48, 49], which assign logic from state transition diagrams, start with state information about the circuit, and assign logic which optimally minimizes the number of gates needed to implement the logic. As mentioned, the predictor set (and hence the wiring) in our problem is fixed, and as such, these logic synthesis methods cannot be used, since they may change the wiring to minimize the logic.

This method we present uses a Boolean satisfiability (SAT) based method to assign logic. Many logic synthesis algorithms are based on SAT, and there are several efficient and well-developed SAT solvers [23, 25]. In the context of genomics, SAT has been applied to the analysis of GRNs. In [50], a method is presented for inferring GRN parameters by expressing GRN constraints in a SAT formula. In [51], a model checking method (based on SAT) is presented to find all attractors in a Boolean network. Boolean satisfiability is also used in [28] to infer the predictor set of the GRN from attractors, and determine optimal drug selection for cancer therapy. Our approach is fundamentally different in that we *generate* a family of BNs for a predictor set given binary valued gene expression data.

III-C. Our Approach

III-C.1. SAT-based Formulation for Gene Function Assignment

In our approach, the problem of gene function assignment is transformed into an instance of Boolean satisfiability, such that each satisfying solution is an assignment of gene functions in accordance to the input predictor set and gene expression states. The SAT formulation is done in two distinct steps – 1) circuit construction from the predictor set and 2) constraining the solution space using the gene expression states.

III-C.1.a. Circuit Construction from the Predictor Set

The first step is to construct a SAT-based circuit that implicitly represents all possible BNs, based on prior knowledge of the predictor set. From the predictor set, we can determine the *wiring* of the genes. A predictor states which genes regulate the target gene, or in circuit terms, which genes are wired to the input of a target gene.

To assign the function for a gene x_i, our approach implicitly enumerates all possible functions $\{g_{i_1}, g_{i_2}, \ldots\}$ and then selects one function as a solution. Thus in our circuit construction, for each gene, all possible functions are enumerated, and a multiplexor (MUX) is added to select exactly one output from all the functions. The inputs to the MUX are the outputs of the functions, and a select signal s_i controls which function to select as the output for gene i. As such, the MUX selection determines which function is assigned to the gene.

The circuit is then converted into a CNF formula. Each function (including the MUX) has a CNF formula associated with it. The formula is true if and only if the variables representing the gate's inputs and outputs take on values consistent with its truth table.

One method to write the corresponding CNF of a Boolean formula is using implications and Boolean transformations. The basic transformation of an implication $a \rightarrow b$ into

CNF is the single clause $(\overline{a}+b)$. The implication states that if a is true (1), then b is also true (1). We can examine the clause $(\overline{a}+b)$ and see that if $a=1$, then $\overline{a}=0$. As such, for the clause to be satisfied (evaluate to 1), b must be set to 1.

For example, consider a 2-input OR gate (g_i) with x and y as inputs and z as output. For an OR gate, the output is 1 if and only if one of the inputs are 1. As such, we have the following two implications:

$$z \rightarrow x+y \tag{3.1}$$

$$x+y \rightarrow z \tag{3.2}$$

For the first implication $z \rightarrow x+y$ (Equation 3.1), we write the corresponding clause.

$$(\overline{z}+x+y) \tag{3.3}$$

This clause is already in CNF format, so no further transformation is needed.

For the second implication $x+y \rightarrow z$ (Equation 3.2), we again write the corresponding clause.

$$(\overline{x+y}+z) \tag{3.4}$$

The clause (Equation 3.4) is not in CNF format, so we use transformations such as DeMorgan's laws to convert it into CNF.

$$(\overline{x}\overline{y}+z) \tag{3.5}$$

$$(\overline{x}+z)(\overline{y}+z) \tag{3.6}$$

The CNF formula G_i is the conjunction of the above clauses (Equations 3.3, 3.6) and is written as:

$$G_i = (\bar{z} + x + y) \cdot (\bar{x} + z) \cdot (\bar{y} + z) \tag{3.7}$$

The CNF for the entire circuit S obtained from steps 1) and 2) above is constructed by forming the conjunction of all CNF formulas for all the gates in the circuit. If there are n gates in the circuit, then the CNF formula for the entire circuit is written as shown in Equation 3.8.

$$S = \prod_{i=1}^{n} G_i \tag{3.8}$$

In logic synthesis, the inputs to a predictor or a function are alo called the support. In general, the total number of possible functions is 2^{2^N} for a gene with N inputs. However, we consider only those functions that have a *true support* of N inputs. The true support of a function are the inputs that a function is actually dependent on. In our method, we require that a function must depend on all inputs specified by the predictor.

For example, with a gene x with 2 inputs y and z, and we are considering the function $x = yz + y\bar{z}$. Since this function can be simplified to $x = y$, it depends on only one input, and is not a true support of the two inputs and will be disregarded in our method. The function $x = yz$ does depend on both inputs, and thus would be considered in our method. The total number of functions F_N with true support of N inputs can be calculated as shown in Equation 3.9.

$$F_N = 2^{2^N} - \binom{N}{N-1} F_{N-1} - \binom{N}{N-2} F_{N-2} - \ldots - \binom{N}{0} F_0 \tag{3.9}$$

$$F_0 = 2^{2^0} = 2 \tag{3.10}$$

III-C.1.b. Constraining SAT Solution Space Using Gene Expression States

As is, the circuit of the previous step describes all possible BNs. To constrain the solution space to obtain one or a subset of BNs for our GRN, we constrain the circuit to make it satisfy gene expression states. A gene expression state is a measurement of the dynamic behavior of the GRN, containing information of the gene state at a time point t, as well as at the next time point $t+1$. In the state transition table (an example is shown in Table III.1), a gene expression state is a minterm (row) on the truth table of the table, and consists of a pair of states (S_1, S_2). This mandates that if the GRN is in state S_1 at time t, it will transition to state S_2 at time $t+1$. Note that table III.1 shows all possible minterms or observations of the GRN. However in practice, we may only have a limited number of observations. Our method determines a SAT solution that satisfies the predictor set and limited number of gene observations.

The overall goal of our approach is, for each gene, to select a function which matches all gene expression states (minterms of the state transition table). In our approach, if there are M minterms, we duplicate the circuit M times. Each circuit copy is assigned a minterm, with the gene values fixed according to the minterm. The select signals for all the MUXes for any gene x_i are connected together in each of the M copies of the circuit, to ensure that the same function for x_i is selected in all the M circuit copies.

The solution is an assignment of the variables in S such that S is satisfied. The assignment of the variables corresponding to the MUX select lines for any gene x_i denote which function was selected for the gene x_i and hence specifies a Boolean network. Depending on the gene expression states used, there may be more than one valid solution, in which case performing an All-SAT will generate *all* possible BNs that match the gene expression observations.

Alternatively, the method can be done on a single copy of the circuit. Each minterm is tested in order, and the circuit is solved using All-SAT to find all results that satisfy the i^{th} minterm. The conjunction of these results and S form a new circuit S before the $(i+1)^{th}$ minterm is processed. This computation was found to require significantly more runtime than the circuit duplication method.

III-C.1.c. Example

To illustrate the method, we consider a small 3 gene example. Let us label the genes in our example Boolean network as genes a, b, and c, with the following gene logic functions:

$a' = \overline{b} + c$

$b' = \overline{ac}$

$c' = a + b$

Current state			**Next state**		
a	b	c	a'	b'	c'
0	0	0	1	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	1	0	1

Table III.1. Example 3-Gene State Transition Table

In the notation, a', b', c' are the next state variables of a, b, c respectively. From these functions, we can derive a state transition table (Table III.1) which describes the next state in the BN given a present state. The predictors for each gene of our example are shown in Figure III.1a), and listed as follows: $p_a = \{b, c\}$, $p_b = \{a, c\}$, and $p_c = \{a, b\}$

A predictor $p_i = \{j, k, \ldots\}$ lists the set of genes $\{j, k, \ldots\}$ which predicts (regulates) the activity of gene i. Note that in our problem, the logic functions for a', b', and c' are unknown, and are to be determined from gene expression observations. The gene expression observations that are provided are any subset of the 8 rows of Table III.1.

Figure III.1 demonstrates the circuit construction. Focusing on gene a, we note it has 2 inputs b and c, meaning there are $F_2 = 10$ possible functions $(g_{a_1}, g_{a_2}, \ldots, g_{a_{10}})$ with a true support of 2 inputs. After enumerating the 10 functions, a MUX and select signal s_a is added to select exactly one of the function outputs, as shown in Figure III.1b). A similar construction is performed for genes b and c, with MUXes and select signals s_b and s_c respectively.

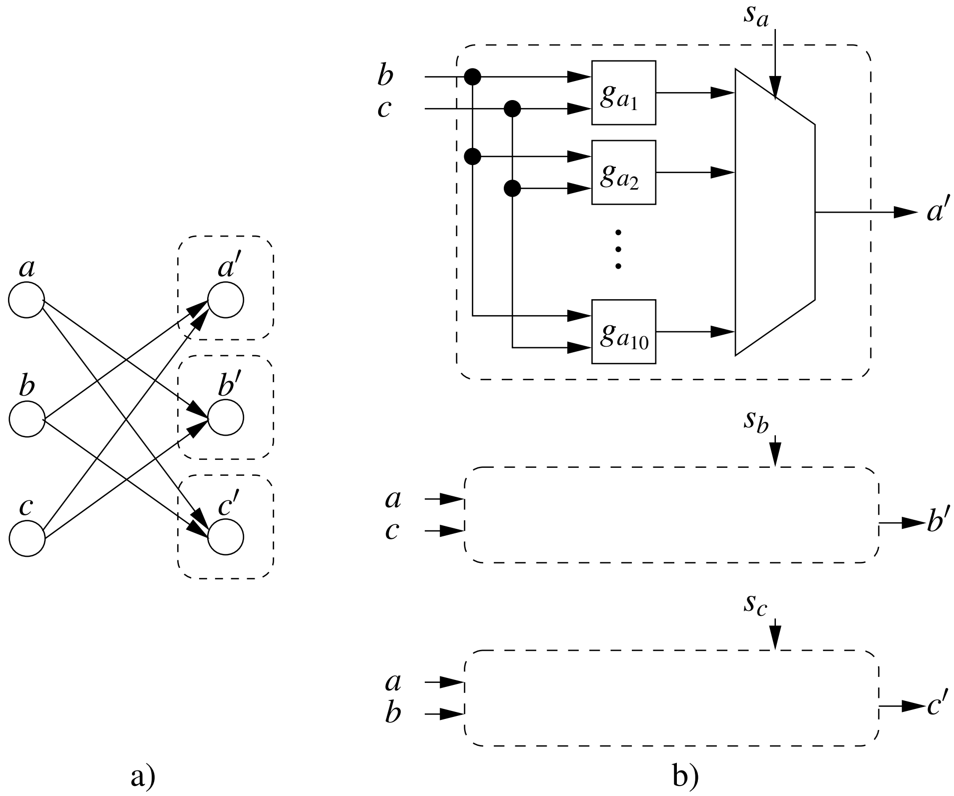


Fig. III.1. Circuit construction example: a) predictor set shown as connectivity graph and b) function enumeration and MUX for gene *a* shown in detail

In the example, let us assume that we have the following 3 gene expression states $\{a,b,c,a',b',c'\} \in \{(001110),(110001),(101101)\}$. We duplicate the circuit 3 times, assigning each copy one of the gene expression states. Accordingly, the MUX select signals for gene *a* are connected together across all 3 copies of the circuit, as are the MUXes for genes *b* and *c*.

Generally, we may have a limited number of gene expression states (in the example, we had 3 out of a possible 8 states). In such situation, there can be several BNs which match the observations as multiple sets of functions can be valid for the input. Using All-SAT will implicitly generate all possible satisfying BNs. From the 3-gene example with only 3 gene expression states, there are 4 possible solutions, of which one solution corresponds to the correct BN. To narrow the search, the results can be pruned using curated, partial, or prior biological information. For example, gene expressions or pathway information for some genes may be known from other research or databases. Or biologists may want to reason on the networks assuming the presence of specific gates or transitions on some genes (for example, gene *a* represses gene *b*). This new information restricts the solution space by providing our method additional logical constraints, and can be added to our algorithm using the same exact steps as described in our approach.

Additionally, our method can detect logical problems in the input data. If the CNF *S* is UNSAT (not satisfiable), there is no possible assignment of logic functions that satisfies the gene expression observations and the predictor set. This result may occur if there is an error in either the gene expression observations or predictor set. For example, if the predictor set was inferred wrongly, the gene expressions were measured incorrectly, or gene expression data from a different GRN were added, can produce an UNSAT result. In such situation, our method can be rerun on a modified predictor set, or the gene expression data can be analyzed to determine which genes is causing the logical error.

Let us examine an example with an UNSAT result. Consider that we have a gene *a* which is predicted by *b* and *c*. Let us assume that we are given gene expression observations $\{a,b,c,a',b',c'\} \in \{(000000),(000100)\}$. We observe that in the first observations (000000), $b = 0$ and $c = 0$, with $a' = 0$. However, in the second observation (000100), $b = 0$ and $c = 0$, with $a' = 1$. Both *b* and *c* have the same values in these two observations, but a' has a different value. Logically, this cannot occur since a function cannot have two different outputs for the same input. The CNF as constructed contains all valid Boolean functions and since no such Boolean function exists for this logical conflict, the result is UNSAT.

III-D. Experimental Results

III-D.1. Model Implementation

We evaluate the SAT-based method for determining the BN on two GRNs, one synthetic (randomly generated) and one real (*p53* network [41, 52]). We first investigate the senstivity of our method regarding the number of available gene expression observations, and then we demonstrate our method on attractor data from the $p53$ network.

The *p53* network is well-studied in genomics and medicine, due to the involvement of *p53* gene in many human cancers. $p53$ is a tumor suppressor gene and is a transcription factor for many downstream genes involved in controlling cell cycle, repairing DNA damage, and inducing apoptosis (cell death) for example. The main pathways for $p53$ [53] involve DNA damage in the form of breaks in the DNA strand, as shown in Figure III.2 In the figure, forward arrows represents activation, while arrows with a line represents repression. The presence of the external signal dna_dsb (DNA strand break damage) activates ATM, which in turn represses $Mdm2$, allowing for activation of $p53$. The expression of $p53$ blocks replication of DNA (a necessary response when DNA is damaged). From these pathways, [41] obtained the corresponding Boolean functions.

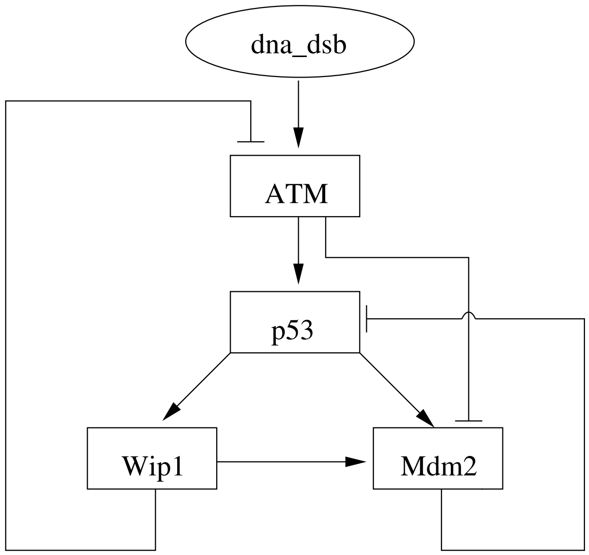


Fig. III.2. *p*53 pathways

In our experiments, the function of each gene in these networks is known, but hidden from our algorithm. We extract both the predictor set and gene expression observations to test our algorithm with. The regulating logic functions of the synthetic and p53 GRNs are shown in Tables III.2 and III.3 respectively (these are kept hidden from our algorithm).

Gene	Regulating Function
x_1	$x_2+x_4+x_5$
x_2	$\overline{x_1x_3x_5}$
x_3	$x_1x_2x_4$
x_4	$\overline{x_1}(x_3\overline{x_5}+\overline{x_3}x_5)+x_1(x_3x_5+\overline{x_3x_5})$
x_5	$\overline{(x_1+x_2+x_3)}$

Table III.2. Boolean Regulating Functions for **Synthetic** 5-Gene Network

Gene	Regulating Function
dna_dsb	(DNA damage is an external signal)
ATM	$\overline{Wip1}(ATM+dna\_dsb)$
$p53$	$\overline{Mdm2}(ATM+Wip1)$
$Wip1$	$p53$
$Mdm2$	$\overline{ATM}(p53+Wip1)$

Table III.3. Boolean Regulating Functions for **p53** Network

Our method uses an open-source and efficient exact SAT-solver, MiniSAT v1.14 [23]. Shell scripts were created to invoke MiniSAT and to implement the All-SAT functionality. All tests were implemented and run on a Core 2 Duo Mac OSX machine with 4 GB ram. Runtimes depend on the input predictor set and number of gene expression observations, but in our tests, each SAT operation is less than 1s. Accordingly, All-SAT runtime takes approximately n seconds for n satisfying solutions.

III-D.2. Method Sensitivity to Input

To investigate how the number of solutions (the number of satisfying BNs) depends on the number of available gene expression observations, we measure the sensitivity of our SAT algorithm in the following manner. For a GRN with n genes, there are 2^n gene expression states in total, which completely determines the GRN. To test the sensitivity of the number of solutions to the number of gene expressions observations i, we randomly select i gene expressions from the 2^n total, and run our algorithm to see how many surviving solutions there are. Because the number of surviving solutions can change depending on the specific gene expressions selected, we resample x times, and find the mean number of satisfying solutions among the x samples. We repeat this process for different values of i between 1 and 2^n.

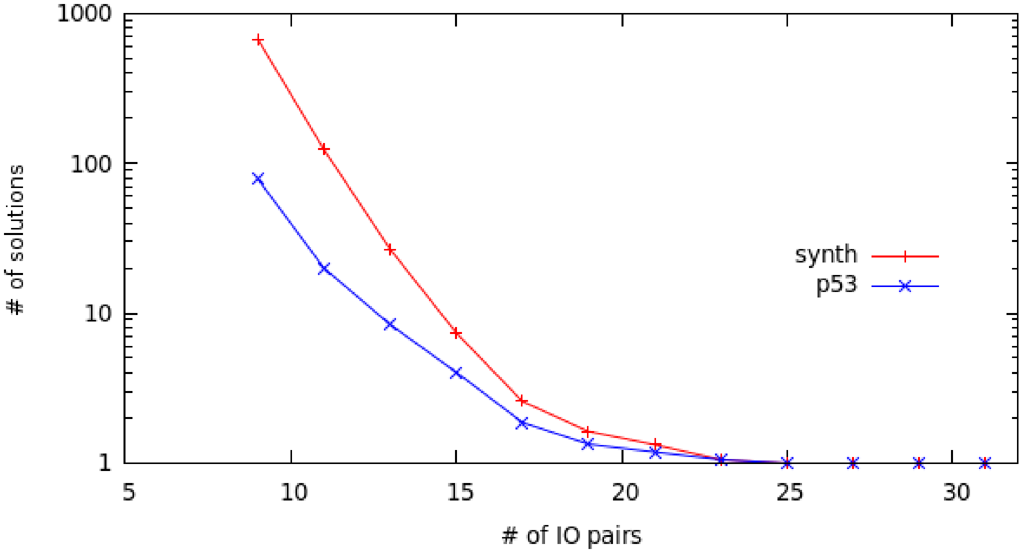


Fig. III.3. Plot of # of mean solutions vs # of gene expressions observations (IO pairs)

In Figure III.3, we show the sensitivity of the algorithm by plotting of the average number of satisfying solutions against i (the number of gene expression observations). For each value of i, we re-sampled $x = 100$ times, and all satisfying solutions were recorded. The mean number of solutions is plotted.

From the plot, we observe that as additional gene expressions are included in the algorithm, the solution space reduces exponentially until only a few surviving solutions remain. At this point, adding more gene expressions do not significantly change the size of the solution space. In both examples, the inflection point appears to be $i = 16$ (roughly half the total number of gene expression observations). These plots show the importance of including additional gene expressions in reducing the size of the solution space.

These results show that our method works well for GRNs with fewer genes. For a network with large number of genes, a corresponding large number of gene expression observations is needed to reduce the solution space and keep the computation under control.

An advantage is that our method is inherently parallelizable. By cofactoring[2] on x and $\overline{x}$, where x is a variable of S, we can partition S into 2 problems, S_x and $S_{\overline{x}}$. Each of these can run in parallel on separate machines. In general, we may partition S into 2^k partitions (by using k variables) and run each partition in parallel.

III-D.3. Function and BN Results for p53

We validate our SAT algorithm using the *p53* network. Let us assume that we have the attractor states as input to our algorithm. Using attractor states is a reasonable assumption since in the long run, a BN would transition to these attractor states, thus these states are most likely to be measured in practice. From the logic function of the *p53* network, we observe 2 attractor cycles containing 8 attractor states in total. We define the state space as $[ATM, p53, Wip1, Mdm2]$ and the attractor cycles are a singleton cycle if $dna\_dsb = 0$ and a 7 state cycle if $dna\_dsb = 1$ as shown Table III.4.

dna_dsb	Attractor Cycle
0	(0000)
1	(1000) → (1100) → (1110) → (0110) → (0111) → (0011) → (0001)

Table III.4. Attractor Cycles and States for $p53$ Network $[ATM, p53, Wip1, Mdm2]$

These attractors become the 8 gene expressions used as input to our method and All-SAT on the CNF results in 72 possible satisfying BNs out. Furthermore, we observe that one of the 72 BNs has the correct logic function for the *p53* network. If we count the number of selected functions per gene, we find that for $ATM, p53, Wip$, and $Mdm2$, there are 6, 4, 1, and 3 functions respectively. These results can help biologists tune their experiments to understand gene regulatory function.

[2] The *cofactor* of $S(x_1, \ldots x_i \ldots x_n)$ wrt x_i is $S_{x_i}(x_1, \ldots x_i \ldots x_n) = S(x_1, \ldots x_i = 1 \ldots x_n)$

III-E. Chapter Summary

In this chapter, we have presented an efficient and general SAT-based method for determining logic functions from gene expression data. Our approach implicitly explores all possible logic functions for each gene based on the predictor set, and selects functions that match the gene expression observations using a SAT formulation. Each SAT solution is a Boolean network, and the results of our method generate a family of BNs that match the predictor set and gene expressions. Our SAT-based method is validated on two GRNs and demonstrates the importance of gene expression data with regards to constraining the space of satisfying BNs. We also test the method on the *p53* network and show how our results can be used to select the gene functions. Due to its generality and efficiency, this algorithm can easily be extended to large networks, and can be augmented to utilize gene expression data from multiple sources.

Thus far, our inference of the GRN predictor set and regulating function has been done using binary valued gene expression data. In practice, gene expressions are initially measured as continuous values, from which the values are converted to binary values. While binary gene expressions simplify our analysis, continuos gene expression may provide a richer and more detailed observation of the gene state and GRN. The next chapter explores methods to infer the GRN from continuous gene expression data.

CHAPTER IV

PREDICTOR RANKING USING MODIFIED ZHEGALKIN FUNCTIONS

Inference of the underlying gene regulatory network structure (i.e. predictors and functions) from gene expression is an important challenge in genomics. With continuing improvements in microarray technology, the ability to measure expression levels of many genes has improved significantly, making available large amount of gene expression data for analysis. In previous chapters, all gene expressions have been assumed to be digital in nature. However, actual gene expressions (from microarrays for example) are continuous. On the other hand, many genes have been observed to exhibit switch-like or Boolean behavior. In this chapter, we utilize Zhegalkin polynomials to express the Boolean behavior of gene expression in an analog or continuous manner. Given gene expression data in the form of microarray measurements normalized to the unit interval, we present a method for ranking and selecting predictors which fits the data with the least mean square error according to the Zhegalkin function. Our methods are validated on synthetic gene expressions from a mutated mammalian cell-cycle network and then demonstrated on measured gene expressions from a melanoma network study. The results of our approach can be used to identify potential genes in future expression experiments or for possible targeted drug development experiments.

IV-A. Background and Previous Work

Advances in microarray technology have allowed biologists the opportunity to measure the expression of thousands, or even tens of thousands of genes simultaneously. This large amount of gene expression data can be used for analysis for modeling and inferring the gene regulatory network. Several methods have been proposed to model the expression data, particularly Boolean networks which use binary (Boolean) representation for gene expression. Boolean networks (BNs) [12] is commonly used for GRN inference [38, 41, 27] and intervention [42, 30]. In the Boolean network, gene expression are binary valued, either 1 (ON) or 0 (OFF). Binary value representation is used as many genes have been observed to exhibit switch-like behavior. In the Boolean network, gene expressions are updated at the following time point according to Boolean functions at the current time point. The deterministic nature of Boolean network allows for fast analysis and application of logic synthesis tools. While Boolean networks exhibit many observed characteristics of gene regulatory networks, Boolean networks cannot model continuous levels of gene expression values.

In context of RNA and protein production, actual gene expression is more complex and is measured as a continuous value from measurement techniques such as microarrays. Other models have been proposed to model the GRN with continuous value gene expression such as Differential Equations [10], Linear Equations [11], Continuous Networks [14], and Stochastic Gene Networks [15]. While such models can determine continuous functions to model the gene expresion data, continuous functions in general cannot capture the Boolean-like behavior of genes.

In [54], a model was proposed to combine continuous gene expression and discrete Boolean-like behavior. This combined model is based on Zhegalkin polynomial functions [55]. Zhegalkin functions is an alternative representation of Boolean functions using continuous values. These Zhegalkin functions can represent any Boolean function having an output value within the unit interval [0,1] if input variables are also within the unit interval. In [54], it was demonstrated how Zhegalkin function can be used to model the next state equation for a given a predictor (target gene and input genes), and time-series expression data for yeast model dataset.

Our approach uses Zhegalkin functions to infer gene predictors and functions from normalized continuous gene expression data. As opposed to [54] which uses a linear expression function, our method uses a sigmoid expression function to more accurately represent the gene expression. Another key difference, [54] only finds a single regulating function for a given predictor and gene expression data, while our methods finds the best predictor and function for a target gene given just the gene expression data, by searching across all possible predictors and functions in the GRN through a ranking of best fitting predictors by mean-squared error.

IV-B. Approach

IV-B.1. Network Model

As described earlier, the Boolean network model can not be used with continuous gene expression values. Instead we use a modified model similar to BN but which uses continuous expression values and Zhegalkin functions (subsection IV-B.2) in place of Boolean values and Boolean functions. In the modified model, we define a set of nodes $\{x_1, x_2, \ldots, x_n\}$ and Zhegalkin functions $\{z_1, z_2, \ldots, z_n\}$. Each node x_i is a gene, and each gene is associated with a Zhegalkin function z_i. The value of a gene is a continuous variable, $x_i \in [0, 1]$ where the value can be within the unit interval, and is updated according to the associated Zhegalkin function z_i. Thus, the gene state x_i can represent varying levels of expressions from fully expressed $(x_i = 1)$, not expressed $(x_i = 0)$, and any expression level in between.

IV-B.2. Zhegalkin Polynomial Function

To model the dynamics of continuous gene expression values and provide a continuous representation of Boolean function, our algorithm utilizes Zhegalkin polynomial functions [56]. Following is a description of a Zhegalkin polynomial function.

Definition IV.1: A **Zhegalkin polynomial function** with n variables is given by Equation 4.1.

$$f(x_1,\ldots,x_n) = a_0 + \sum_{j=1}^{n} a_j x_j + \sum_{k=2}^{n}\sum_{j=1}^{k-1} a_{jk} x_j x_k + \sum_{l=3}^{n}\sum_{k=2}^{l-1}\sum_{j=1}^{k-1} a_{jkl} x_j x_k x_l + \ldots + a_{1\ldots n} x_1 \ldots x_n \tag{4.1}$$

The Zhegalkin function is a linear function consisting of coefficients and products of the input variables. The first term, a_0 is a constant. The second term $\sum_{j=1}^{n} a_j x_j$ is weighted sum of all possible single inputs. The third term $\sum_{k=2}^{n}\sum_{j=1}^{k-1} a_{jk} x_j x_k$ is weighted sum of all possible combinations of two inputs, and so on. The last term is a weighted product of all inputs.

The coefficients or weights $a_0, a_1, \ldots, a_{1\ldots n}$ of a Zhegalkin function are called Zhegalkin coefficients. In general, any Boolean function can be converted to a Zhegalkin function by selecting the appropriate Zhegalkin coefficients. In [56], it was determined that the possible range of values for the Zhegalkin coefficients to define any Boolean function of input size n are listed in Table IV.1.

Coefficient	Set of Possible Values	Notation
a_0	0,1	A_0
a_j	-1,0,1	A_1
a_{jk}	-2,-1,0,1,2	A_2
a_{jkl}	-4,-3,-2,0,1,2,3,4	A_3
$\vdots$	$\vdots$	$\vdots$
$a_{123\ldots n}$	$-2^{(n-1)},\ldots,-1,0,1,\ldots,2^{(n-1)}$	A_n

Table IV.1. Possible Values for Coefficients of Zhegalkin Function

To demonstrate how Zhegalkin function can represent a Boolean function, we show two simple examples. Consider two Boolean functions:

$$f^B = \overline{x_1} x_2$$

$$g^B = \overline{x_1}\,\overline{x_2} + x_1x_2$$

The truth tables for f^B and g^B are shown in Tables IV.2 and IV.3. We select the appropriate coefficients to find the corresponding Zhegalkin functions are:

$$f^Z = x_2 - x_1x_2$$

$$g^Z = 1 - x_1 - x_2 + 2x_1x_2$$

Plotting the Zhegalkin functions as a surface plot, we observe in Figures IV.1 and IV.2 that the corner points (where the inputs are 0 or 1) match the Boolean function output and the surface confirms to expected values for continuous Boolean function in the unit interval.

x_1	x_2	f^B
0	0	0
0	1	1
1	0	0
1	1	0

Table IV.2. Truth Table for $f^B = \overline{x_1}x_2$

x_1	x_2	g^B
0	0	1
0	1	0
1	0	0
1	1	1

Table IV.3. Truth Table for $g^B = \overline{x_1}\,\overline{x_2} + x_1x_2$

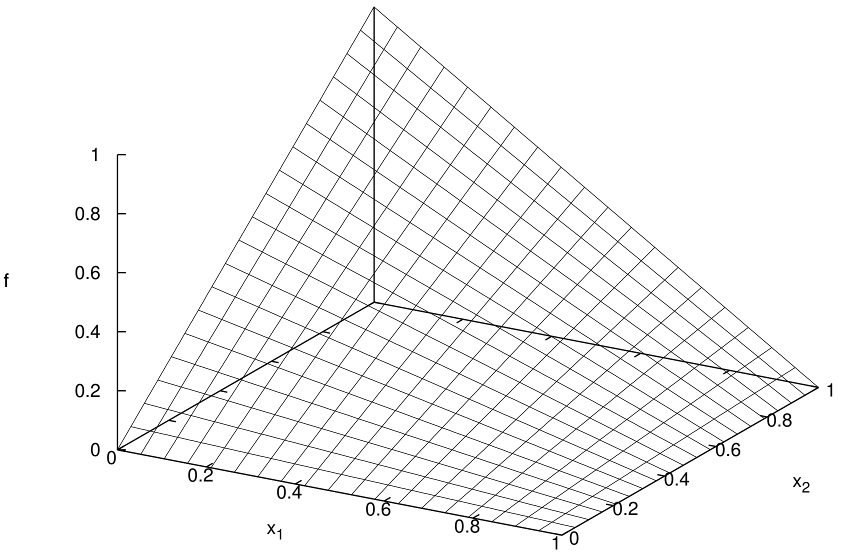


Fig. IV.1. Zhegalkin function $f^Z = x_2 - x_1 x_2$ for Boolean function $f^B = \overline{x_1} x_2$

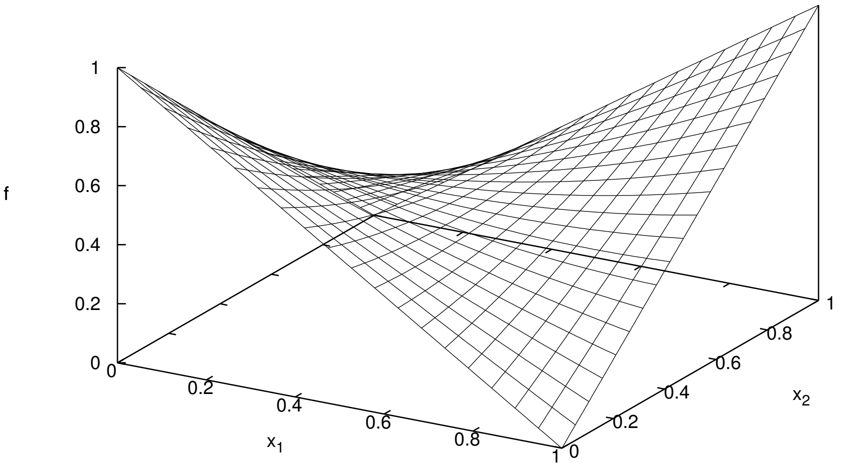


Fig. IV.2. Zhegalkin function $g^Z = 1 - x_1 - x_2 + 2x_1 x_2$ for Boolean function $g^B = \overline{x_1}\,\overline{x_2} + x_1 x_2$

IV-B.3. Sigmoid Function

The inputs of the Zhegalkin function as used by [54] are continuous gene expression values represented in a linear function. However, actual gene expression is more complex and depends on several simultaneous and competing factors such as RNA/protein formation or degradation, chemical reaction rates, and molecular transport. It has been observed that the expression values tends to saturate at low and high gene activity and has been suggested that a more accurate representation of continuous gene expression is a sigmoid function [57, 58].

Definition IV.2: A **sigmoid function** has a ”S” shape curve defined by Equation 4.2.

$$s(t) = 1/(1+e^{-t}) \tag{4.2}$$

Figure IV.3 plots the linear and sigmoid function to compare the two functions. Note, the sigmoid function $s(x)$ is shifted and scaled to the unit interval.

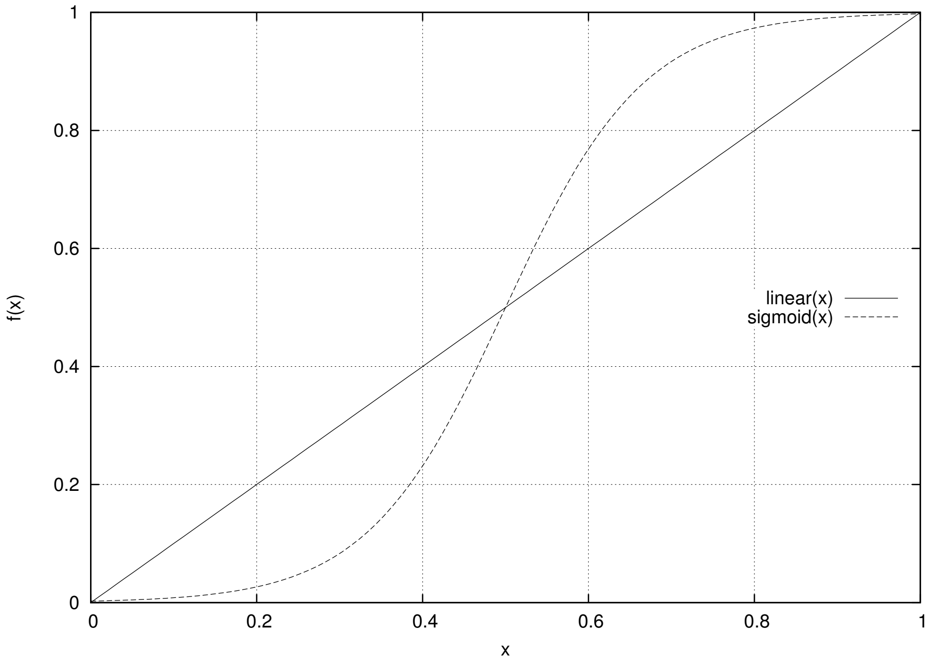


Fig. IV.3. Linear function x compared with sigmoid function $s(x) = 1/1 + e^{-12*x+6}$ over the unit interval

In our approach, the Zhegalkin function can accept either the linear x or sigmoid $s(x)$ representation of gene expression. In section IV-C, we will compare the accuracy between the two representations.

IV-B.4. Predictor Ranking Algorithm

Algorithm 1 describes the general procedure for our predictor ranking. The inputs to our algorithm are gene expression observations X and Y with m observations or samples of each. In detail, X_i is the observed current state of the network, and Y_i is the resulting next state of the network. X can be the linear representation or sigmoid representation as specified at runtime. Given a target gene x_i, our algorithm determines a ranking of the best predictors for x_i with minimal error in the expected output (best fitting Zhegalkin function upon X) and the actual output Y.

In line 2 of the algorithm, the method iterates through all possible predictor combinations $p_j \in p$ for the target gene x_i. For each p_j, the method considers all valid Zhegalkin functions $z_k \in z$ (line 3) by iterating over all possible coefficient combinations corresponding to Boolean functions. In line 4, the algorithm determines the mean squared error (MSE) of the expected output and actual output for each Zhegalkin function z_k. The MSE is used to measure how well the Zhegalkin function fits or matches the actual output expression values. In line 6, the minimum MSE of all Zhegalkin functions for predictor p_j is chosen as representative MSE for p_j. After the MSEs for all predictors have been determined, the predictors are sorted by their MSE (line 8), and the ranked predictors (as well as its Zhegalkin function) and corresponding MSE are returned.

Algorithm 1 Pseudocode of Predictor Ranking

```
1: $PRED\_RANK(X,Y,x_i)$
2: for all predictor combinations $p_j \in p$ for $x_i$ do
3:     for all valid Zhegalkin functions $z_k \in z$ for $p_j$ do
4:         $MSE_{z_k} = 1/m \sum_{l=1}^{m} (z_k(X_l) - Y_l)^2$
5:     end for
6:     $MSE_{p_j} = \min(MSE_z)$
7: end for
8: sort $p$ by $MSE_p$
9: return ranked $(p, MSE_p)$
```

The algorithm returns a ranked list of predictors rather than a single predictor with the lowest MSE for a several reasons. The main reason is the actual or correct predictor is expected to be one the top ranked predictors as the actual predictor should best match the input data resulting in low MSE. However, the ”correct” predictor may not have the lowest MSE if the expression samples are not adequately distributed. Ideally, the samples should be uniformly distributed throughout the state space. However, limited number of samples may result in some areas in the state space not adequately represented. For example, if none of the samples were represented in a particular region, several Zhegalkin functions or predictors may match equally well. Second, the expression data may be noisy or may contain errors, resulting in a higher MSE for the ”correct” predictor, and in turn potentially decreasing its rank. The ranked list ensures that the ”correct” predictor is not prematurely disqualified from the results.

In general, given adequately distributed expression samples, the top ranking predictor can be selected as the inferred predictor, if the top ranked predictor has significantly lower MSE than the second ranked predictor. In the next section, we describe one method for selecting a predictor from the ranked list. Otherwise, if several predictors have similarly low MSE (due to sample distribution or noisy data), the ranked list can be used to help guide follow-up lab experiments to test and verify those particular predictors.

IV-C. Results

We demonstrate our predictor ranking method on two GRNs. To validate our method, we use the mutated 9-gene mammalian cell-cycle network using synthetic gene expression data. We use both linear and sigmoid representation for gene expression values. From these results, we determine a predictor selection method and find the sigmoid representation is more accurate. Lastly, we apply both ranking and selection method on melanoma study data assuming a sigmoid representation.

IV-C.1. Mutated Mammalian Cell-Cycle Network

In this experiment, we use a mutated mammalian cell-cycle network to illustrate and validae our approach. For a normal mammal, the cell cycle is tightly controlled through extracellular signals that indicate whether a cell should divide/grow or not. These signals activate the gene CyclinD ($CycD$) which is a key gene in mammalian cell-cycle. Another important gene is retinoblastoma (Rb) which is a tumor-suppressor when the other cyclin genes are not expressed. Another key gene is $p27$, which when active, represses the cyclin genes, stopping the cell cycle. In the mutated mammalian cell-cycle, $p27$ is mutated and is always off, leading to possible cell cycle in the absence of extracelluar signals. For the mutated 9-gene mammalian cell-cycle network, [59] determined the regulating functions for genes to be those shown in Table IV.4. To validate our method, we will use the regulating functions to create synthetic continuous gene expression values, on which we apply our algorithm (linear and sigmoid) to determine predictor rankings for target genes in the mutated network. In this setup, the actual functions and predictors are hidden from our algorithm.

	Gene	Regulating Function
x_1	$CycD$	extracellular signal
x_2	Rb	$\overline{CycD} \cdot \overline{CycE} \cdot \overline{CycA} \cdot \overline{CycB}$
x_3	$E2F$	$\overline{Rb} \cdot \overline{CycA} \cdot \overline{CycB}$
x_4	$CycE$	$\overline{E2F} \cdot \overline{Rb}$
x_5	$CycA$	$(E2F \cdot \overline{Rb} \cdot \overline{Cdc20} \cdot \overline{(Cdh1 \cdot UbcH10)}) + (CycA \cdot \overline{Rb} \cdot \overline{Cdc20} \cdot \overline{(Cdh1 \cdot UbcH10)})$
x_6	$Cdc20$	$CycB$
x_7	$Cdh1$	$(\overline{CycA} \cdot \overline{CycB}) + Cdc20$
x_8	$UbcH10$	$\overline{Cdh1} + (Cdh1 \cdot UbcH10 \cdot (Cdc20 + CycA + CycB))$
x_9	$CycB$	$\overline{Cdc20} \cdot \overline{Cdh1}$

Table IV.4. Boolean Regulating Functions for Mutated 9-Gene Mammalian Cell-Cycle Network

To synthesize normalized and continuous gene expression data similar to those measured in practice, we perform the following procedure. From the Boolean functions in Table IV.4, we create a state transition (truth) table listing all current states and next states. Each pair of current and next state forms a minterm (row) in the table. Since there are $n = 9$ genes in the mutated network, the state transition table contains $2^9 = 512$ minterms. We randomly sample m minterms and convert the binary values of each gene to a continuous value. The conversion process takes a binary value (0,1) and uniformly and randomly perturbs the value up to p, resulting in a continuous value $([0,p],[1-p,1])$. The value of p can be from 0 to 0.5 and is proportional to the number of occurrences of a binary value for a gene in the set of minterms. For example, if a gene x_i has the value 1 occuring 75% in the set of minterms, $p = (0.75) * (0.5) = 0.375$, and as such the 1 value is perturbed from [0.625,1] for gene x_i. Each gene will have a different perturbation that is dependent on the occurances of the binary values 1 and 0 in the the input set.

As an additional constraint to improve runtime, we limit our algorithm to search on predictors with 4 or less inputs. In general, this is a reasonable assumption as most genes have been observed to have relatively few inputs. We individually select genes x_2 to x_9 as the target gene and then apply our method on the mutated network to determine predictor rankings for each of these 8 genes. We exclude gene $CycD(x_1)$ as it is an extracellular signal, and thus not predicted by any genes in the mutated cell-cycle network.

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_1	0.1626	x_1,x_4	0.0676	x_1,x_4,x_9	0.0501	x_1,x_4,x_8,x_9	0.0365
2	x_8	0.2268	x_1,x_3	0.0952	x_1,x_3,x_4	0.0512	x_1,x_4,x_6,x_9	0.0377
3	x_7	0.2323	x_1,x_6	0.1030	x_1,x_4,x_5	0.0561	x_1,x_3,x_4,x_8	0.0385
4	x_3	0.2398	x_1,x_9	0.1037	x_3,x_4,x_7	0.0592	x_1,x_3,x_4,x_9	0.0397
5	x_6	0.2425	x_3,x_4	0.1058	x_1,x_4,x_6	0.0623	x_1,x_4,x_5,x_6	0.0405
6	x_4	0.2482	x_5,x_6	0.1062	x_3,x_4,x_5	0.0626	x_1,x_4,x_5,x_9	0.0415
7	x_5	0.2517	x_1,x_8	0.1083	x_1,x_4,x_7	0.0660	x_1,x_4,x_5,x_8	0.0445
8	x_9	0.2572	x_1,x_7	0.1086	x_3,x_4,x_9	0.0698	x_1,x_3,x_4,x_5	0.0446
9			x_1,x_5	0.1111	x_1,x_3,x_5	0.0705	x_1,x_4,x_7,x_9	0.0450
10			x_4,x_7	0.1251	x_3,x_4,x_8	0.0710	x_1,x_3,x_4,x_6	0.0454

Table IV.5. Linear Predictor Ranking by MSE for Gene $Rb(x_2)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_1,x_4,x_5,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_7	0.1916	x_2,x_9	0.0770	x_2,x_5,x_9	0.0532	x_2,x_5,x_8,x_9	0.0354
2	x_2	0.2007	x_2,x_5	0.0778	x_2,x_5,x_6	0.0672	x_2,x_5,x_6,x_7	0.0409
3	x_5	0.2015	x_5,x_9	0.1127	x_2,x_4,x_9	0.0677	x_2,x_5,x_6,x_9	0.0412
4	x_1	0.2033	x_5,x_7	0.1158	x_2,x_4,x_5	0.0699	x_2,x_5,x_7,x_9	0.0424
5	x_9	0.2051	x_2,x_7	0.1184	x_2,x_5,x_8	0.0717	x_2,x_4,x_5,x_9	0.0438
6	x_4	0.2097	x_7,x_9	0.1210	x_1,x_2,x_9	0.0731	x_1,x_2,x_5,x_9	0.0446
7	x_8	0.2474	x_5,x_6	0.1217	x_2,x_5,x_7	0.0755	x_1,x_2,x_5,x_7	0.0473
8	x_6	0.2585	x_5,x_8	0.1263	x_1,x_2,x_5	0.0770	x_2,x_4,x_5,x_7	0.0478
9			x_1,x_2	0.1291	x_1,x_2,x_7	0.0788	x_2,x_5,x_6,x_8	0.0513
10			x_1,x_7	0.1308	x_1,x_4,x_7	0.0792	x_1,x_2,x_7,x_9	0.0525

Table IV.6. Linear Predictor Ranking by MSE for Gene $E2F(x_3)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_5,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_3	0.0852	x_2,x_3	0.0222	x_2,x_3,x_5	0.0258	x_1,x_2,x_3,x_7	0.0110
2	x_2	0.1510	x_3,x_8	0.0851	x_2,x_3,x_6	0.0316	x_2,x_3,x_6,x_9	0.0135
3	x_6	0.1816	x_2,x_6	0.0863	x_2,x_3,x_9	0.0322	x_2,x_3,x_5,x_6	0.0143
4	x_5	0.2048	x_3,x_6	0.1000	x_2,x_3,x_7	0.0342	x_1,x_2,x_3,x_5	0.0179
5	x_7	0.2171	x_1,x_3	0.1002	x_2,x_3,x_8	0.0387	x_1,x_2,x_3,x_9	0.0181
6	x_1	0.2418	x_3,x_9	0.1028	x_1,x_2,x_3	0.0416	x_2,x_3,x_7,x_8	0.0184
7	x_9	0.2436	x_3,x_7	0.1188	x_3,x_7,x_9	0.0614	x_2,x_3,x_7,x_9	0.0187
8	x_8	0.2508	x_3,x_5	0.1197	x_3,x_7,x_8	0.0628	x_2,x_3,x_5,x_8	0.0190
9			x_1,x_2	0.1220	x_3,x_5,x_8	0.0657	x_1,x_2,x_3,x_6	0.0198
10			x_7,x_9	0.1224	x_3,x_6,x_8	0.0657	x_2,x_3,x_8,x_9	0.0198

Table IV.7. Linear Predictor Ranking by MSE for Gene $CycE(x_4)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_3

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_6	0.1425	x_2,x_6	0.0581	x_2,x_6,x_7	0.0523	x_2,x_6,x_8,x_9	0.0336
2	x_3	0.1434	x_3,x_6	0.0713	x_2,x_6,x_9	0.0528	x_2,x_4,x_6,x_9	0.0343
3	x_7	0.1651	x_2,x_3	0.0822	x_3,x_6,x_8	0.0588	x_2,x_6,x_7,x_8	0.0360
4	x_9	0.1943	x_6,x_8	0.0839	x_2,x_6,x_8	0.0609	x_2,x_4,x_6,x_7	0.0369
5	x_2	0.2072	x_6,x_7	0.0877	x_2,x_3,x_6	0.0612	x_2,x_3,x_6,x_9	0.0370
6	x_8	0.2281	x_3,x_7	0.0880	x_6,x_7,x_9	0.0624	x_1,x_3,x_6,x_7	0.0376
7	x_4	0.2368	x_2,x_7	0.0976	x_3,x_7,x_8	0.0644	x_1,x_3,x_6,x_9	0.0381
8	x_1	0.2439	x_3,x_8	0.0995	x_3,x_6,x_9	0.0656	x_3,x_4,x_6,x_8	0.0410
9			x_3,x_9	0.1038	x_2,x_4,x_6	0.0670	x_3,x_6,x_8,x_9	0.0424
10			x_4,x_7	0.1105	x_6,x_7,x_8	0.0672	x_1,x_3,x_4,x_6	0.0425

Table IV.8. Linear Predictor Ranking by MSE for Gene $CycA(x_5)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_3,x_6,x_7,x_8

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_9	0.0126	x_1,x_9	0.0784	x_1,x_4,x_9	0.0171	x_3,x_4,x_7,x_9	0.0106
2	x_1	0.2219	x_4,x_9	0.0985	x_7,x_8,x_9	0.0176	x_3,x_4,x_8,x_9	0.0109
3	x_5	0.2279	x_2,x_9	0.1062	x_2,x_4,x_9	0.0186	x_4,x_7,x_8,x_9	0.0112
4	x_8	0.2416	x_7,x_9	0.1109	x_4,x_7,x_9	0.0208	x_1,x_3,x_7,x_9	0.0127
5	x_3	0.2434	x_8,x_9	0.1156	x_3,x_8,x_9	0.0277	x_2,x_4,x_8,x_9	0.0128
6	x_4	0.2608	x_5,x_9	0.1230	x_2,x_5,x_9	0.0305	x_4,x_5,x_8,x_9	0.0129
7	x_2	0.2653	x_3,x_7	0.1302	x_1,x_7,x_9	0.0315	x_3,x_7,x_8,x_9	0.0131
8	x_7	0.2689	x_3,x_9	0.1345	x_1,x_3,x_9	0.0319	x_1,x_5,x_8,x_9	0.0131
9			x_7,x_8	0.1439	x_1,x_8,x_9	0.0320	x_1,x_4,x_5,x_9	0.0133
10			x_2,x_7	0.1550	x_1,x_2,x_9	0.0322	x_1,x_3,x_5,x_9	0.0133

Table IV.9. Linear Predictor Ranking by MSE for Gene $Cdc20(x_6)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_6	0.1231	x_6,x_9	0.0565	x_5,x_6,x_9	0.0215	x_5,x_6,x_8,x_9	0.0173
2	x_9	0.1310	x_6,x_8	0.0965	x_1,x_6,x_9	0.0297	x_1,x_4,x_6,x_9	0.0244
3	x_8	0.1821	x_5,x_9	0.1017	x_4,x_6,x_9	0.0493	x_1,x_5,x_6,x_9	0.0245
4	x_3	0.2088	x_1,x_6	0.1023	x_5,x_6,x_8	0.0562	x_4,x_5,x_6,x_9	0.0252
5	x_1	0.2390	x_5,x_6	0.1048	x_6,x_8,x_9	0.0573	x_1,x_3,x_6,x_9	0.0257
6	x_4	0.2486	x_8,x_9	0.1093	x_1,x_3,x_6	0.0578	x_3,x_5,x_6,x_9	0.0272
7	x_5	0.2520	x_3,x_8	0.1166	x_4,x_6,x_8	0.0586	x_2,x_5,x_6,x_9	0.0286
8	x_2	0.2797	x_2,x_6	0.1168	x_2,x_6,x_9	0.0596	x_1,x_6,x_8,x_9	0.0309
9			x_3,x_6	0.1173	x_1,x_5,x_6	0.0656	x_4,x_6,x_8,x_9	0.0358
10			x_1,x_3	0.1214	x_3,x_6,x_9	0.0659	x_2,x_4,x_6,x_9	0.0362

Table IV.10. Linear Predictor Ranking by MSE for Gene $Cdh1(x_7)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_5,x_6,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_7	0.0981	x_1,x_7	0.1093	x_1,x_4,x_7	0.0681	x_1,x_3,x_6,x_7	0.0597
2	x_1	0.1924	x_3,x_4	0.1134	x_1,x_2,x_7	0.0694	x_3,x_5,x_6,x_7	0.0619
3	x_4	0.2168	x_3,x_5	0.1356	x_1,x_6,x_7	0.0744	x_4,x_6,x_7,x_9	0.0620
4	x_3	0.2476	x_3,x_6	0.1390	x_1,x_3,x_7	0.0749	x_1,x_4,x_6,x_7	0.0622
5	x_9	0.2487	x_6,x_7	0.1479	x_1,x_5,x_7	0.0751	x_1,x_3,x_4,x_7	0.0623
6	x_6	0.2546	x_7,x_9	0.1482	x_3,x_5,x_7	0.0768	x_1,x_3,x_5,x_7	0.0625
7	x_5	0.2559	x_4,x_5	0.1500	x_1,x_7,x_9	0.0781	x_1,x_2,x_5,x_7	0.0626
8	x_2	0.2565	x_4,x_7	0.1512	x_2,x_5,x_7	0.0822	x_2,x_3,x_4,x_7	0.0647
9			x_3,x_9	0.1531	x_3,x_4,x_7	0.0829	x_2,x_3,x_6,x_7	0.0650
10			x_3,x_7	0.1587	x_5,x_6,x_7	0.0831	x_3,x_5,x_7,x_9	0.0653

Table IV.11. Linear Predictor Ranking by MSE for Gene $UbcH10(x_8)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_5,x_6,x_7,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_7	0.1238	x_6,x_7	0.0241	x_6,x_7,x_8	0.0202	x_3,x_6,x_7,x_8	0.0176
2	x_6	0.1502	x_5,x_7	0.0813	x_5,x_6,x_7	0.0401	x_4,x_6,x_7,x_8	0.0182
3	x_3	0.2220	x_1,x_7	0.0957	x_1,x_6,x_7	0.0429	x_1,x_6,x_7,x_8	0.0189
4	x_8	0.2506	x_7,x_8	0.0995	x_4,x_6,x_7	0.0494	x_5,x_6,x_7,x_8	0.0204
5	x_4	0.2652	x_3,x_7	0.1002	x_3,x_6,x_7	0.0520	x_4,x_5,x_6,x_7	0.0206
6	x_5	0.2716	x_4,x_7	0.1064	x_2,x_6,x_7	0.0531	x_3,x_5,x_6,x_7	0.0207
7	x_2	0.2792	x_3,x_6	0.1096	x_2,x_3,x_7	0.0605	x_2,x_6,x_7,x_8	0.0207
8	x_1	0.2884	x_6,x_8	0.1165	x_3,x_4,x_6	0.0634	x_3,x_4,x_6,x_7	0.0208
9			x_4,x_6	0.1211	x_1,x_7,x_8	0.0657	x_1,x_5,x_6,x_7	0.0208
10			x_2,x_7	0.1264	x_5,x_7,x_8	0.0660	x_1,x_3,x_6,x_7	0.0216

Table IV.12. Linear Predictor Ranking by MSE for Gene $CycB(x_9)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_6,x_7

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_1	0.2722	x_1,x_4	0.1244	x_1,x_4,x_9	0.0404	x_1,x_4,x_7,x_9	0.04035
2	x_8	0.2815	x_5,x_8	0.1475	x_1,x_3,x_4	0.0646	x_1,x_4,x_8,x_9	0.04050
3	x_7	0.3732	x_1,x_3	0.1503	x_1,x_4,x_5	0.0673	x_1,x_4,x_5,x_9	0.04076
4	x_6	0.3899	x_1,x_5	0.1534	x_1,x_3,x_6	0.0745	x_1,x_4,x_6,x_9	0.04073
5	x_3	0.3901	x_5,x_6	0.1571	x_1,x_3,x_5	0.0922	x_1,x_3,x_4,x_5	0.05836
6	x_5	0.3923	x_1,x_6	0.1583	x_1,x_5,x_9	0.0926	x_1,x_3,x_4,x_8	0.06408
7	x_9	0.4013	x_8,x_9	0.1623	x_1,x_6,x_9	0.0951	x_1,x_3,x_4,x_9	0.06436
8	x_4	0.4068	x_7,x_8	0.1656	x_1,x_4,x_8	0.0951	x_1,x_3,x_4,x_7	0.06441
9			x_1,x_8	0.1689	x_3,x_4,x_9	0.0954	x_1,x_3,x_4,x_6	0.06525
10			x_3,x_6	0.1774	x_1,x_4,x_6	0.0982	x_1,x_4,x_5,x_7	0.06658

Table IV.13. Sigmoid Predictor Ranking by MSE for Gene $Rb(x_2)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_1,x_4,x_5,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_2	0.2993	x_2,x_5	0.0591	x_2,x_5,x_9	0.0064	x_2,x_5,x_7,x_9	0.0058
2	x_7	0.3034	x_2,x_9	0.1210	x_2,x_5,x_6	0.0592	x_2,x_5,x_8,x_9	0.0060
3	x_1	0.3600	x_2,x_7	0.1552	x_2,x_5,x_8	0.0694	x_2,x_5,x_6,x_9	0.0060
4	x_5	0.3627	x_1,x_2	0.1784	x_2,x_5,x_7	0.0698	x_1,x_2,x_5,x_9	0.0061
5	x_4	0.3699	x_5,x_7	0.1797	x_1,x_2,x_5	0.0851	x_2,x_4,x_5,x_9	0.0261
6	x_9	0.4052	x_7,x_9	0.1840	x_2,x_4,x_5	0.0857	x_1,x_2,x_5,x_7	0.0291
7	x_8	0.4236	x_5,x_6	0.1938	x_1,x_2,x_7	0.0871	x_2,x_4,x_5,x_7	0.0555
8	x_6	0.4286	x_4,x_7	0.1938	x_5,x_7,x_9	0.0983	x_2,x_4,x_5,x_8	0.0572
9			x_5,x_8	0.1985	x_1,x_2,x_6	0.1138	x_2,x_4,x_5,x_6	0.0580
10			x_4,x_5	0.2099	x_2,x_4,x_9	0.1211	x_1,x_2,x_5,x_8	0.0588

Table IV.14. Sigmoid Predictor Ranking by MSE for Gene $E2F(x_3)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_5,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_3	0.1540	x_2,x_3	0.0014	x_2,x_3,x_6	0.0252	x_1,x_2,x_3,x_7	0.0013
2	x_2	0.2384	x_2,x_6	0.1191	x_2,x_3,x_5	0.0329	x_2,x_3,x_5,x_8	0.0014
3	x_6	0.3028	x_3,x_8	0.1287	x_2,x_3,x_8	0.0350	x_2,x_3,x_5,x_6	0.0014
4	x_5	0.3538	x_1,x_3	0.1542	x_2,x_3,x_9	0.0597	x_1,x_2,x_3,x_5	0.0015
5	x_9	0.3688	x_3,x_6	0.1549	x_2,x_3,x_7	0.0606	x_1,x_2,x_3,x_9	0.0015
6	x_7	0.3712	x_3,x_7	0.1852	x_1,x_2,x_3	0.0615	x_1,x_2,x_3,x_6	0.0017
7	x_8	0.4165	x_3,x_9	0.1872	x_3,x_7,x_9	0.0915	x_2,x_3,x_6,x_8	0.0017
8	x_1	0.4243	x_1,x_2	0.1873	x_3,x_5,x_7	0.1111	x_2,x_3,x_7,x_9	0.0022
9			x_5,x_6	0.1913	x_1,x_3,x_8	0.1131	x_2,x_3,x_8,x_9	0.0023
10			x_3,x_5	0.1915	x_1,x_3,x_6	0.1176	x_2,x_3,x_7,x_8	0.0023

Table IV.15. Sigmoid Predictor Ranking by MSE for Gene $CycE(x_4)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_3

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_3	0.2598	x_2,x_6	0.0683	x_2,x_6,x_8	0.0376	x_2,x_3,x_6,x_8	0.0098
2	x_6	0.2626	x_6,x_8	0.0948	x_3,x_6,x_8	0.0388	x_2,x_6,x_7,x_8	0.0358
3	x_7	0.3031	x_3,x_6	0.1008	x_2,x_3,x_6	0.0397	x_2,x_6,x_8,x_9	0.0368
4	x_2	0.3443	x_3,x_7	0.1256	x_2,x_6,x_7	0.0408	x_3,x_6,x_7,x_8	0.0370
5	x_9	0.3452	x_4,x_7	0.1287	x_2,x_6,x_9	0.0417	x_1,x_2,x_6,x_8	0.0373
6	x_4	0.3531	x_2,x_3	0.1300	x_6,x_7,x_8	0.0657	x_3,x_6,x_8,x_9	0.0380
7	x_8	0.4170	x_4,x_6	0.1319	x_1,x_2,x_6	0.0693	x_2,x_4,x_6,x_9	0.0380
8	x_1	0.4258	x_6,x_7	0.1348	x_2,x_3,x_7	0.0694	x_2,x_4,x_6,x_7	0.0384
9			x_2,x_7	0.1439	x_3,x_6,x_7	0.0703	x_1,x_3,x_4,x_6	0.0385
10			x_3,x_4	0.1591	x_1,x_3,x_6	0.0714	x_1,x_3,x_6,x_8	0.0386

Table IV.16. Sigmoid Predictor Ranking by MSE for Gene $CycA(x_5)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_2,x_3,x_6,x_7,x_8

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_9	0.0011	x_1,x_9	0.0983	x_2,x_4,x_9	0.0013	x_1,x_3,x_5,x_9	0.0008
2	x_5	0.3335	x_4,x_9	0.0988	x_1,x_4,x_9	0.0015	x_2,x_7,x_8,x_9	0.0009
3	x_8	0.3757	x_7,x_9	0.1288	x_4,x_7,x_9	0.0015	x_2,x_4,x_8,x_9	0.0009
4	x_3	0.4063	x_2,x_9	0.1291	x_7,x_8,x_9	0.0019	x_3,x_4,x_5,x_9	0.0009
5	x_7	0.4199	x_8,x_9	0.1308	x_1,x_7,x_9	0.0286	x_2,x_4,x_7,x_9	0.0010
6	x_2	0.4220	x_5,x_9	0.1574	x_1,x_8,x_9	0.0288	x_1,x_3,x_7,x_9	0.0010
7	x_4	0.4262	x_3,x_9	0.1598	x_2,x_7,x_9	0.0288	x_1,x_3,x_4,x_9	0.0010
8	x_1	0.4345	x_2,x_7	0.2270	x_3,x_7,x_9	0.0313	x_1,x_3,x_8,x_9	0.0011
9			x_7,x_8	0.2467	x_2,x_8,x_9	0.0319	x_3,x_4,x_7,x_9	0.0011
10			x_3,x_7	0.2692	x_5,x_8,x_9	0.0321	x_2,x_5,x_8,x_9	0.0011

Table IV.17. Sigmoid Predictor Ranking by MSE for Gene $Cdc20(x_6)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_6	0.1722	x_6,x_9	0.0370	x_5,x_6,x_9	0.0069	x_1,x_3,x_6,x_9	0.0062
2	x_9	0.1823	x_5,x_9	0.1009	x_1,x_6,x_9	0.0346	x_1,x_4,x_6,x_9	0.0065
3	x_8	0.2875	x_2,x_6	0.1198	x_2,x_6,x_9	0.0557	x_5,x_6,x_8,x_9	0.0066
4	x_3	0.3265	x_8,x_9	0.1282	x_6,x_8,x_9	0.0611	x_3,x_5,x_6,x_9	0.0066
5	x_1	0.3492	x_1,x_6	0.1310	x_4,x_6,x_9	0.0677	x_1,x_5,x_6,x_9	0.0067
6	x_2	0.3922	x_5,x_6	0.1322	x_3,x_6,x_9	0.0769	x_2,x_5,x_6,x_9	0.0067
7	x_5	0.3988	x_1,x_3	0.1352	x_5,x_6,x_8	0.0922	x_4,x_5,x_6,x_9	0.0070
8	x_4	0.4357	x_3,x_6	0.1417	x_4,x_6,x_8	0.0954	x_3,x_4,x_6,x_9	0.0318
9			x_4,x_9	0.1511	x_1,x_5,x_6	0.0966	x_1,x_6,x_8,x_9	0.0339
10			x_1,x_9	0.1569	x_1,x_3,x_6	0.0986	x_4,x_6,x_8,x_9	0.0363

Table IV.18. Sigmoid Predictor Ranking by MSE for Gene $Cdh1(x_7)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_5,x_6,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_7	0.1799	x_1,x_7	0.1581	x_3,x_5,x_7	0.1136	x_3,x_5,x_6,x_7	0.1126
2	x_4	0.3708	x_6,x_7	0.2414	x_1,x_4,x_7	0.1214	x_1,x_3,x_5,x_7	0.1129
3	x_1	0.3714	x_4,x_7	0.2425	x_1,x_5,x_7	0.1415	x_3,x_5,x_7,x_9	0.1133
4	x_5	0.4013	x_7,x_9	0.2481	x_1,x_3,x_7	0.1480	x_2,x_3,x_5,x_7	0.1152
5	x_6	0.4340	x_5,x_7	0.2707	x_1,x_7,x_9	0.1511	x_1,x_4,x_5,x_7	0.1204
6	x_9	0.4369	x_3,x_7	0.2755	x_1,x_6,x_7	0.1512	x_1,x_4,x_7,x_9	0.1209
7	x_2	0.4431	x_3,x_4	0.2755	x_1,x_2,x_7	0.1529	x_1,x_2,x_5,x_7	0.1215
8	x_3	0.4668	x_2,x_7	0.2757	x_4,x_6,x_7	0.1696	x_1,x_3,x_4,x_7	0.1220
9			x_1,x_5	0.2876	x_2,x_6,x_7	0.1740	x_1,x_4,x_6,x_7	0.1225
10			x_3,x_5	0.2927	x_4,x_5,x_7	0.1752	x_1,x_3,x_6,x_7	0.1237

Table IV.19. Sigmoid Predictor Ranking by MSE for Gene $UbcH10(x_8)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_5,x_6,x_7,x_9

Rank	1 input		2 inputs		3 inputs		4 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_6	0.1868	x_6,x_7	0.0073	x_6,x_7,x_8	0.0071	x_3,x_4,x_6,x_7	0.0061
2	x_7	0.1874	x_5,x_7	0.1219	x_5,x_6,x_7	0.0385	x_1,x_3,x_6,x_7	0.0068
3	x_3	0.3262	x_7,x_8	0.1259	x_4,x_6,x_7	0.0617	x_2,x_6,x_7,x_8	0.0069
4	x_8	0.3465	x_3,x_7	0.1288	x_3,x_6,x_7	0.0640	x_5,x_6,x_7,x_8	0.0070
5	x_1	0.4170	x_1,x_7	0.1481	x_2,x_6,x_7	0.0656	x_4,x_6,x_7,x_8	0.0071
6	x_4	0.4173	x_6,x_8	0.1562	x_1,x_6,x_7	0.0671	x_1,x_6,x_7,x_8	0.0071
7	x_2	0.4183	x_3,x_6	0.1607	x_2,x_4,x_7	0.0853	x_3,x_6,x_7,x_8	0.0072
8	x_5	0.4214	x_2,x_7	0.1624	x_2,x_3,x_7	0.0884	x_3,x_5,x_6,x_7	0.0074
9			x_5,x_6	0.1747	x_3,x_7,x_8	0.0911	x_1,x_5,x_6,x_7	0.0074
10			x_4,x_6	0.1851	x_3,x_4,x_6	0.0984	x_4,x_5,x_6,x_7	0.0074

Table IV.20. Sigmoid Predictor Ranking by MSE for Gene $CycB(x_9)$ in Mutated Network (Top 10 Predictors Shown), Correct Predictor is x_6,x_7

For the mutated mammal network, Table IV.5 through Table IV.12 (linear representation) and Table IV.13 through Table IV.20 (sigmoid representation) lists the top 10 predictors for genes x_2 through x_9 respectively as determined by our algorithm. Gene CycD (x_1) not included as this gene is controlled by an extracellular signal and as such is not regulated by any of the other 8 genes in the network. For each target gene, the correct (actual) predictor is listed in the table captions. Each table shows predictors for a specific target gene and is organized as follows. The 1 input column lists the 1 input predictors ranked by their associated MSE from lowest MSE to highest MSE. The top ranked 1 input predictor has the lowest MSE and therefore is the best fitting 1 input predictor. Similarly, the 2 input column lists the 2 input predictors ranked by MSE. And so on for the 3 input column and 4 input columns. For example, Table IV.12 lists the predictors for $CycB(x_9)$. In the 1 input column, the best (lowest MSE) 1 input predictor for *CycB* is x_7 with a MSE of 0.123888. Looking at the 2 input column, the best 2 input predictor is x_6,x_7 with a MSE of 0.024118. For *CycB*, x_6,x_7 happens to be the actual or correct predictor.

In general, we find the correct predictor is identified as a top rank predictor in one of the input columns for majority of genes ($E2F(x_3),CycE(x_4),Cdc20(x_6),Cdh1(x_7)$, and

$CycB(x_9)$) in the mutated mammal network. The exceptions are for gene $Rb(x_2)$ where the correct predictor is the sixth ranked predictor in the list, and for genes $CycA(x_5)$ and $UbcH10(x_8)$ which have more than 4 inputs, and thus not listed in the tables which only show up to 4 input predictors.

For gene $Rb(x_2)$, the distribution of samples do not completely cover the 4-input state space, hence several predictors and Zhelgakin functions can closely fit with low error. However, we observe that while the top rank predictor $\{x_1, x_4, x_8, x_9\}$ is not the correct predictor $\{x_1, x_4, x_5, x_9\}$, the top rank predictor does contains 3 of the 4 correct input genes. We make similar observation with genes $CycA(x_5)$ and $UbcH10(x_8)$, in that the top rank predictors contain many of the correct input genes in the actual predictors. This information can be useful helpful in refining future tests for gene expression measurements.

IV-C.2. Predictor Selection Method

While the algorithm produces a ranked list of predictors for a gene, it may be desirable to select a single best predictor. As observed from the predictor tables for the mutated mammal network, the correct predictor is generally the top ranked predictor from either the 1, 2, 3, or 4-input predictor lists. To select which i-input predictor list to choose from, we use a metric called the *resolution ratio* R_i, which measures the difference between the top ranked predictor and second ranked predictor of a gene with i-inputs. The resolution ratio is defined as the ratio between the second and top ranked gene as shown in Equation 4.3.

$$R_i = MSE_{i,second}/MSE_{i,top} \tag{4.3}$$

A high resolution ratio R_i indicates the top rank predictor has significantly lower error than all other predictors of the same input size, and thus likely to be the correct predictor. While a low resolution ratio indicates that several predictors (including the top rank predictor) have similarly low error due to underfitting of the data (missing some of the input genes), overfitting of the data (including additional or wrong input genes), or inadequate sample distribution.

For example, let us assume for gene x_i its predictor is x_j, x_k, or in other words the target gene x_i is regulated by two input genes x_j and x_k. Given adequate expression samples, we expect the MSE of the 2-input predictor x_j, x_k will be low since this is the actual predictor, while any other 2-input predictors for x_i will have a high MSE. As such, the resolution ratio for this 2-input predictor R_2 will be expected to be high. Now let us consider the underfit situation. For the target gene x_i, we expect either the 1-input predictor x_j or the 1-input predictor x_k will have low MSE as both predictors contain input genes from the actual predictor x_j, x_k. However, as the MSE of these two predictors will be similar, the resolution ratio for the 1-input predictor R_1 will be low. Next, we consider the overfit situaton. For the target gene x_i, we expect any 3-input or larger predictor that contains x_j, x_k as a subset will have low MSE since the that subset is the actual predicotr, while any additional input genes add only noise. As a result, several predictors will have similarly low MSE and the resolution ratio R_3 will again be low.

Our selection method determines the resolution ratio of all top rank predictors for each input size, and then selects the top rank predictor with the highest resolution ratio.

	Gene	$p_{1,top}$	R_1	$p_{2,top}$	R_2	$p_{3,top}$	R_3	$p_{4,top}$	R_4
x_2	*Rb*	x_1	1.394	x_1, x_4	1.407	x_1, x_4, x_9	1.021	x_1, x_4, x_8, x_9	1.034
x_3	*E2F*	x_7	1.047	x_2, x_9	1.009	x_2, x_5, x_9	1.262	x_2, x_5, x_8, x_9	1.156
x_4	*CycE*	x_3	1.177	x_2, x_3	3.827	x_2, x_3, x_5	1.222	x_1, x_2, x_3, x_7	1.227
x_5	*CycA*	x_6	1.006	x_2, x_6	1.228	x_2, x_6, x_7	1.009	x_2, x_6, x_8, x_9	1.019
x_6	*Cdc*20	x_9	17.494	x_1, x_9	1.255	x_1, x_4, x_9	1.031	x_3, x_4, x_7, x_9	1.029
x_7	*Cdh*1	x_6	1.064	x_6, x_9	1.707	x_5, x_6, x_9	1.384	x_5, x_6, x_8, x_9	1.408
x_8	*UbcH*10	x_7	1.959	x_1, x_7	1.037	x_1, x_4, x_7	1.037	x_1, x_3, x_6, x_7	1.036
x_9	*CycB*	x_7	1.212	x_6, x_7	3.373	x_6, x_7, x_8	1.982	x_3, x_6, x_7, x_8	1.035

Table IV.21. Resolution Ratio R_i for Top Rank Predictors from Mutated Network (Linear Representation)

	Gene	$p_{1,top}$	R_1	$p_{2,top}$	R_2	$p_{3,top}$	R_3	$p_{4,top}$	R_4
x_2	Rb	x_1	1.034	x_1,x_4	1.185	x_1,x_4,x_9	1.597	x_1,x_4,x_7,x_9	1.005
x_3	$E2F$	x_2	1.013	x_2,x_5	2.046	x_2,x_5,x_9	9.160	x_2,x_5,x_7,x_9	1.032
x_4	$CycE$	x_3	1.547	x_2,x_3	81.104	x_2,x_3,x_6	1.302	x_1,x_2,x_3,x_7	1.056
x_5	$CycA$	x_3	1.010	x_2,x_6	1.388	x_2,x_6,x_8	1.032	x_2,x_3,x_6,x_8	3.659
x_6	$Cdc20$	x_9	298.041	x_1,x_9	1.005	x_2,x_4,x_9	1.112	x_1,x_3,x_5,x_9	1.080
x_7	$Cdh1$	x_6	1.058	x_6,x_9	2.720	x_5,x_6,x_9	5.007	x_1,x_3,x_6,x_9	1.052
x_8	$UbcH10$	x_7	2.061	x_1,x_7	1.526	x_3,x_5,x_7	1.058	x_3,x_5,x_6,x_7	1.002
x_9	$CycB$	x_6	1.003	x_6,x_7	16.578	x_6,x_7,x_8	5.424	x_3,x_4,x_6,x_7	1.114

Table IV.22. Resolution Ratio R_i for Top Rank Predictors from Mutated Network (Sigmoid Representation)

	Gene	Correct Predictor	Selected Predictor (linear)	Selected Predictor (sigmoid)
x_2	Rb	x_1,x_4,x_5,x_9	x_1,x_4	x_1,x_4,x_9
x_3	$E2F$	x_2,x_5,x_9	x_2,x_5,x_9	x_2,x_5,x_9
x_4	$CycE$	x_2,x_3	x_2,x_3	x_2,x_3
x_5	$CycA$	x_2,x_3,x_6,x_7,x_8	x_2,x_6	x_2,x_3,x_6,x_8
x_6	$Cdc20$	x_9	x_9	x_9
x_7	$Cdh1$	x_5,x_6,x_9	x_6,x_9	x_5,x_6,x_9
x_8	$UbcH10$	x_5,x_6,x_7,x_9	x_7	x_7
x_9	$CycB$	x_6,x_7	x_6,x_7	x_6,x_7
correct			4	5

Table IV.23. Comparison of Selected Predictors Using Highest R_i for Mutated Network

Table IV.21 (linear) and Table IV.22 (sigmoid) lists all the resolution ratios and top rank predictors for the mutated mammal cell cycle network. The selected predictors for each gene as chosen by our method is shown in Table IV.23. In general, we find the majority of selected predictors are the correct predictors for genes with adequate expression sampling. Also, we find higher number of correctly select predictors using the sigmoid representation for gene expression values.

IV-C.3. Melanoma Network

Based on the results from the mutated mammal cell-cycle network using synthetic data, we evaluate our predictor ranking and selection algorithms to the actual data from a melanoma network study [32]. This study identified seven genes *PIRIN*, *S*100*P*, *RET*1, *MART*1, *HADHB*, *STC*2, and *WNT*5*A*, to be closely related with the metastasis of melanoma. From [32], 31 gene expression lines (states) were measured, and then reduced to seven distinct lines (shown in Table IV.24).

PIRIN x_1	*S*100*P* x_2	*RET*1 x_3	*MART*1 x_4	*HADHB* x_5	*STC*2 x_6	*WNT*5*A* x_7
0.002	0.020	0.275	0.010	0.227	1.000	1.000
0.056	0.000	0.239	0.011	0.430	0.583	0.318
0.387	0.006	0.440	0.008	0.070	0.055	0.511
0.137	0.147	0.156	0.005	0.227	0.014	0.026
0.222	0.168	0.532	0.141	0.395	0.000	0.000
0.751	0.016	0.349	0.152	0.564	0.197	0.005
0.401	0.411	0.028	0.778	0.663	0.016	0.018

Table IV.24. Normalized Gene Expression Lines for Melanoma Network

We apply our ranking algorithm on each gene x_1 to x_7 in the melanoma network and show the results in Table IV.25 through Table IV.31 respectively. From [27] which inferred the predictor set on the binary valued melanoma gene expression data, we observed that one of the attractor cycle ordering is common to the majority of predictor selection results. This attractor cycle ordering is used along with the normalized gene expression data in our algorithm.

Due to the limited sample size of 7 states, we limit the algorithm to search predictors up to 3 input genes. Similar to assumptions made in the mutated mammal network, this is a reasonable constraint as most genes have been observed to be regulated by few input genes.

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_4	0.012746	x_2,x_4	0.012406	x_3,x_4,x_7	0.007550
2	x_5	0.054450	x_4,x_6	0.012765	x_3,x_4,x_5	0.010336
3	x_2	0.082764	x_4,x_7	0.012824	x_2,x_4,x_5	0.012355
4	x_3	0.181548	x_3,x_4	0.012928	x_2,x_3,x_4	0.012401
5	x_7	0.294790	x_4,x_5	0.014873	x_2,x_4,x_6	0.012401
6	x_6	0.357402	x_3,x_5	0.037186	x_2,x_4,x_7	0.012402
7			x_5,x_6	0.040313	x_4,x_5,x_7	0.012740
8			x_5,x_7	0.051799	x_4,x_6,x_7	0.012747
9			x_2,x_5	0.054959	x_3,x_4,x_6	0.012750
10			x_2,x_6	0.082925	x_4,x_5,x_6	0.012755

Table IV.25. Predictor Ranking by MSE for Gene $PIRIN(x_1)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_3	0.066051	x_1,x_3	0.002817	x_1,x_5,x_7	0.001633
2	x_1	0.082764	x_3,x_5	0.005744	x_1,x_3,x_6	0.002146
3	x_4	0.140740	x_1,x_5	0.006163	x_1,x_3,x_7	0.002355
4	x_5	0.153993	x_1,x_6	0.007762	x_1,x_3,x_5	0.002661
5	x_7	0.191924	x_3,x_4	0.009302	x_1,x_3,x_4	0.002848
6	x_6	0.224416	x_4,x_6	0.009450	x_3,x_5,x_7	0.005743
7			x_4,x_7	0.009475	x_3,x_4,x_5	0.005754
8			x_5,x_7	0.009613	x_3,x_5,x_6	0.005834
9			x_3,x_6	0.009832	x_1,x_4,x_5	0.005978
10			x_1,x_7	0.010902	x_1,x_5,x_6	0.006015

Table IV.26. Predictor Ranking by MSE for Gene $S100P(x_2)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_2	0.066051	x_6,x_7	0.059060	x_1,x_6,x_7	0.054525
2	x_4	0.163481	x_1,x_7	0.061057	x_2,x_6,x_7	0.055919
3	x_1	0.181548	x_1,x_2	0.064650	x_1,x_2,x_7	0.056163
4	x_7	0.183881	x_2,x_5	0.065870	x_5,x_6,x_7	0.058549
5	x_5	0.218690	x_1,x_5	0.066056	x_4,x_6,x_7	0.058892
6	x_6	0.260163	x_2,x_6	0.066109	x_1,x_4,x_7	0.059260
7			x_4,x_5	0.066131	x_1,x_5,x_7	0.060139
8			x_2,x_7	0.066222	x_1,x_2,x_5	0.062339
9			x_2,x_4	0.066566	x_2,x_4,x_5	0.063420
10			x_1,x_4	0.067849	x_1,x_4,x_5	0.063754

Table IV.27. Predictor Ranking by MSE for Gene $RET1(x_3)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_1	0.012746	x_1,x_7	0.008141	$x1,x_2,x_7$	0.004954
2	x_5	0.137126	x_1,x_2	0.009512	$x1,x_6,x_7$	0.008154
3	x_2	0.140740	x_1,x_3	0.012364	$x1,x_5,x_7$	0.008190
4	x_3	0.163481	x_1,x_6	0.012821	$x1,x_2,x_3$	0.009267
5	x_7	0.315267	x_1,x_5	0.019267	$x1,x_3,x_7$	0.009269
6	x_6	0.343011	x_2,x_5	0.088936	$x1,x_2,x_6$	0.009520
7			x_3,x_5	0.110609	$x1,x_3,x_5$	0.009832
8			x_5,x_6	0.125847	$x1,x_2,x_5$	0.009886
9			x_3,x_6	0.132900	$x1,x_3,x_6$	0.012235
10			x_5,x_7	0.133036	$x1,x_5,x_6$	0.012721

Table IV.28. Predictor Ranking by MSE for Gene $MART1(x_4)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_1	0.054450	x_1,x_4	0.025557	x_1,x_4,x_7	0.021039
2	x_4	0.137126	x_1,x_2	0.033795	x_1,x_3,x_4	0.022426
3	x_2	0.153993	x_1,x_7	0.050079	x_1,x_2,x_4	0.024602
4	x_3	0.218690	x_1,x_3	0.051083	x_1,x_4,x_6	0.025467
5	x_7	0.315565	x_1,x_6	0.054356	x_1,x_2,x_7	0.029309
6	x_6	0.334355	x_2,x_4	0.124102	x_1,x_2,x_3	0.030218
7			x_4,x_7	0.136953	x_1,x_2,x_6	0.033643
8			x_4,x_6	0.137194	x_1,x_3,x_6	0.048583
9			x_3,x_4	0.137458	x_1,x_3,x_7	0.049295
10			x_2,x_7	0.154147	x_1,x_6,x_7	0.049949

Table IV.29. Predictor Ranking by MSE for Gene $HADHB(x_5)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_7	0.096190	x_3,x_7	0.075564	x_1,x_3,x_7	0.069109
2	x_2	0.224416	x_1,x_7	0.081605	x_3,x_5,x_7	0.075394
3	x_3	0.260163	x_2,x_7	0.096030	x_2,x_3,x_7	0.075572
4	x_5	0.334355	x_4,x_7	0.096044	x_3,x_4,x_7	0.075572
5	x_4	0.343011	x_5,x_7	0.101558	x_1,x_2,x_7	0.081524
6	x_1	0.357402	x_2,x_4	0.216856	x_1,x_4,x_7	0.081537
7			x_2,x_5	0.216989	x_1,x_5,x_7	0.081647
8			x_3,x_5	0.218127	x_2,x_4,x_7	0.095690
9			x_3,x_4	0.218175	x_2,x_5,x_7	0.095914
10			x_2,x_3	0.218195	x_4,x_5,x_7	0.096012

Table IV.30. Predictor Ranking by MSE for Gene $STC2(x_6)$ in Melanoma Network (Top 10 Predictors Shown)

Rank	1 input		2 inputs		3 inputs	
	Predictor	MSE	Predictor	MSE	Predictor	MSE
1	x_6	0.096190	x_5,x_6	0.063681	x_1,x_5,x_6	0.051371
2	x_3	0.183881	x_3,x_6	0.091656	x_3,x_5,x_6	0.060039
3	x_2	0.191924	x_1,x_6	0.095633	x_2,x_5,x_6	0.063405
4	x_1	0.294790	x_4,x_6	0.095824	x_4,x_5,x_6	0.063546
5	x_4	0.315267	x_2,x_6	0.095870	x_1,x_3,x_6	0.084099
6	x_5	0.315565	x_3,x_5	0.162585	x_3,x_4,x_6	0.091255
7			x_1,x_5	0.171238	x_2,x_3,x_6	0.091303
8			x_1,x_3	0.176896	x_1,x_4,x_6	0.095274
9			x_2,x_3	0.182140	x_1,x_2,x_6	0.095318
10			x_3,x_4	0.182581	x_2,x_4,x_6	0.095329

Table IV.31. Predictor Ranking by MSE for Gene $WNT5A(x_7)$ in Melanoma Network (Top 10 Predictors Shown)

From the predictor rankings, we apply our selection method based on MSE and resolution ratio and find the best predictors for each gene. The selected predictors are shown in Table IV.32 and these results can provide direction for further validation in lab experiments. In addition, our algorithm also returns the associated Zhegalkin function along with the selected predictor, also shown in Table IV.32.

	Gene	Selected Predictor	Zhegalkin function
x_1	*PIRIN*	x_3,x_4,x_7	$x_4+x_3x_7-x_4x_7$
x_2	$S100P$	x_1,x_3	x_1x_3
x_3	$RET1$	x_1,x_6,x_7	$x_7-x_1x_7-x_6x_7+2x_1x_6x_7$
x_4	$MART1$	x_1,x_2,x_7	$x_1-x_1x_2-x_1x_7+x_1x_2x_7$
x_5	*HADHB*	x_1,x_4	$x_1+x_4-2x_1x_4$
x_6	$STC2$	x_1,x_3,x_7	$x_7-x_1x_7-x_3x_7+x_1x_3x_7$
x_7	$WNT5A$	x_1,x_5,x_6	$x_1+x_6-x_1x_5-2x_1x_6-x_5x_6+2x_1x_5x_6$

Table IV.32. Predictor Selection for Melanoma Network

For example, let us examine the predictor selected for WNT5A. In [32], the expression of WNT5A was observed to been associated with the metastasis of melanoma, so determining the predictors of WNT5A is of great interest for GRN control and intervention. Our algorithm finds the best predictor (predictor containing Zhelgakin function with least error) for WNT5A to be x_1, x_5, x_6, or in other words $PIRIN$, $HADHB$, and $STC2$, the results of which appear consistent with literature [2, 60]. From our algorithm, the corresponding Zhegalkin function for WNT5A is $x_1 + x_6 - x_1x_5 - 2x_1x_6 - x_5x_6 + 2x_1x_5x_6$, which we can convert to the Boolean function $\overline{HADHB}(PIRIN \oplus STC2)$. These findings can be used by biologists to develop drugs that target $PIRIN$, $HADHB$, and $STC2$ to modify the expression of $WNT5A$ and control the metastasis of melanoma.

IV-D. Chapter Summary

In this chapter, we have presented a method for inferring and ranking predictors from normalized continuous gene expression data using Zhegalkin functions. Our algorithm explores all possible predictor combinations for a target gene and measures the error of each predictor based on its best fitting Boolean logic function (represented as a Zhegalkin function) upon the gene expression data (linear or sigmoid representation). The predictors are then ranked by error to determine a list of top predictors for the target gene, from which a single predictor can be chosen, or can be used to guide future expression measurement experiments. We validate our Zhegalkin predictor inference method on synthetic data from the mutated mammalian network and show how results can be used to rank and select predictors for genes. We also demonstrate our method on actual data from melanoma network.

Additionally, the ranked list can be used to improve predictor set inference (see Chapter 2) by assigning weights to predictors relative to the MSE. The SAT formulation can be modified to a Weighted Partial Max-SAT (WPMS) formulation to select predictors that satisfy GRN constraints as well as minimizing the overall MSE weights.

The work presented in this and preceding chapters have focused on inferring the GRN using logic synthesis tools. An accurate representation of the GRN is necessary to understand how genes are regulated in a system, how regulation can fail leading to disease, and more importantly, how to control the GRN to treat the disease. In the next chapter, we look at applying logic synthesis to the problem of GRN control. In particular, cancer is described in the stuck-at fault model, and weighted partial Max-SAT algorithms based on ATPG techniques are used to determine optimum drug selection for cancer therapy.

CHAPTER V

ATPG FOR CANCER THERAPY[1]

Cancer and other gene related diseases are usually caused by a failure in the signaling pathway between genes and cells. These failures can occur in different areas of the gene regulatory network, but can be abstracted as faults in the regulatory function. For effective cancer treatment, it is imperative to identify faults and select appropriate drugs to treat the faults. In this chapter, we present an extensible Max-SAT based automatic test pattern generation (ATPG) algorithm for cancer therapy [30, 31]. This ATPG algorithm is based on Boolean Satisfiability (SAT) and utilizes the stuck-at fault model for representing signaling faults. A weighted partial Max-SAT formulation is used to enable efficient selection of the most effective drug.

Several usage cases are presented for fault identification and drug selection. These cases include the identification of testable faults, optimal drug selection for single/multiple known faults, and optimal drug selection for overall fault coverage. Experimental results on growth factor (GF) signaling pathways demonstrate that our algorithm is flexible, and can yield an exact solution for each feature in much less than 1 second.

V-A. Background

In all organisms, cell function is supported by the interaction of genes and protein products, forming an interconnected network called the gene regulatory network (GRN) [33]. The interaction or communication between genes and cells is highly complex and multivariate. Cancer and gene-related diseases are often the result of a failure in the signaling, leading

[1] Part of the data reported in this chapter is reprinted with permission from ”Efficient Cancer Therapy using Boolean Networks and Max-SAT-based ATPG” by Pey-Chang Kent Lin, Sunil P. Khatri. *IEEE International Workshop on Genomic Signal Processing and Statistics (GENSIPS) 2011*, Dec. 2011, pp. 97-90, Copyright 2011 by IEEE

to incorrect gene regulation and its associated functions.

The modeling of the gene interactions is thus highly important for understanding the mechanism and therapy of cancer. Because genes are observed to have a switch-like expression (active or inactive), the Boolean network model [12] has become popular for representing the GRN. In the Boolean network, the genes and biochemical pathways are represented as logic functions, much like logic gates in an integrated circuit (IC). This network can be extended to include signaling failures and defects in the GRN, which are represented as faulty lines in the circuit [42].

The issue of faults in circuits is well understood in electronic testing. For example, in chip manufacturing, circuits are typically tested to check that the IC is defect free before shipment to vendors. Manufacturing defects manifest themselves as logical faults modeled as lines (wires) stuck-at '1' or '0'. Using this *stuck-at fault model*, automatic test pattern generation (ATPG) algorithms determine a set of tests (bit vectors on the inputs of the circuit) to test for stuck-at faults in the circuit.

In this chapter, we use the stuck-at fault model for the GRN [42] and employ ATPG techniques to determine a drug vector (set of drugs) to rectify the fault. The ATPG algorithm is developed as a Boolean satisfiability (SAT) based method, where the Boolean network is transformed into a conjunctive normal form (CNF) expression and solved for satisfiability to find the drug vector. In therapy, the goal is to treat the cancer (represented by one or more faults) using drugs with the least negative impact on the patient, ideally by prescribing the fewest number of drugs necessary to avoid unnecessary side-effects and cost. The SAT method is further extended by assigning weights to the circuit outputs and drug vectors, and solved with a weighted partial Max-SAT to find the optimal set of drugs to fix or rectify the fault.

The key contributions of this chapter are:

- In contrast to previous approaches [42] which performs an explicit search, we develop an implicit SAT-based ATPG approach to model and identify detectable faults (single and multiple) in a Boolean network.
- By assigning weights to model output and drug vectors, we use a weighted partial Max-SAT formulation to determine the optimum selection of drugs to rectify a specific fault.
- Our approach can be trivially extended to handle multiple faults.
- We utilize the above techniques for drug therapy to select the minimum set of drugs to provide the best coverage across all single/multiple faults.

V-B. Previous work

In the actual GRN, the gene expression or protein concentration is continuous. However, in our method, the Boolean network (BN) [12] is chosen as preferred network for modeling the GRN. There are several reasons for this choice. First, it has been observed that many genes exhibit a switch-like ON/OFF activity in terms of their expression [19]. Second, a discrete model like the BN is relatively simple and easy to analyze and simulate. And lastly, there are many logic synthesis and test algorithms already developed in circuit design and testing that can be applied to the Boolean network.

In [42], the authors proposed modeling cancer as faults in the signaling network and applied fault analysis for drug intervention to control the GRN. Cancer is a disease that arise from fault(s) in the network leading to loss of cell cycle control and uncontrolled cell proliferation. Therapy involves both identification of the fault and a suitable drug combination to target the fault. To test our method, we focused on the growth factor (GF) signaling pathways, which are often associated with proliferation of cancer. The GRN is modeled using Boolean logic gates and all possible single faults are enumerated. All drug combinations were also simulated to determine the effectiveness of drug combinations towards each fault.

The method proposed in [42] is an ATPG technique in principle. Our approach is similar to [42] in that it uses the BN and models cancer as faults in the network. However, the differences are several. Instead of explicit enumeration of the BN, we use an extensible, implicit SAT-based ATPG approach to efficiently model and identify faults, and perform drug selection. Further, unlike [42], we include weighted clauses for outputs and drugs in the SAT formulation. Using this, the algorithm can implicitly and efficiently determine the drug combination which is maximally effective. Finally, our approach can handle multiple faults easily. The runtimes of our approach are typically much less than a second per set of faults.

In the past, ATPG has been extensively studied in research and industry. One such ATPG technique is the SAT-based ATPG [61, 62, 63] which translates the testing condition into a SAT instance that retains the circuit structure. A test for the fault can then be found by invoking a SAT solver. In the context of cancer therapy, we extend the SAT based approach to handle drugs and multiple faults.

SAT-based approaches have been applied to the analysis of GRNs and Boolean networks. In [27, 31], SAT-based approaches are presented to infer gene predictors and determine gene function from gene expression data using a BN model. Another SAT-based approach for GRN inference is presented in [50]. Assuming an asynchronous logical description of the GRN, [50] expresses GRN constraints into a Boolean formula, from which they infer parameters of the GRN. While in [51], an algorithm is presented to find all attractors in a Boolean network based on a SAT-based bounded model checking. This algorithm uses a SAT-solver to identify paths of a particular length in the state-transition graph of a Boolean network. In these previous works, SAT has been used to infer the GRN. This fundamentally differs from our work which uses SAT to simulate the faulty GRN and control the GRN using drugs.

Control of Boolean networks has been studied from a theoretical standpoint in [64] and using a model checking algorithm in [65]. In these papers, a BN with control nodes is given, and the control strategy denotes a sequence of control signals that deterministically drive the BN from a given initial state, to a desired final state, in *t* time steps. Conceptually, our SAT-based ATPG approach is similar to these methods of Boolean network control, in that we construct a SAT formula to check whether a selection of drugs can drive the system to a desired state. However we differ in a few key areas. First, our approach considers the BN under a stuck-at fault model, in that one or more of the genes can be faulty. This model allows us to apply ATPG techniques to identify faulty genes in the BN which can lead to undesired GRN behavior. And secondly, our approach weighs the drugs and outputs in the ATPG formulation, allowing for different control strategies depending on desired specifications (i.e. selection with fewest drugs or fewest side effects). Unlike [64, 65], our method can also determine the best drug selection on a BN where the faulty gene location is unknown.

V-C. Method

In this section, we present our SAT-based ATPG method. Before the method is described in detail, we first provide definitions for fault modeling and Boolean Satisfiability.

V-C.1. Fault Terminology

Definition V.1: A manifestation of a defect at the abstracted function level is called a **fault**.

In an IC, the difference between a defect and a fault can be explained as imperfections in the hardware and function, respectively. While in genomics, examples of biological defects can include mutations in the gene activation site, malformation of the protein folding, and problems in the gene product transport. Likewise, an example of a biological fault is a modification of the logical function representing a gene, producing the incorrect output.

Definition V.2: A **stuck-at fault** is modeled by assigning a fixed (0 or 1) value to a signal line (input or output of a logic gate) in the circuit.

Definition V.3: An **untestable fault** is a fault which no test can detect. Untestable faults appear in two situations.

- Faults that are *redundant*, whose presence does not change the output behavior of the circuit.
- Faults that change the output behavior of the circuit, but no test (drug vector in the context of cancer therapy) can be generated to propagate or rectify the fault.

V-C.2. Stuck-at Fault Modeling

In the Boolean network model for a GRN, the activity of genes is modeled as a Boolean circuit. We assume the circuit is modeled as an interconnection of Boolean gates. A stuck-at fault is assumed to only affect interconnections (wires or nets) between gates. Each net can have one of two types of faults: stuck-at-1 or stuck-at-0 (s-a-1 and s-a-0, respectively). Thus, a net with a stuck-at-0 fault will always have a logic value 0, irrespective of the correct logic output of the gate (gene) driving the net.

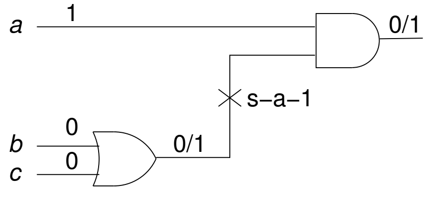


Fig. V.1. Circuit with stuck-at fault

As an example, consider the circuit of Figure V.1 comprising of an OR gate driving an AND gate. Also consider a stuck-at-1 fault at the output of the OR gate, which means that the faulty line remains 1 irrespective of the input state of the OR gate. If the normal (good) output of the OR gate is 1 (in the case where its inputs were $< bc >= 01, 10, 11$), then this fault will not affect any signal in the circuit. However, the input $< bc >= 00$ to the OR gate should produce a 0 output in the good circuit. The good (faulty) value 0 (1) is applied to the AND gate. If the input vector $< abc >= 100$, the good circuit output (true response) and faulty output would differ. Hence $< abc >= 100$ is called a test for the s-a-1 fault on the output of the OR gate.

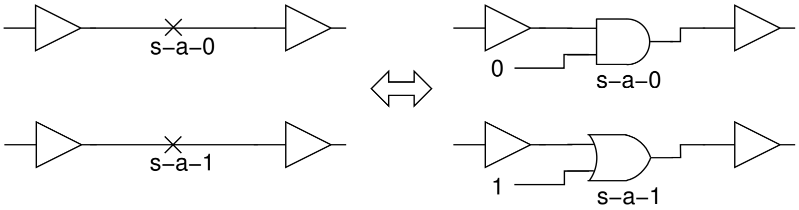


Fig. V.2. Fault modeling and injection

A stuck-at-0 fault is modeled by inserting a two-input AND gate at the fault site as shown in Figure V.2. The side input of the gate is driven by a signal which is set to 1 to simulate a fault-free site, or set to 0 to inject the s-a-0 fault. Similarly, the circuit with a s-a-1 fault is modeled by inserting an OR gate at the site. The side input of this OR gate is set to 0 to simulate a fault-free site, or set to 1 to inject the s-a-1 fault. These gates are inserted at every net (wire), allowing the simulator to inject faults at any site.

Note that drugs are modeled the same as stuck-at faults, wherein a drug that inhibits a gene is modeled as a s-a-0 ”fault”, while a drug that activates a gene is modeled as s-a-1 ”fault”. The gates for drug injection are inserted at the nets of the genes that they target.

V-C.3. SAT-based Formulation for Stuck-at Fault Model

In the SAT based ATPG method, we first generate a formula in CNF to represent tests for the fault. To do so, the circuit from the stuck-at fault model must be converted to a CNF. Every gate (g_i) of the circuit has CNF formula (G_i) associated with it, which represent the function performed by the gate. The formula is true if and only if the variables representing the gate's inputs and outputs take on values consistent with its truth table.

For example, consider a 2-input AND gate (g_j) with the lines x and y as inputs and z as output. The CNF formula (G_j) for the AND gate is written as:

$$G_j = (\overline{z}+x)\cdot(\overline{z}+y)\cdot(z+\overline{x}+\overline{y})$$

A CNF formula for the entire circuit S is obtained by forming the conjunction of the CNF formulas for all the gates of the circuit. If there are n gates in the circuit, then the CNF formula S for the entire circuit is written as:

$$S = \prod_{i=1}^{n} G_i$$

When all the s-a-0 and s-a-1 variables are set to false (0), the CNF formula S describes the good (fault-free) circuit behavior. The faulty circuit is a copy of the fault-free circuit, with faults (s-a-0 or s-a-1 variables) injected at the gates to be affected by faults.

We explain our approach using a simple example. Assume we are given the BN network from Figure V.1, which has two gates g_1 and g_2, primary inputs a, b, c, and primary output z. Also assume and we want to model a stuck-at 1 fault on the output of gate g_1 as shown in the figure. From our stuck at model, we insert an OR gate g_3 at that location. We label the output of g_3 as e, which is now an input to gate g_2. The gate g_3 has two inputs, d (the output of gate g_1) and a side input f. With all inputs and outputs labeled, we obtain the CNF formula for each gates and the entire circuit.

$$G_1 = (\overline{d} + b + c) \cdot (d + \overline{b}) \cdot (d + \overline{c})$$

$$G_2 = (\overline{z} + a) \cdot (\overline{z} + e) \cdot (z + \overline{a} + \overline{e})$$

$$G_3 = (\overline{e} + d + f) \cdot (e + \overline{d}) \cdot (e + \overline{f})$$

$$S = G_1 \cdot G_2 \cdot G_3$$

The value of f, the side input to gate g_3, determines whether the stuck-at 1 fault is activated or now. To activate the fault, f is set true by adding a clause (f) to the CNF, thus $S = G_1 \cdot G_2 \cdot G_3 \cdot (f)$. Likewise, to deactivate the fault, f is set false by adding the clause $(\overline{f})$ to the CNF, thus $S = G_1 \cdot G_2 \cdot G_3 \cdot (\overline{f})$. With our CNF formula for the circuit, we now describe several usage cases employing this CNF in SAT.

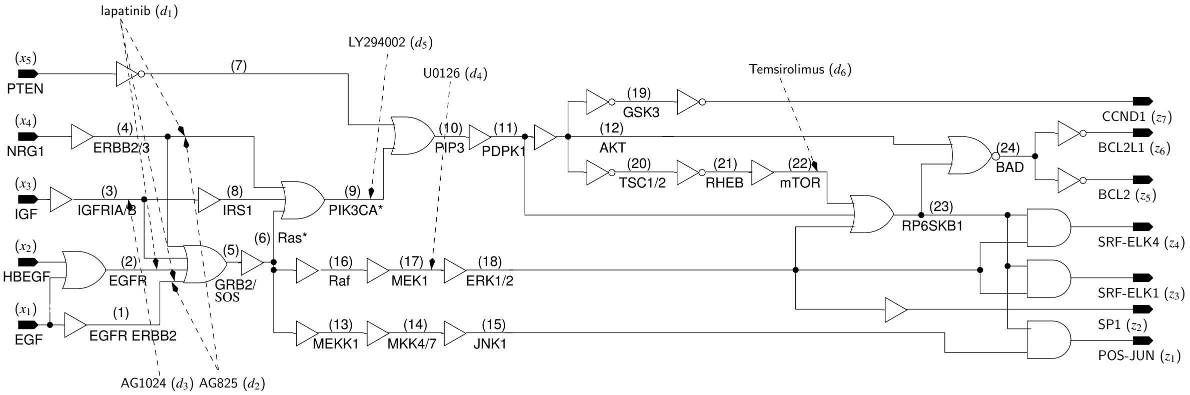


Fig. V.3. Logic circuit stuck-at fault model for GF signaling pathways

V-C.4. Implementation of Fault and Drug Simulation

V-C.4.a. Case 1: Single Stuck-at Fault Identification

In this method, we find all single stuck-at faults which are non-redundant, as well as the faulty outputs that they generate. To proceed with this method, we first simulate the original circuit to determine the correct fault-free output. The circuit is simulated using our SAT formulation in the fault-free and drug-free model for a specified primary input value, and the resulting primary output value for the true response is saved as Z^0.

The next step is to find all faults which are non-redundant. To avoid having to do an exhaustive search on all single stuck-at faults, we perform an All-SAT on the circuit S where we constrain the output to be not Z^0. Assuming n output signals, this constraint is formed as the clause C^1 shown in Equation 5.1.

$$C^1 = (\overline{Z^0_0} + \overline{Z^0_1} + \cdots \overline{Z^0_n}) \tag{5.1}$$

Here Z^0_i is the variable corresponding to the i^{th} output bit.

Furthermore, we also add a constraint to S that the circuit contains only one fault that is injected at a time. This second constraint C^2 (Equation 5.2) is formed by writing clauses of all pairwise combinations of faults, where k is the number of stuck-at faults and f_i is the i^{th} fault.

$$C^2 = (\overline{f_1} + \overline{f_2}) \cdot (\overline{f_1} + \overline{f_3}) \cdots (\overline{f_{k-1}} + \overline{f_k}) \tag{5.2}$$

We now form a new CNF $S^1 = S \cdot C^1 \cdot C^2$ which is a conjunction of (Equation 5.1 and 5.2). The resulting All-SAT on S^1 is a list of all non-redundant single stuck-at faults and their faulty output. These faults are flagged for drug simulation using any of the next three cases.

The results from this case can also be used immediately in several ways. For example, this method classifies for each single stuck-at fault whether it is redundant or non-redundant. That is, any fault which is redundant does not produce an incorrect output, and can be ignored from a therapy standpoint. In a second example, the faulty output from the stuck-at model can be compared to a previously measured output from expression data, in order to identify which genes are potentially faulty. This information can be used to target genes for potential drug development, avoiding genes that are untestable.

V-C.4.b. Case 2: Fault Rectification with Fewest Drugs

In the presence of a particular fault, the problem is determining whether a selection of drugs can rectify the circuit, i.e. change the faulty output to the correct output. If this is not possible, we want to obtain the ”best” or ”closest” output to the correct output, by using drugs. To do this, we guide the WPMS solver by assigning weights to the output states. For example, in the GF network used in our experiments, the fault-free output Z^0 is assigned the highest weight (80) and remaining output states are assigned decreasing weights (70, 60, 50, etc.) based on increasing Hamming distance (1, 2, 3, etc.) from the fault-free output. We assume that faulty states that have a larger Hamming-distance have a more pronounced cancer proliferative effect.

Additionally, the selection of drugs to achieve the best output should use the least number of drugs to minimize the side-effects on the patient. To incorporate this in the WPMS solver, each drug that is *not* selected is given a weight of 1. The GF network example has 6 drugs, thus if no drugs are selected, then the cumulative drug weight is 6. Likewise, if all drugs are selected, the drug weight is 0.

Note that the output and drug weights are assigned in such a way as to avoid the situation where a less-desirable output (with few drugs) is chosen over a higher weight output with more drugs. We assume that from a clinical standpoint, the priority is to first produce the best possible output, and secondarily to use the fewest drugs required for that output.

All faulty circuits with non-redundant faults from Case 1 are augmented with the output and drug weights and simulated using WPMS. The WPMS solver will implicitly and deterministically find the assignment of drugs that achieves the best possible output and with the fewest drugs. The output values, selected drugs, and highest weight of the fault+drug circuits are recorded and compared with the drug-free circuits. An immediate result from this method is that a fault where the fault+drug circuit which obtains its best output with zero drugs is in fact an *untestable fault*, wherein no drug combination can improve the output.

In general, several stuck-at faults can be simultaneously present in the circuit. A circuit with n lines can have $3^n - 1$ possible stuck line combinations. This is because each line can be in one of the three states: s-a-1, s-a-0, or fault-free. All combinations (except one which has all lines in their fault-free state) are counted as faulty. In our implementation, multiple stuck-at faults can easily be modeled for rectification, by setting one or more lines to their faulty state.

V-C.4.c. Case 3: Fault Rectification with Minimal Drug Cost

In the previous case, all drugs are equal in terms of their weight. However, there may be a situation where we would want to differentiate the drugs based on some cost function based on characteristics such as price, number of side-effects, or ease of availability. For example, two drugs with few side-effects may be more desirable than one drug with many side-effects, if both drug selections produce the same output. As such, in the presence of a particular faulty circuit and desired output, the problem is determining a selection of drugs with lowest total cost.

Each drug that is not selected is given a weight proportional to its cost. In our example, we use the number of side-effects as the drug's cost. All faulty circuits with detectable faults from Case 2 are modified with the new drug weights. In addition, the output of the circuit is fixed to the best output as determined in Case 2. These circuits are then solved using WPMS to obtain the selected drugs with lowest cost.

V-C.4.d. Case 4: Determining Therapy with Fewest Drugs and Best Coverage

From Case 2, we identify the drug selection that best rectifies a *certain* fault. However, in drug therapy, the fault location may be unknown. In this situation, a drug selection that rectifies all faults (or as many faults as possible) with the fewest drugs, is desirable.

For each faulty circuit (with a single fault), we find all combinations of 1, 2, and 3 drugs that yield the best output from Case 2. This is done by performing a WPMS All-SAT to find *all* satisfying drug selections with drug weight greater than or equal to $d-3$, where d is the total number of drugs. Each drug selection (or vector) is analyzed to see how many testable faults are rectified or covered by it. The drug vector with the highest coverage and fewest drugs is recorded as a best candidate for therapy.

V-D. Results

V-D.1. Model Implementation

We evaluate the WPMS-based ATPG methods on the GRN that models growth factor (GF) pathways [42]. In multicellular organisms, cell growth and replication is tightly controlled by the cell cycle control. This system receives signals from other cells which are used to decide whether the cell should grow. A failure in these signals can lead to unwanted or unregulated cell growth, leading to cancer. These signaling pathways are well studied, and several drugs have been developed to target different pathways for cancer therapy.

We begin with a BN model of the GF pathways as derived in [42]. In this model, pathways are converted to an equivalent BN logic gate. Each interconnection (net) between logic gates is then assigned a numerical label.

As stated in our approach section,

defects in the GRN are represented as stuck-at faults that permanently set a signal net to 1 or 0. At each net, the logic gates for injecting a s-a-0 or s-a-1 are inserted. If there is a drug that targets the net, the appropriate logic gates are also inserted. The conversion of the faults and drug locations to a logic netlist is shown in Figure V.3. The final circuit is then converted to CNF for further analysis.

In the results, stuck-at faults are referred by the net numbers that are affected (i.e. net 7 s-a-0, means that the signal corresponding to net 7 is stuck-at 0). The network has 5 primary input (PI) signals and 7 primary output (PO) signals. The PIs will be defined as a 5-bit binary vector:

$$X = [EGF, HBEGF, IGF, NRG1, PTEN]$$

The POs will be defined as a 7-bit binary vector:

$$Z = [FOS-JUN, SP1, SRF-ELK1, SRF-ELK4, BCL2, BCL2L1, CCND1]$$

In all tests, the PIs are fixed to $X = 00001$ as this input leads to the non-proliferative output in the fault-free case.

For this network, six drugs are available, defined as a 6-bit vector. Each bit corresponds to a drug, such that a value of 1 on the i^{th} bit indicates that drug i is selected, and a value of 0 indicates that drug i is not selected. The drug vector is:

$$D = [lapatinib, AG825, AG1024, U0126, LY249002, Temsirolimus]$$

All the methods (Case 1 through 4) were implemented using an open-source weighted partial Max-SAT solver called Maxsatz [66, 67]. Our procedure consists of scripts which take the initial CNF, selects desired fault variables, sets output and drug weights, and solves the CNF using Maxsatz. The satisfying assignments are then parsed for the output and drug vectors, and reported in the results. In all examples listed in this section, the WPMS runtime was significantly less than 1 second per CNF.

V-D.2. Simulation Results

V-D.2.a. Case 1: Single Stuck-at Fault Identification

In the single stuck-at fault model, each net was simulated for s-a-0 and s-a-1 with no drugs, and results compared with the fault-free circuit. For fault-free circuit with $X = 00001$, the output vector is $Z^0 = 0000000$. All single nonredundant stuck-at faults, which have an output different from the fault-free circuit, are recorded and shown in Table V.1. In this table, the first three columns show the affected net, the stuck-at value, and the faulty output, respectively.

From this table, we observe that nets 13, 14, and 15 are not listed. The presence of a fault (s-a-0 or s-a-1) on these nets does not generate an incorrect PO, and as such, these are redundant faults. From a therapy standpoint, the genes corresponding to these faults can be ignored.

V-D.2.b. Case 2: Fault Rectification with Fewest drugs

From the results in Case 1, all non-redundant faults are simulated with drugs. The outputs are first weighted where the fault-free output $Z^0 = 0000000$ has a maximum weight of 80 as it represents a non-proliferative output. All remaining output vectors are given weights of $80 - 10h$, where h is their Hamming distance from the fault-free output. The drugs are also given weights where the non-selection of a drug has a weight of 1. With six drugs, the maximum score is therefore $80 + 6 = 86$.

Table V.1 shows for each non-redundant stuck-at fault, the best output (Column 4), the drug vector to achieve such output (Column 5), and the weight score (Column 6). We observe that for many faults, there exists a drug vector that can completely rectify the fault, and produce a fault-free circuit. Additionally, the corresponding reported drug vector is minimal in the number of drugs used, which is desirable in therapy usage. We also determine that faults on nets 7, 10-15, 18, 19, 23, and 24 are untestable, as no combination of drugs can produce a change in the output. This can be explained as there are no drugs on the fan-out (downstream) of these genes to rectify the fault.

To demonstrate the adaptability of our algorithm, we test it on a few examples of multiple stuck-at faults. Table V.2 shows for a circuit with multiple stuck-at faults, the best drug selection for fault rectification (when possible). The columns of Table V.2 have the same meaning as in Table V.1.

Net	s-a	Faulty PO	Best PO	Drug Vector	Score
1	1	1111111	0000000	010000	85
2	1	1111111	0000000	100000	85
3	1	1111111	0000000	001000	85
4	1	1111111	0000000	010000	85
5	1	1111111	0000000	000110	84
6	1	0000111	0000000	000110	84
7	1	0000111	0000111	000000	56
8	1	1111111	0000000	000010	85
9	1	0000111	0000000	000010	85
10	1	0000111	0000111	000000	56
11	1	0000111	0000111	000000	56
12	1	0000111	0000111	000000	56
16	1	0111110	0000000	000100	85
17	1	0111110	0000000	000100	85
18	1	0111110	0111110	000000	36
19	0	0000001	0000001	000000	76
20	0	0000110	0000000	000001	85
21	1	0000110	0000000	000001	85
22	1	0000110	0000000	000001	85
23	1	0000110	0000110	000000	66
24	0	0000110	0000110	000000	66

Table V.1. Drug Selection for Single Stuck-at Faults

Net	s-a	Faulty PO	Best PO	Drug Vector	Score
1,21	1,1	1111111	0000000	010001	84
4,9	1,1	1111111	0000000	000001	85
5,19	1,0	1111111	0000001	000110	74
6,8	1,1	1111111	0000000	000110	84
7,20	1,1	0000111	0000111	000000	56
8,21	1,0	0000111	0000000	000010	85
13,16	1,1	1111110	0000000	000100	85
1,3,6	1,0,1	1111111	0000000	000110	84
2,14,20	1,1,0	1111111	0000000	100001	84
4,7,17	1,1,1	1111111	0000111	010100	54
4,12,23	1,1,1	1111111	0000111	010000	55
8,9,11	1,1,1	0000111	0000111	000000	56
8,9,21	1,1,0	0000111	0000000	000010	85
12,18,20	0,0,0	0000110	0000000	000001	85
15,17,21	0,0,1	0000110	0000000	000001	85

Table V.2. Drug Selection for Multiple Stuck-at Faults

V-D.2.c. Case 3: Fault Rectification with Minimal Drug Cost

When selecting drugs, there may be multiple drug combinations that may rectify a fault, but where each drug has a different associated cost. We first assign weights to drugs, according to their cost. For this case, we use the number of side-effects as the drug's cost. Drugs AG825, lapatinib, Temsirolimus are assigned weights of 10, 15, and 35, respectively, which correspond to their approximate number of side-effects [68, 69]. However, drugs AG1024, U0126, and LY294002 have yet to under go clinical trial and the number of side-effects is unknown. As such, these drugs are assigned a weight 20, which is an average of the 3 previous weights.

In this GF example, Case 3 simulation provides the same results as in Case 2. This is due to a lack of drugs that share paths in the circuit. In fact, for almost every non-redundant fault, the best output state can only be achieved through one drug vector.

V-D.2.d. Case 4: Determining Therapy with Fewest Drugs and Best Coverage

Using the results from Case 2, we observe that the GF network has 13 testable faults. For these 13 faults, we perform an All-SAT to find the top three scoring drug combinations yielding the best output. All drug combinations are analyzed across all single faults and presented in Table V.3 showing drug vector, count of faults rectified, and fault coverage. Drug vectors are ordered in increasing number of drugs selected.

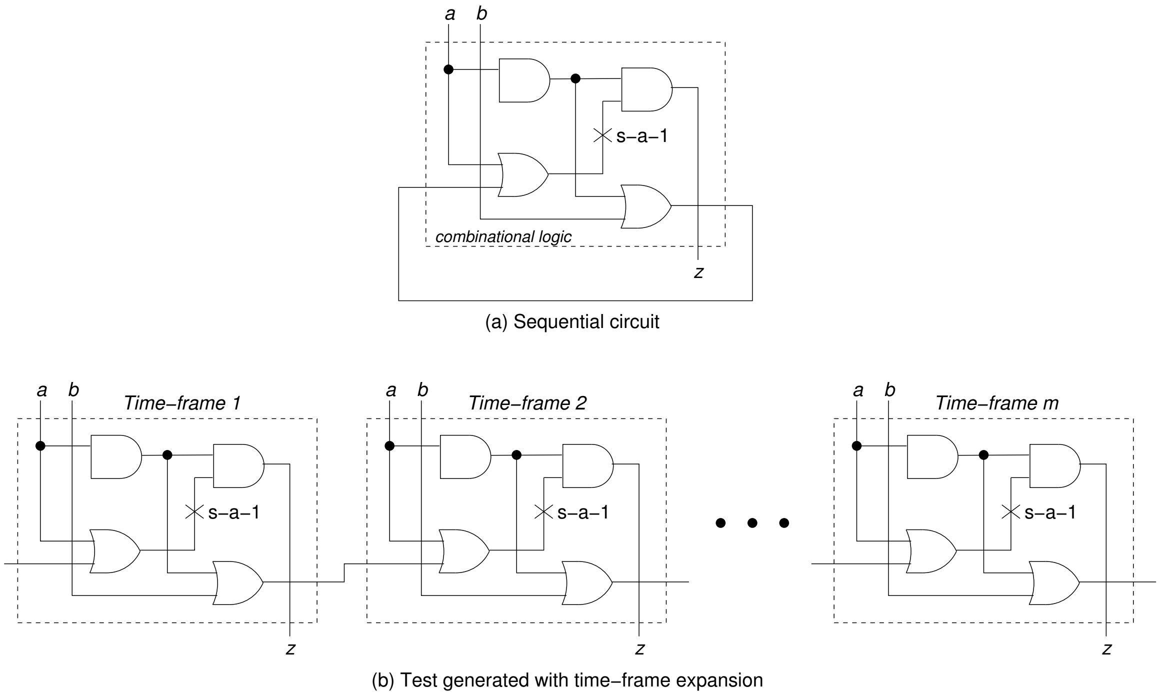
From these results, we observe that with only 1 drug selected, the best coverage is only 23% of faults using lapatinib (d_1) or Temsirolimus (d_6). When allowing for 2 drugs, coverage increases to 77% using the drug combination of U0126 (d_4) and LY294002 (d_5). Finally, we achieve 100% coverage of all testable faults when using the 3 drug combination of U0126 (d_4), LY294002 (d_5), and Temsirolimus (d_6). When the single stuck-at fault location is unknown, these selected drug combinations will be the most effective for therapy and for preventing the proliferation of cancer.

Drug Vector	Count	Coverage	Drug Vector	Count	Coverage
000001	**3**	**23%**	**000111**	**13**	**100%**
000010	2	15%	001011	6	46%
000100	2	15%	001101	6	46%
001000	1	8%	001110	10	77%
010000	2	15%	010011	7	54%
100000	**3**	**23%**	010101	7	54%
000011	5	38%	010110	10	77%
000101	3	23%	011001	6	46%
000110	**10**	**77%**	011010	5	38%
001001	4	31%	011100	5	38%
001010	3	23%	100011	8	62%
001100	3	23%	100101	8	62%
010001	5	38%	100110	10	77%
010010	4	31%	101001	7	54%
010100	4	31%	101010	6	46%
011000	3	23%	101100	6	46%
100001	6	46%	110001	6	46%
100010	5	38%	110010	5	38%
100100	5	38%	110100	5	38%
101000	4	31%	111000	4	31%
110000	3	23%			

Table V.3. Drug Selection Count and Fault Coverage

V-E. Sequential and Feedback Circuits

In this section, we discuss the generalization of our approach to sequential circuits. Thus far, the SAT-based ATPG algorithm has been described for and performed on purely combinational circuits, wherein the primary output of the circuit is dependent only on the primary inputs. We observe that the output of the GF signaling pathway from the experiment is fixed based on the primary inputs, where the drug vector is technically also an input. In general though, the circuit representation of the BN can be sequential, where the primary output is determined by current state in addition to the input. The local GRN for mammalian cell-cycle [59] is one such example of a sequential circuit where gene expression updates based on the current gene state. If we consider a directed graph where the genes are nodes and edges are regulations upon other genes, then a combinational circuit (such as the GF signaling pathway) is acyclic. However, for a directed graph of a sequential circuit, a subset of genes will be inter-regulated forming directed cycles. As such, in the BN, a gene takes its current input (state of its regulatory genes and/or external inputs) and outputs a new state or value for the next time point. We assume in the BN that all genes update synchronously. In other words, for each primary input and current state, the resulting primary output and next state are determined for all genes, and that the next state becomes the new current state. While a synchronous update is biologically unrealistic, it allows us to have deterministic state transitions and simplifies the analysis for our ATPG algorithm.



(a) Sequential circuit

(b) Test generated with time–frame expansion

Fig. V.4. Sequential ATPG by time-frame expansion method

There are several methods for performing sequential ATPG, the most common of which is *Time-Frame expansion* [70]. As shown in Figure 4 V.4, the sequential circuit is replicated m times into a combinational circuit, which models m time steps of the sequential circuit behavior. The i^{th} copy is connected to the $(i+1)^{th}$ copy such that the regulating genes from the i^{th} copy are connected to their target genes in the $(i+1)^{th}$ copy. Each copy is called a frame, and additional frames can be added to the circuit for any length m. In this way, the sequential circuit is converted to a combinational circuit. After the conversion of the sequential circuit to a combinational m step expansion, we can apply our SAT-based ATPG algorithm. When we consider the fault-model of the circuit, we must assume the fault is persistent (i.e. the fault exists in all frames). The corresponding ATPG method must target multiple faults, or in other words, the same fault, but in different time frames.

One consideration for the sequential ATPG is the initialization of state in the first time frame. Ideally a known state should be used, such as one obtained from a previous microarray expression measurement. An alternative is to use an attractor state. In the long-term behavior, the dynamics of the BN transition to the attractors (attractor cycles), thus using an attractor state is a reasonable starting state for therapy.

The complexity of applying SAT-based ATPG to sequential circuits depends on the length of time-frame expansion. For a circuit with k variables in its SAT formulation, each frame increases the number of variables by k. The SAT search space is then 2^{km} for an expanded circuit with m frames. The number of frames for expansion can be bounded. If a subsequence of states has the same first and last state, then the sequence can be stopped. For a BN, the number of frames m can be bounded by the sum of the number of steps it takes to reach an attractor cycle and the maximum length of the attractor cycles for all combinations of drugs under consideration. In the worst case, the number of frames required would equal to the number of possible states, which is 2^{n+d} for a BN with n target genes and d drugs.

V-F. Chapter Summary

In this chapter, we have presented an efficient and extensible SAT-based ATPG methodology for cancer therapy. We approach this problem by representing the BN and cancer as a logic circuit stuck-at fault model. This circuit, along with the testing conditions, is converted into a CNF. The CNF is then augmented with output and drug vectors weights and solved using a weighted partial Max-SAT solver for four different usage cases: (1) single stuck-at fault identification, (2) fault rectification with fewest drugs, (3) fault rectification with minimum drug cost, and (4) determining therapy with fewest drugs and best coverage. We demonstrate these methods on the growth factor signaling pathway, and have presented results that are applicable to cancer therapy. While the GF network example in the case study is a combinational network, our algorithm can easily be extended to address sequential networks, like those found in transcriptional GRNs, by simply unrolling the sequential circuit in time and applying the same methods. Furthermore, all nets, inputs, outputs, and drugs can be assigned weights, which can be made variable, allowing the user to fine-tune the network or design therapies for any number of test situations.

CHAPTER VI

SUMMARY AND FUTURE WORK

VI-A. Summary

With more diseases and health related issues being attributed to genetics, it is imperative to improve our knowledge of gene regulation within the biological system. While single-point measurement of gene expression/detection is relatively simple using micro-arrays or gene chips, measuring or determining the dynamic characteristics of genes in lab is time and labor intensive. Understanding the dynamic interaction of genes is essential in the medical field to study and control cancer and other genetic diseases. As a result, in recent times genomics has become a popular field of research within computational and molecular biology, for modeling and analyzing gene networks and regulation. While biological systems have been observed to exhibit circuit-like properties, there has been little existing work that exploits logic synthesis to model such systems.

Systems engineering approaches are gradually becoming more accepted and necessary as a means to tackle gene regulatory networks and genetic diseases. In our research, we show how several techniques from the field of logic synthesis can be used to model, infer, and control the GRN related to cancer. In particular, this thesis present logic synthesis and SAT based approaches to help infer the predictor sets for GRNs, to determine gene regulating function, and to determine the ”best” set of drugs for cancer therapy. The results from these algorithms can be used by clinicians to determine an optimal drug therapy, by drug developers to target drugs for specific genes, and by biologists to design experiments to extract specific gene interactions. Our research have applied our approaches and presented results for gene networks involving melanoma, p53, mammalian, and growth factor pathways.

VI-B. Future Work

Our work in applying logic synthesis to GRNs only touches the surface of research in genomics. But by presenting our interdisciplinary work as part of this PhD effort, we hope to inspire several additional lines of research using logic synthesis to fundamentally improve and expand our understanding of gene regulation and control.

The following discussion introduces several genomics research ideas for exploration using logic synthesis.

1. A key issue in genomics is handling data with error and noise, particularly in measurement of gene expression. One topic is to analyze the GRN behavior which can lead to measurement of incorrect gene expression. Using the ATPG method discussed in Chapter 5, we can quantify the sensitivity of a GRN to N ”faults” in the GRN (where faults represent incorrectly measured data).

2. Another topic of value is to check the logical equivalence of two GRNs (or subsets of the GRN), using functional equivalence techniques [71]. Such a method may be useful to identify subsets of GRN between two organisms with the same cellular or genetic function, or to compare GRNs of patients to determine effective treatment strategies.

3. Model checking [72], a technique to verify the temporal behavior of logical systems, can be utilized to query the temporal properties of a GRN in a very efficient manner. Given the GRN, and given a state that it is currently in, questions such as ”Is there a way to reach state X in k steps” or ”Is there a way in which state Y is visited infinitely often in the future” can be answered automatically by model checking systems. This approach can be useful in cancer therapy to ask questions about a patients prognosis or determine drug effect on the GRN.

4. Also in the context of uncertainty modeling, probabilistic Boolean Networks (PBNs) [13] are used to model the GRN. Suppose there are *k* BNs which match some observed data. Then each edge in the PBN has a probability, which is the average of the corresponding *k* edges in the BNs. This can result in the allowing of behaviors that are not present in any of the *k* BNs. To avert this issue, Non-Deterministic Finite State Machine (NDFSM) [73] models of the GRN can be developed. Many techniques from the field of automata theory can be brought to bear to develop such an NDFSM.

5. Another possible research direction is to perform ATPG on the state transition graph rather than the logic circuit as we have shown in Chapter 5. Such methods are used in sequential ATPG and present an a method for drug selection given a GRN with feedback or sequential properties.

6. As discussed in Chapter 4, gene expression values are initially measured as continuous values. Gene expressions values can then be thresholded to binary value for use in Boolean logic synthesis algorithms. While Chapter 4 explores using continuous expression values with Zhelgakin function, other alternate logic representations such as asynchronous logic and multi-valued logic may be valuable in the context of genomics as well to more accurately represent gene expression values and regulation.

7. The majority of our algorithms utilizes SAT, and as a consequence, the run time of our approaches is dominated by the SAT solver. Although our algorithms utilize efficient SAT solvers, these solvers are optimized for general or circuit SAT instances. A possible research topic is understanding and improving SAT solving for GRN problems. One interesting observation from our SAT implementation in Chapter 2 and 3, is that the predictor or functions of each gene are encoded in one-hot fashion. A gene will have many possible predictors or functions, each represented by a Boolean variable, however only one per gene can be selected in the SAT solution. This one-hot

encoding is a natural fit for accelerating the SAT process on GPU. The implication for this method of accelerating SAT on GPU extends beyond our algorithm, possibly lending to SAT research in one-hot and/or multi-valued variables.

REFERENCES

[1] B. Alberts, D. Bray, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Essential Cell Biology: An Introduction to the Moleuclar Biology of the Cell*, Garland Publishing Inc., 1997.

[2] A. Datta and E.R. Dougherty, *Introduction to Genomic Signal Processing with Control*, CRC Press, 2007.

[3] W. Saenger, *Principles of nucleic acid structure*, Springer-Verlag, 1983.

[4] W.J. Gehring, "The master control gene for morphogenesis and evolution of the eye," *Genes to Cells*, vol. 1, pp. 11–15, January 1996.

[5] A. Ashley-Koch, Q. Yang, and R.R. Olney, "Sickle hemoglobin (hb s) allele and sickle cell disease: A huGe review," *American Journal of Epidemiology*, vol. 151, no. 9, pp. 839–845, 2000.

[6] P. Mitchell and R. Tjian, "Transcriptional regulation in mammalian cells by sequence-specific dna binding proteins," *Science*, vol. 245, pp. 371–378, July 1989.

[7] B.D. Ripley, "The R project in statistical computing," *MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network*, vol. 1, no. 1, pp. 23–25, 2001.

[8] 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*, vol. 10, no. 4, pp. 337–357, 2002.

[9] G. Vahedi, B. Faryabi, J.-F. Chamberland, A. Datta, and E.R. Dougherty, "Intervention in gene regulatory networks via a stationary mean-first-passage-time control

policy," *Biomedical Engineering, IEEE Transactions on*, vol. 55, no. 10, pp. 2319 –2331, oct. 2008.

[10] T. Chen, H.L. He, G.M. Church, et al., "Modeling gene expression with differential equations," in *Pacific Symposium on Biocomputing*, 1999, vol. 4, p. 4.

[11] F.X. Wu, W.J. Zhang, and A.J. Kusalik, "Modeling gene expression from microarray expression data with state-space equations," in *Pacific Symposium on Biocomputing*, 2004, vol. 9, pp. 581–592.

[12] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437 – 467, 1969.

[13] Ilya Shmulevich and Edward R. Dougherty, *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*, SIAM – Society for Industrial and Applied Mathematics, Philadelphia, PA, 2009.

[14] Nicholas Geard and Janet Wiles, "A gene network model for developing cell lineages," *Artif. Life*, vol. 11, no. 3, pp. 249–268, 2005.

[15] Adam Arkin, John Ross, and Harley H. McAdams, "Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells," *Genetics*, vol. 149, pp. 1633–1648, 1998.

[16] R. Shalgi, D. Lieber, M. Oren, and Y. Pilpel, "Global and local architecture of the mammalian microRNA–transcription factor regulatory network," *PLoS Computational Biology*, vol. 3, no. 7, pp. e131, 2007.

[17] O. Voinnet, "Origin, biogenesis, and activity of plant microRNAs," *Cell*, vol. 136, no. 4, pp. 669–687, 2009.

[18] S. Maslov and K. Sneppen, "Specificity and stability in topology of protein networks," *Science Signalling*, vol. 296, no. 5569, pp. 910, 2002.

[19] Francois Jacob and Jacques Monod, "Genetic regulatory mechanisms in the synthesis of proteins," *Journal of Molecular Biology*, vol. 3, no. 3, pp. 318–356, 1961.

[20] Wylie Burke and Bruce M. Psaty, "Personalized Medicine in the Era of Genomics," *JAMA*, vol. 298, no. 14, pp. 1682–1684, 2007.

[21] M. Teutsch et al., "The evaluation of genomic applications in practice and prevention (EGAPP) initiative: methods of the EGAPP working group," *Genetics in Medicine*, vol. 11, no. 1, pp. 3–14, 2009.

[22] Martin Davis, George Logemann, and Donald Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[23] Niklas Een and Niklas Sorensson, "The minisat page," `http://minisat.se/`, Accessed April 6, 2010.

[24] Bart Selman and Henry Kautz, "Gsat-users-guide," `http://www.cs.rochester.edu/u/kautz/papers/`, Accessed September 27, 2010.

[25] M Moskewicz, C Madigan, Y Zhao, L Zhang, and S Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the Design Automation Conference*, July 2001.

[26] M Silva and J Sakallah, "GRASP-a new search algorithm for satisfiability," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, November 1996, pp. 220–7.

[27] Pey-Chang Kent Lin and S.P. Khatri, “Inference of gene predictor set using Boolean satisfiability,” in *Genomic Signal Processing and Statistics (GENSIPS), 2010 IEEE International Workshop on*, Nov. 2010, pp. 1 –4.

[28] Pey-Chang Kent Lin and Sunil P. Khatri, “Application of logic synthesis to the understanding and cure of genetic diseases,” *Proceedings of the 49th Annual Design Automation Conference*, pp. 734–740, 2012.

[29] Pey-Chang Kent Lin and S.P. Khatri, “Determining gene function in Boolean networks using Boolean satisfiability,” in *Genomic Signal Processing and Statistics (GENSIPS), 2012 IEEE International Workshop on*. IEEE, 2012, pp. 1–4.

[30] Pey-Chang Kent Lin and S.P. Khatri, “Efficient cancer therapy using Boolean networks and Max-SAT-based ATPG,” in *Genomic Signal Processing and Statistics (GENSIPS), 2011 IEEE International Workshop on*. IEEE, 2011, pp. 87–90.

[31] Pey-Chang Kent Lin and S. Khatri, “Application of Max-SAT-based ATPG to optimal cancer therapy design,” *BMC Genomics*, vol. 13, no. Suppl 6, pp. S5, 2012.

[32] M. Bittner et al., “Molecular classification of cutaneous malignant melanoma by gene expression profiling,” *Nature*, vol. 406, no. 3, pp. 536–540, 2000.

[33] Nabil Guelzim et al., “Topological and causal structure of the yeast transcriptional regulatory network,” *Nature Genetics*, vol. 31, pp. 60–63, 2002.

[34] Edward R. Dougherty, Seungchan Kim, and Yidong Chen, “Coefficient of determination in nonlinear signal processing,” *Signal Processing*, vol. 80, no. 10, pp. 2219 – 2235, 2000.

[35] Wentao Zhao, Erchin Serpedin, and Edward R. Dougherty, “Inferring connectivity of genetic regulatory networks using information-theoretic criteria,” *IEEE/ACM Trans.*

Comput. Biol. Bioinformatics, vol. 5, no. 2, pp. 262–274, 2008.

[36] Xiaobo Zhou, Xiaodong Wang, and Edward R. Dougherty, "Gene prediction using multinomial probit regression with Bayesian gene selection," *EURASIP Journal on Applied Signal Processing*, pp. 115–124, 2004.

[37] Wentao Zhou, Erchin Serpedin, and Edward R. Dougherty, "Inferring gene regulatory networks from time series data using the minimum description length principle," *Bioinformatics*, vol. 17, pp. 2129–2135, 2006.

[38] Ranadip Pal, Ivan Ivanov, Aniruddha Datta, Michael L. Bittner, and Edward R. Dougherty, "Generating Boolean networks with a prescribed attractor structure," *Bioinformatics*, vol. 21, no. 21, pp. 4021–4025, 2005.

[39] Stuart A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, USA, 1 edition, June 1993.

[40] Niklas. Een and Niklas Sorensson, *An Extensible SAT-solver*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004.

[41] R. Layek, A. Datta, and E.R. Dougherty, "From biological pathways to regulatory networks," *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 5781 –5786, dec. 2010.

[42] Ritwik Layek, Aniruddha Datta, Michael Bittner, and E.R. Dougherty, "Cancer therapy design based on pathway logic," *Bioinformatics*, vol. 27, no. 4, pp. 548–555, 2011.

[43] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *Trans. AIEE. pt. I*, vol. 72, no. 9, pp. 593–599, 1953.

[44] W. Gosti, S.P. Khatri, and A.L. Sangiovanni-Vincentelli, "Addressing the timing closure problem by integrating logic optimization and placement," in *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, Piscataway, NJ, USA, 2001, ICCAD '01, pp. 224–231, IEEE Press.

[45] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *Proceedings of the International Conference on Computer-Aided Design*, Nov. 1996, pp. 254–61.

[46] R. Brayton, "Understanding SPFDs: A new method for specifying flexibility," in *Workshop Notes, International Workshop on Logic Synthesis*, Tahoe City, CA, May 1997.

[47] S. Sinha and R. Brayton, "Implementation and use of SPFDs in optimizing Boolean networks," in *Proceedings of the International Conference on Computer-Aided Design*, Nov 1998, pp. 103–10.

[48] B. Lin and A. R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," in *Proc. of the Intl. Conf. on VLSI*, Aug. 1989, pp. 187–196.

[49] T. Villa and A. L. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 9, no. 9, pp. 905–924, Sept. 1990.

[50] F. Corbin, L. Bordeaux, Y. Hamadi, E. Fanchon, and L. Trilling, "A SAT-based approach to decipher gene regulatory networks," *Integrative Post-Genomics, RIAMS, Lyon*, 2007.

[51] E. Dubrova and M. Teslenko, "A SAT-based algorithm for finding attractors in synchronous Boolean networks," *IEEE/ACM Trans. Computational Biology and Bioinformatics*, vol. 8, no. 5, pp. 1393–1399, Sept. 2011.

[52] R.A. Weinberg, *The Biology of Cancer*, Garland Science, Princeton, 2006.

[53] Eric Batchelor, Alexander Loewer, and Galit Lahav, "The ups and downs of p53: understanding protein dynamics in single cells," *Nature Reviews Cancer*, vol. 9, no. 5, pp. 371–377, 2009.

[54] S. Faisal, G. Lichtenberg, and H. Werner, "Canalizing zhegalkin polynomials as models for gene expression time series data," in *Engineering of Intelligent Systems, 2006 IEEE International Conference on*. IEEE, pp. 1–6.

[55] S. Faisal, G. Lichtenberg, and H. Werner, "An approach using shegalkin polynomials for modelling microarray timeseries data of eucaryotes," in *Proceedings of International Conference on Systems Biology*, 2004, p. 312.

[56] Marshall H Stone, "The theory of representation for boolean algebras," *Transactions of the American Mathematical Society*, vol. 40, no. 1, pp. 37–111, 1936.

[57] Patrik Dhaeseleer, Shoudan Liang, and Roland Somogyi, "Gene expression data analysis and modeling," in *Pacific Symposium on Biocomputing*, 1999, vol. 99.

[58] Alexander Zien, Thomas Aigner, Ralf Zimmer, and Thomas Lengauer, "Centralization: a new method for the normalization of gene expression data," *Bioinformatics*, vol. 17, no. suppl 1, pp. S323–S331, 2001.

[59] B. Faryabi, J.-F. Chamberland, G. Vahedi, A. Datta, and E.R. Dougherty, "Optimal intervention in asynchronous genetic regulatory networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 3, pp. 412–423, June 2008.

[60] Yufei Xiao and Edward R Dougherty, “The impact of function perturbations in boolean networks,” *Bioinformatics*, vol. 23, no. 10, pp. 1265–1273, 2007.

[61] T. Larrabee, “Efficient Generation of Test Patterns Using Boolean Difference,” in *Proc. of the Intl. Test Conf.*, 1989, pp. 795–801.

[62] P. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, “Combinational test generation using satisfiability,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1167–1176, Sept. 1996.

[63] N.S. Saluja, K. Gulati, and S.P. Khatri, “SAT-based ATPG using multilevel compatible don’t-cares,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, pp. 24:1–24:18, April 2008.

[64] T. Akutsu, M. Hayashida, W.K. Ching, and M.K. Ng, “Control of Boolean networks: Hardness results and algorithms for tree structured networks,” *Journal of Theoretical Biology*, vol. 244, no. 4, pp. 670–679, 2007.

[65] C.J. Langmead and S.K. Jha, “Symbolic approaches for finding control strategies in Boolean networks,” *Journal of Bioinformatics and Computational Biology*, pp. 323–338, April 2009.

[66] Chu Min Li, Felip Manya, and Jordi Planes, “Maxsatz,” `http://home.mis.u-picardie.fr/\textasciitildecli/EnglishPage.html/`, Accessed July 10, 2011.

[67] Chu Li, Felip Manya, Nouredine Mohamedou, and Jordi Planes, “Exploiting cycle structures in Max-SAT,” in *Theory and Applications of Satisfiability Testing - SAT 2009*, Oliver Kullmann, Ed., vol. 5584 of *Lecture Notes in Computer Science*, pp. 467–480. Springer Berlin / Heidelberg, 2009.

[68] Santa Cruz Biotechnology Inc, "Santa cruz biotechnology, inc home," http://www.scbt.com/, Accessed August 15, 2011.

[69] National Center for Biotechnology Information, "Pubmed health - national library of medicine," http://www.ncbi.nlm.nih.gov/pubmedhealth/, Accessed August 15, 2011.

[70] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[71] G. Hachtel and R. Jacoby, "Verification algorithms for vlsi synthesis," in *IEEE Transactions on Computer-Aided Design*, May 1988, pp. 616–640.

[72] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, "Symbolic Model Checking: 10^{20} States and Beyond," *Information and Computation*, vol. 98, no. 2, pp. 142–170, 1992.

[73] M. Damiani, "Nondeterministic finite-state machines and sequential don't cares," in *European Design and Test Conference, 1994. EDAC, The European Conference on Design Automation. ETC European Test Conference. EUROASIC, The European Event in ASIC Design, Proceedings.*, feb-3 mar 1994, pp. 192 –198.