

António José Santos Freitas Gonçalves

Discovering Gene Networks



Department of Computer Science
Faculty of Sciences at University of Porto
November 2012

António José Santos Freitas Gonçalves

Discovering Gene Networks

*Dissertation presented to Faculty of Sciences at University of Porto in partial fulfillment
of the requirements for the degree of Master in Computer Science*

Scientific Advisor: Vítor Manuel de Moraes Santos Costa, PhD

Scientific Co-Advisor: Irene May Lin Ong, PhD

Department of Computer Science

Faculty of Sciences at University of Porto

November 2012

to my parents and girlfriend
without whom, none of this would have been possible...

Acknowledgements

First and foremost, I wish to thank my scientific advisor: Vítor Santos Costa. I learned a lot from him. I am grateful for his support, guidance and patience especially over this past year.

A special word of thanks to Irene Ong who also supported and helped a lot with the guidance of this work.

A very special thank and appreciation to my parents, who always helped, believed and encouraged me in every way they could so that I could have a proper education and also succeed in my professional life.

Also, a very special thanks and appreciation to my girlfriend Raquel, who never let me give up and always believed in me, and gave me her unconditional support every day.

Thanks also to Jeffrey Lewis who provided data and consulted.

A thanks to all my Masters teachers at DCC FCUP who inspired me to pursue knowledge discovery and research.

I also wish to thank the Fundação para a Ciência e a Tecnologia. This work was supported with the FCT grant PTDC/EIA-EIA/100897/2008.

A word of appreciation to my friends and colleagues for the support.

Last, but not least, a thanks to my crazy cats Nero and Sasha who were always able to make me laugh when my work was not going very well at that time.

Abstract

Transcriptional regulation plays an important role in every cellular decision. Gaining an understanding of the dynamics that govern how a cell will respond to diverse environmental cues is difficult using intuition alone. In this work, logic-based regulation models based on state-of-the-art work on statistical relational learning are introduced, and evaluated on time-series gene expression data of the Hog1 pathway. Results show that plausible regulatory networks can be learned from time series gene expression data using a probabilistic logical model. Hence, network hypotheses can be generated from existing gene expression data for use by experimental biologists.

Keywords:-Gene Regulation, Network/Pathway Analysis, Statistical Relational Learning

Palavras-Chave: Regulação de Genes, Análise de Redes/Caminhos, Aprendizagem Relacional Estatística

Contents

Abstract	9
List of Tables	13
List of Figures	16
Acronyms	17
1 Introduction	19
1.1 Introduction	21
1.2 Thesis Roadmap	22
2 Biology Background	25
2.1 DNA	27
2.2 RNA	30
2.2.1 Structure	31
2.2.2 Synthesis	32
2.2.3 Types	34
2.3 Gene Regulatory Networks	36
2.4 Expression Data	39
2.4.1 Data Measurement	39

3	Related Work	41
3.1	Differential equations models	43
3.2	Logical models	47
3.3	Other approaches	50
4	Introduction to Logic Programming	53
4.1	Logic	55
4.2	Binary Decision Diagrams	58
4.2.1	Ordered Binary Decision Diagrams	59
4.3	Problog	63
4.3.1	Learning Problog Programs	67
5	Implemented Work	69
5.1	Representing gene networks with Problog	71
5.2	Experimental Methodology	72
5.2.1	Implementation	76
5.3	Results	77
5.3.1	Problog Performance	80
6	Conclusions	85
6.1	Conclusions	87
6.2	Future Work	87
A	Code	89
	References	98

List of Tables

4.1	Propositional logic symbols	56
5.1	Variation in Correlation Between All Genes and Experiments Along Time	77
5.2	Analysis 2: proposed parents with threshold $0.7 \leq P \leq 0.9$	78
5.3	Experiment 3 (VV): proposed parents per gate. Notice that $\ominus+$ represents the positive parent, and $\ominus-$ the negative parent.	80
5.4	Experiment 3 (LV): proposed parents per gate. Notice that $\ominus+$ represents the positive parent, and $\ominus-$ the negative parent.	80
5.5	Calculation times differences between the two ProbLog versions with no BDDs rebuild	81
5.6	Calculation times differences between the two ProbLog versions with BDDs rebuild after 20 iterations	82
5.7	Calculation times differences between the two ProbLog versions with BDDs rebuild after 45 iterations	82

List of Figures

2.1	The flow on information in a cell, a.k.a. Central dogma of biology . . .	27
2.2	DNA Chemical Structure	28
2.3	DNA double helix structure	29
2.4	DNA base pairs	29
2.5	DNA zipper	30
2.6	RNA Chemical Structure	31
2.7	RNA Secondary Structures	32
2.8	Promoter Sequences	33
2.9	RNA Synthesis	33
2.10	Messenger RNA	34
2.11	RNA abundance in cells	34
2.12	Transfer-messenger RNA	35
2.13	Example of an architecture of the inducible gene network	36
2.14	Gene Regulation: The activation or inhibition of a gene transcription by one or several transcription factors.	38
2.15	Processes in a Gene Regulatory Network	38
2.16	A general scheme for a microarray.	40
3.1	Example of a Boolean network	47

4.1	On the left a Decision tree; In the middle, a corresponding truth table; On the right a corresponding BDD	58
4.2	Reduction of a BDD	59
4.3	Both OBDDs are reduced, but they equivalent. Although only the left one is ordered $[x, y, z]$	60
4.4	Execution of the <i>reduce</i> algorithm	61
4.5	Two arguments for a call <i>apply</i> ($+, B_f, B_g$).	62
4.6	The recursive call structure for <i>apply</i> for example in Fig.4.5.	63
4.7	The result of <i>apply</i> ($+, B_f, B_g$).	63
4.8	A simple directed graph, where each edge has a probability of being true.	64
4.9	A BDD that computes the total probability of the path ae	66
5.1	A simple BDD	72
5.2	Implementation Method	77
5.3	Same-step Correlation Generated Hog1 Promoter Network	78
5.4	Learning Curves for Test Data Mean Square Error, resp for VV ($\Delta \Rightarrow$ Δ), VL ($\Delta \Rightarrow L$) and LL ($L \Rightarrow L$).	79
5.5	Speedup graph based on Table 5.5 results	81
5.6	Speedup graph based on Table 5.6 results	82
5.7	Speedup graph based on Table 5.7 results	83

Acronyms

AIK - Akaike's Information Criterion

BDD - Binary Decision Diagram

BN - Bayesian Network

COD - Coefficient of Determination

CTL - Computation Tree Logic

CUDD - CU Decision Diagram

DBN - Dynamic Bayesian Network

DNA - Deoxyribonucleic acid

dsRNA - Double-stranded Ribonucleic acid

EM-algorithm - Expectation–Maximization algorithm

GE - Gene Expression

GRN - Gene Regulatory Network

GP - Genetic Programming

Lac - Lactose

LMS - Least Mean Square

LNA - Linear Noise Approximation

LTM - Linear Transcription Model

MAPK - Mitogen-activated protein kinases

Mg²⁺ - Magnesium

miRNA - Micro Ribonucleic acid

mRNA - Messenger Ribonucleic acid

MWSLE - Minimum Weight Solutions to Linear Equations

Na⁺ - Sodium

NaCl - Sodium Chloride

ncRNA - Non-coding Ribonucleic acid

OBDD - Ordered Binary Decison Diagram

ODE - Ordinary Differential Equation

PBN - Probabilistic Boolean Network

PDE - Partial Differential Equation

PIN - Protein Interaction Network

PRM - Probabilistic Relational Model

PSN - Protein Signaling Network

RNA - Ribonucleic acid

RNAseq - Whole Transcriptome Shotgun Sequencing

rRNA - Ribosomal Ribonucleic acid

SDE - Stochastic Differential Equation

siRNA - Small interfering Ribonucleic acid

SRL - Statistical Relational Learning

SVD - Singular Value Decomposition

Tet - Tetracycline

tmRNA - Transfer-messenger Ribonucleic acid

tRNA - Transfer Ribonucleic acid

YAP - Yet Another Prolog

Chapter 1

Introduction

*If you don't know where you are
going, you'll end up someplace else.*

Yogi Berra

1.1 Introduction

Many major cellular decisions involve changes in transcriptional regulation. With the advent of high-throughput technologies and advanced measurement techniques molecular biologists and biochemists are rapidly identifying components of these networks and determining their biochemical activities, but understanding these complex multicomponent networks that govern how a cell will respond to diverse environmental cues is difficult using intuition alone. In this work, our goal is to build a probabilistic logical model that can aid in uncovering the structure and dynamics of such networks and how they regulate their targets. Gaining insight into transcriptional regulation is important not just for understanding the fundamental biological processes, but also for advancing research.

A cell responds to environmental changes by detecting molecules that bind to receptors on the surface of the cell and transmits this information to proteins within the cell by activating a cascade of molecular events. These signaling networks are typically studied by measuring molecular events after treatments that stimulate or perturb key elements in the network at the mRNA or gene level. Combining these measurements with phenotypic response enables the study of these treatments on the architecture and function of the underlying signaling networks as well as the relationship between the network behavior and phenotypic response [1]. In this work, to infer the architecture and function of the underlying signaling network of budding yeast, we decided to focus on the pathways activated by MAPK¹ Hog1 during osmotic stress response. According to previous results [2], this pathway interacts with the general stress (**Msn2/Msn4**) pathways, so we consider the genes belonging to both pathways in the following study.

Despite the challenge of inferring genetic regulatory networks from gene expression data, various computational models have been developed for regulatory network analysis. Examples include approaches based on logical gates [3, 4], and probabilistic approaches, often based on bayesian networks [5]. On one hand, logic gates provide a natural, intuitive way to describe interactions between proteins and genes. On the other hand, probabilistic approaches can handle incomplete and imprecise data in a

¹Mitogen-activated protein (MAP) kinases are serine/threonine-specific protein kinases belonging to the CMGC (CDK/MAPK/GSK3/CLK) kinase group. These kinases regulate gene expression, proliferation, differentiation, mitosis, cell survival - among many others.

very robust way.

Our main contribution is in introducing a model that combines the two approaches. Our approach is based on the probabilistic logic programming language ProbLog [6, 7]. In this language, we can express true logical statements (expressed as *true rules*) about a world where there is uncertainty over data, expressed as *probabilistic facts*. In the setting of gene expression, this corresponds to establishing:

- a set of *true rules* describing what are the possible interactions existing in a cell;
- a set of *uncertain facts* describing which possible rules are applicable to a certain gene or set of genes.

Given time-series gene expression data, we want to choose the probability parameters that best describe the data. Our approach is to reduce this problem to an optimization problem, and use a gradient ascent algorithm to estimate a local solution [8] in the style of logistic regression. We further contribute an efficient implementation to this algorithm that computes both probabilities and gradients through binary decision diagrams (BDD).

We evaluate our approach by using it to study expression data on an important gene-expression pathway, the Hog1 pathway [2]. It is well known that under conditions of osmotic stress, the protein kinase Hog1, and the paralogous proteins Msn2 and Msn4 interact to create a response that involves the expression of a large number of proteins. We model these pathways by also incorporating the two transcription factors activated by Hog1: Hot1 and Sko1.

1.2 Thesis Roadmap

This thesis is divided into a total of six chapters. In **Chapter 1** we present a small introduction to the biological process and the problem of discovering gene networks. In **Chapter 2** we explain some important concepts for understanding the work in this thesis, we talk about DNA, RNA, gene regulatory networks and also about expression data, the data we use for our models. Next, in **Chapter 3** we talk about related work in the area, presenting some of the techniques that can be used for creating gene regulatory networks. **Chapter 4** is a brief introduction to logic programming and also concepts that are necessary for understanding the processes behind the calculations. A short introduction to Problog is also given. In **Chapter 5** we explain how to represent

gene networks with Problog, how we perform our experiments and develop our models and we show the results we obtained. Last, **Chapter 6** presents the conclusion we took from our work and we also talk about the future work we intend to perform.

Chapter 2

Biology Background

Every biologist has at some time asked 'What is life?' and none has ever given a satisfactory answer. Science is built on the premise that Nature answers intelligent questions intelligently; so if no answer exists, there must be something wrong with the question.

Albert Szent-Györgyi

In this chapter we will explain some fundamental biological concepts that will be important for understanding this work. We will start by discussing DNA, one of the fundamental basis of life, then we will survey RNA, which is usually created by DNA (although there are some exceptions in retro-viruses such as Human Immunodeficiency Virus (HIV)). Afterwards an introduction to gene regulatory networks will be presented and finally we will present the data used for this type of work, where it comes from, how it is obtained and what it provides.

One of the goals in computational biology is to understand the regulation processes in a cell at the gene level, which can in turn lead to specific interventions for genetic diseases and drug design [9]. The global understanding of an organism in great detail is a long term goal, as is the understanding and analysis of the information flow in a cell (Fig. 2.1)

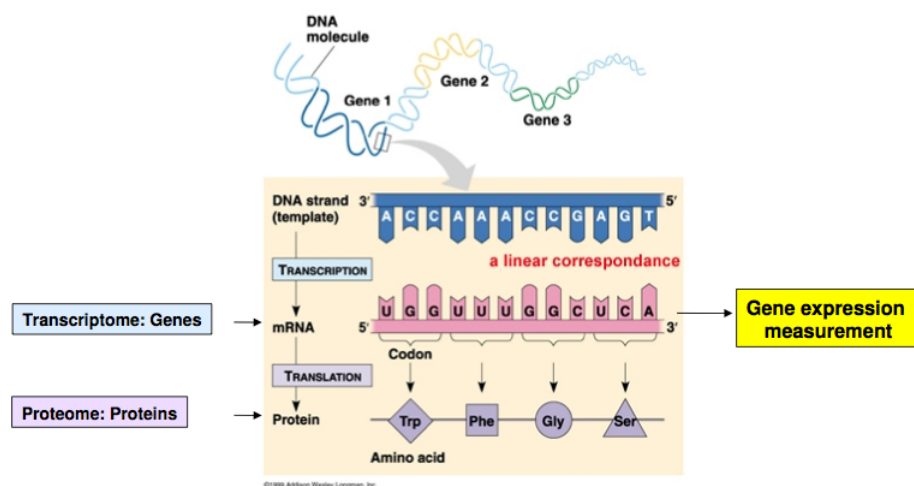


Figure 2.1: The flow on information in a cell, a.k.a. Central dogma of biology

2.1 DNA

The deoxyribonucleic acid (DNA) is a nucleic acid containing the genetic instructions used in the development and functioning of all known living organisms (with the exception of RNA viruses). DNA contains segments known as genes that carry the genetic information. This acid is one of the three major macromolecules that are essential for all known forms of life, the other two macromolecules are RNA and

proteins. DNA is composed of two nucleotides (molecules that make up the individual structural units), with backbones made of sugars and also phosphate groups [10] that are joined by ester (chemical compound consisting of a carbonyl adjacent to an ether linkage) bonds. The sugars are joined together by phosphate groups that form phosphodiester bonds (Fig. 2.2) between the third and fifth carbon atoms of adjacent sugar rings. These bonds are asymmetric, hence a strand of DNA has a direction. Notice that the two strands are anti-parallel, that is, they run in opposite directions to each other. Attached to each sugar is one of the four nucleobases of DNA: Adenine (A), Cytosine (C), Guanine (G), Thymine (T). These nucleobases are

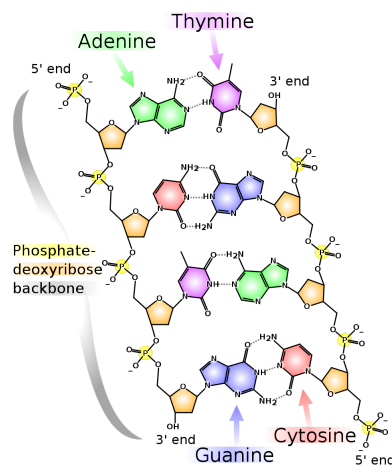


Figure 2.2: DNA Chemical Structure

classified into two types: the purines, A and G, are fused five- and six-membered heterocyclic compounds, and the pyrimidines, are six-membered rings C and T [11]. Information is encoded by this sequence of bases along the backbone, and one can say that DNA is a long polymer made from repeating units called nucleotides[12], [13]. In order to read this information, the genetic code that specifies the sequence of amino acids within proteins is used. The reading of the code is done through a process called *transcription* (the process of creating a complementary RNA copy of a sequence of DNA), we will describe it further in the next section. DNA is organized within cells by using chromosomes, which are duplicated in the process of DNA replication. Due to this replication, each cell has its own complete set of chromosomes. On the other hand, DNA is organized and compacted within the chromosomes by chromatin proteins, creating a type of compact structures that are responsible for guiding the interactions between DNA and other proteins and which help control which parts of DNA are transcribed.

Usually DNA exists as a pair of molecules that are held tightly together like two

entwined vines, forming the known shape of a double helix [14] as shown in Fig. 2.3. The nucleotide repeats contain both the segment of the backbone of the molecule, which holds the chain together, and a nucleobase, which interacts with the other DNA strand in the helix[15]. As we discussed earlier, DNA has two asymmetric ends which are known as the 5' (five prime) and 3' (three prime) ends. The five prime has a terminal phosphate group and the three prime has a terminal hydroxyl group as shown in Fig. 2.2. In a DNA double helix there is a property called complementary

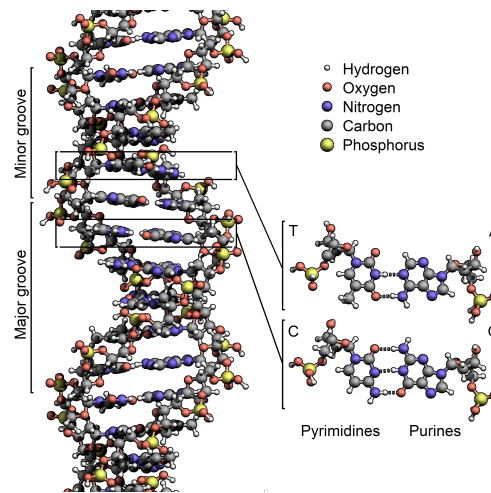


Figure 2.3: DNA double helix structure

base pairing, which is defined by the fact that each type of nucleobase on one strand normally interacts with just one type of nucleobase on the other strand, A only bonds to T and C only bonds to G forming the base pairs shown in Fig.2.4. The two base

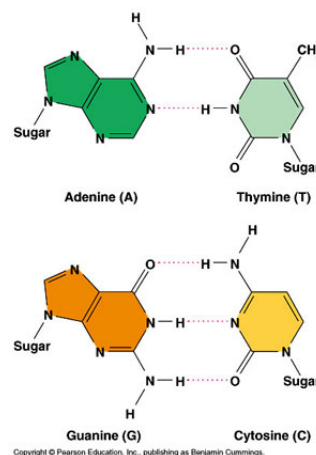


Figure 2.4: DNA base pairs

pairs form different numbers of hydrogen bonds, namely AT forms two hydrogen bonds

and *GC* forms three hydrogen bonds, therefore, DNA with high *GC*-content is said to be more stable than DNA with low *GC*-content. Due to the non-covalent property of hydrogen bonds, these can be broken and rejoined easily. This enables the strands of DNA to be pulled apart like a zipper, using high temperature or even a mechanical force [16]. Notice that, all the information in the double-stranded sequence of a DNA



Figure 2.5: DNA zipper

helix is duplicated on each of the strands, which is vital in DNA replication (the basis for biological inheritance). This reversible interaction between complementary base pairs is critical for all the functions of DNA in living organisms.

2.2 RNA

The Ribonucleic acid (RNA) is a biologically important type of molecule that consists of a long chain of nucleotide units, it is one of the four major macromolecules (along with lipids, carbohydrates and proteins) and essential for all known forms of life, but it differs from DNA: RNA is usually single-stranded in the cell, while DNA is usually double-stranded; RNA nucleotides contain ribose while DNA contains deoxyribose (a type of ribose that lacks one oxygen atom), as there is no hydroxyl group attached to the pentose ring in the 2' position in DNA; and RNA has the base uracil rather than thymine that is present in DNA. The sequence of nucleotides allows RNA to encode genetic information. RNA is transcribed from DNA by enzymes called RNA polymerases and is generally further processed by other enzymes. RNA is in the center of protein synthesis, as a type of RNA called messenger RNA (mRNA) carries information from DNA to cellular structures called ribosomes. It is also known that many viruses use RNA instead of DNA as their genetic material.

Ribosomes are made from ribosomal RNAs (rRNAs) and proteins, which come together to form a type of *molecular machine* that is able to read the mRNAs and *translate* their information into proteins. This is attained by a process that uses transfer RNA (tRNA) molecules to deliver amino acids to the ribosome, where ribosomal RNA (rRNA) links amino acids together to form the proteins. Other types of RNA molecules play an active role in cells by controlling gene expression, sensing and

communicating responses to cellular signals or catalyzing biological reactions. Most RNA molecules are single-stranded and can adopt very complex three-dimensional structures as described next.

2.2.1 Structure

Each one of RNA's nucleotides contain a ribose sugar, this sugar has carbons numbered from 1' through 5'. A base, usually Cytosine (C), Adenine (A), Uracil (U), Guanine(G), is attached to the 1' position. Cytosine and uracil are pyrimidines, guanine and adenine are purines. Attached to the 3' position of one ribose and to the 5' position of the next is a phosphate group, these groups have a negative charge at physiological pH, which makes RNA a charged molecule (polyanion). It is known that the bases may form hydrogen bonds between guanine and uracil, between cytosine and guanine and also between adenine and uracil. Besides these bases, there are numerous modified bases and sugars in mature RNAs, some example are the *Pseudouridine* - in this base the linkage between uracil and ribose is changed from a *C-N bond* to a *C-C bond* (usually found in the *TC loop* of tRNA [17]) and the *Hypoxanthine* - a deaminated adenine base whose nucleoside is called inosine (I), which plays a key role in the wobble hypothesis (a non-Watson-Crick base pairing composed by two nucleotides in RNA molecules) of the genetic code. One of the important structural

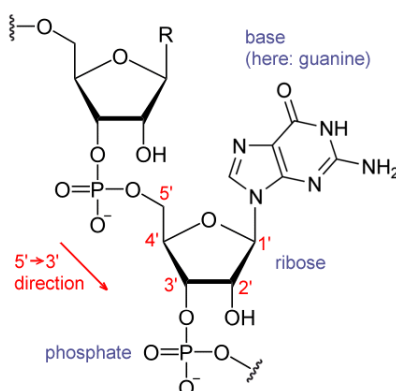


Figure 2.6: RNA Chemical Structure

features that distinguishes RNA from DNA is the presence of a hydroxyl group at the 2' position of the ribosome sugar, which causes the helix to adopt an A-form geometry instead of the B-form that is commonly observed in DNA [18]. The presence of the 2'-hydroxyl group also has another consequence, in conformationally flexible regions of a RNA molecule (not involved in formation of a double helix) it can chemically attack

It is known to exist about 100 naturally occurring modified nucleosides, although the specific role of these modifications in RNA are not yet fully understood. Many of the post-transcriptional modifications occur in highly functional regions, which implies that they are important for normal function. Frequently, the functional form of single stranded RNA molecules require a specific tertiary structure which is provided by the secondary structural elements (hydrogen bonds within the molecule). The result of this are the recognizable secondary structure known as *hairpin loops*, *bulges* and also *internal loops*, as shown in Fig. 2.7.



2.2.2 Synthesis

RNA synthesis, or transcription is under the control of the enzyme RNA *polymerase*. The first step that this enzyme takes is to find the start of the gene on the coding

strand of the DNA, as the DNA has lots of genes strung out along the coding strand (the enzyme has to pick the right strand and identify the beginning of each gene). This is done by recognizing and binding with one or more short sequences of bases (also known as promoter sequences, Fig. 2.8) "upstream" of the start of each gene. The transcription process (Fig. 2.9) is composed of the following steps:

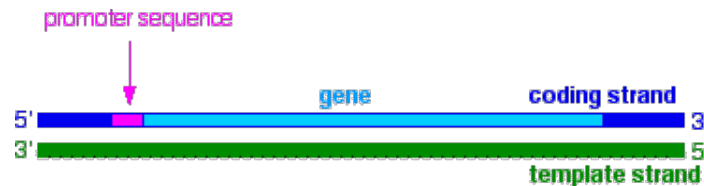


Figure 2.8: Promoter Sequences

- The DNA double helix is unwound due to the helicase activity of the enzyme
- Enzyme progression along the template strand following the 3' to 5' direction
- Synthesis of a complementary RNA molecule

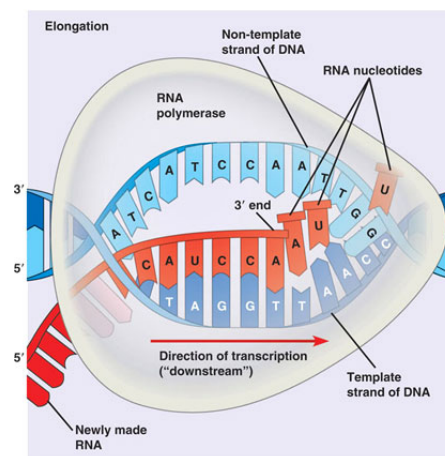


Figure 2.9: RNA Synthesis

The end of RNA synthesis is indicated by the DNA sequence. RNAs are often modified by enzymes after transcription. There are also a number of RNA-dependent RNA polymerases that use RNA as their template for synthesis of a new strand of RNA. For instance, a number of RNA viruses use this type of enzyme to replicate their genetic material.

2.2.3 Types

There are several types of RNA, as discussed before:

- **Messenger RNA (mRNA)** - molecule in cells that carry codes from the DNA to the sites of protein synthesis (the ribosomes). Information in DNA cannot be decoded directly into proteins, and thus it is first transcribed, or copied, into mRNA. Each molecule of mRNA encodes the information for one protein (or more than one protein in the case of a bacteria), with each sequence of three nitrogen-containing bases in the mRNA. The mRNA takes the copy of the blueprint to the ribosome where it is used to build the protein.

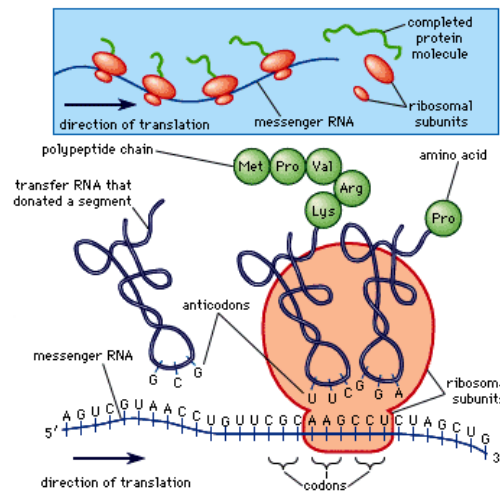


Figure 2.10: Messenger RNA

- **Ribosomal RNA (rRNA)** - these molecules form the structural and functional components of ribosomes, the subcellular units responsible for protein synthesis [21]. This type of RNA is the catalytic component of the ribosomes. rRNA

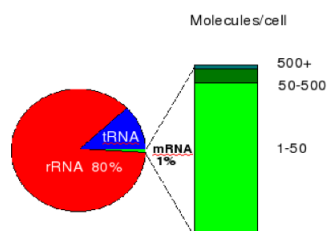


Figure 2.11: RNA abundance in cells

constitutes approximately 80% to 85% of the total RNA in a cell (shown in Fig. 2.11). In eukaryotes, rRNA synthesis occurs in the nucleolus, a specialized structure within the nucleus.

- Transfer-messenger RNA (**tmRNA**) - molecule of RNA (found in many bacteria and plastids) that has dual functions as both a transfer RNA and a messenger RNA. As a tRNA, it recognizes and binds ribosomes stalled by aberrant mRNAs with the help of its protein partner SmpB. As an mRNA, it adds a degradation tag to protein fragments, targeting them for proteolysis. Two types of tmRNAs are known: single-chain tmRNAs and two-piece tmRNAs.

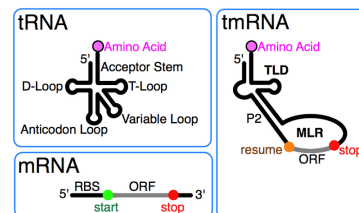


Figure 2.12: Transfer-messenger RNA

- Non-coding RNA (**ncRNA**) - functional molecule that is not translated into protein. ncRNAs have been shown to regulate important biological processes that support normal cellular functions, but relatively little is known about the general structure, function, and transcriptional control of ncRNAs, and even less about their potential functions as a group or as a single entity [22].
- Other types of RNA are the following: Double-stranded RNA (**dsRNA**) - RNA with two complementary strands, similar to the DNA found in all cells. dsRNA forms the genetic material of some viruses (dsRNA viruses). dsRNA such as viral RNA or siRNA can trigger RNA interference in eukaryotes, as well as interferon response in vertebrates. MicroRNA (**miRNA**) - small type of dsRNA molecules that regulate translation in eukariotic cells, it is related to RNAi. Unlike other small RNAs, the genes for miRNA are transcribed by RNA polymerase II. MicroRNAs are not only used in eukariotic cells but also by some of the more complex viruses that infect them. Short-interfering RNA (**siRNA**) - short dsRNA fragments that are known to bound by the the RNA-induced silencing complex¹ (RISC) [23]. It is class of double-stranded RNA molecules

¹RNA-induced silencing complex, or RISC, is a multiprotein complex that incorporates one strand of a small interfering RNA (siRNA) or micro RNA (miRNA). RISC uses the siRNA or miRNA as a

that play many roles, being the most notable the RNA interference (RNAi) pathway.

2.3 Gene Regulatory Networks

Every cell is a complex processor of information, as it is able to integrate and respond to multiple signals in a robust way. The mechanism that cells use to achieve this are remarkable and in a large number. One can make an analogy to electrical circuits, that is, we can decompose the high complexity involved in cellular response into modules that connect to each other by input and output signals [24]. We can take this comparison a little further, as genetic network engineers manipulate living organisms using the biological equivalent of transistors and inverters. A possible description of gene networks is circuits of interconnected functional modules, each consisting of specialized interactions (information flow) between proteins, RNA, DNA and small molecules. The importance of these modules (components) can be exemplified by their

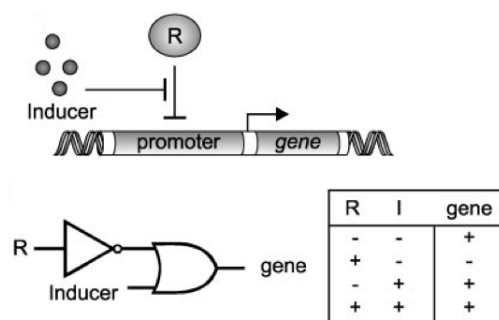


Figure 2.13: Example of an architecture of the inducible gene network

use in networks that function in cells that have a higher complexity. An advantage of prokaryotic components such as repressors and their operating sites is that they can be transplanted into eukaryotic cells without any loss of their binding or function specificity. A second advantage is the avoidance of unwanted interference with the expression of non-targeted genes [24]. There has been a large development of switch systems based on the *Tet* or *Lac* repressor in a large number of organisms ([25], [26], [27], [28]). This development sets the stage for the construction of more complex gene networks.

template for recognizing complementary mRNA. When it finds a complementary strand, it activates RNase and cleaves the RNA.

In order to synthesize proteins that carry out specific functions in the cell, this information is extracted through a process called gene expression. On the other hand, gene regulation describes all the cellular processes which control the expression of proteins. This regulation can occur at the following steps of the DNA information extraction:

- Transcription initiation - start of the binding of RNA polymerase to the promoter in DNA
- Translation - the mRNA produced by transcription is decoded by the ribosome in order to produce a polypeptide, or a specific amino acid chain, that will later fold into an active protein.
- Modifications of mRNAs - modified to remove certain stretches of non-coding sequences called introns.
- DNA packing - process in which DNA and associated proteins are formed into a compact, orderly structure.
- Post-translational modifications of amino-acid sequences - this modification extends the range of functions of the protein by attaching it to other biochemical functional groups, making structural changes, or changing the chemical nature of an amino-acid.

There are many different gene expression patterns in a cell, they depend of factors like the state of the cell, nutrition, environment or even the cell type. The process of gene expression consists of translation and transcription, with the regulation of this expression taking place at different steps, but usually more during transcription (truest when talking about procaryotes like bacteria). As exons and introns are not distinguished in their mRNA molecules, an alternative splicing (one of the more relevant regulation processes taking place after the transcription step [29]) takes place, leading to different proteins from the same mRNA molecule. In order to build a model for gene regulation, one needs to specify the interactions (the ones that the dynamic behavior of gene expression can be explained by them) that should be captured by the model.

The transcription of a gene can be activated or inhibited by transcription factors which can bind to a site on the DNA near the promoter (the place where the following gene transcription is started) in a way that the transcription factor that binds inhibits the

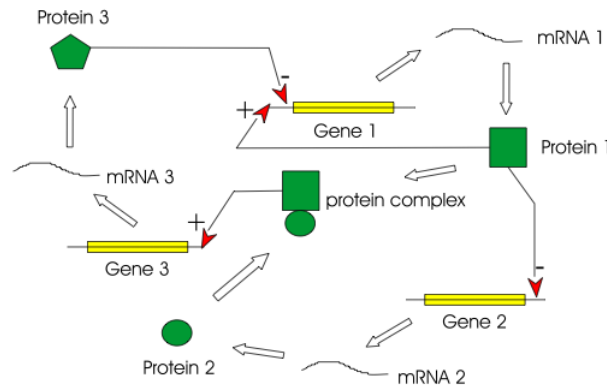


Figure 2.14: Gene Regulation: The activation or inhibition of a gene transcription by one or several transcription factors.

start of the transcription. In order to activate a transcription, transcription factors can absorb other molecules that bind to that particular site. We can see different possibilities of inhibition and activation in Fig. 2.14. The product of a gene can regulate its own transcription or of another gene, it can also regulate the expression with the help of other products of genes.

Regulatory proteins and their binding sites are also modular in that different domains from different proteins can be combined to yield hybrid proteins of novel function. The state in which a living organism is at any certain point of time is not only described by its genome, but also by its set of expressed *regulatory genes* and its concentration levels of the corresponding gene products. All the possible phenotypic states of a cell correspond to distinct gene expression patterns [30].

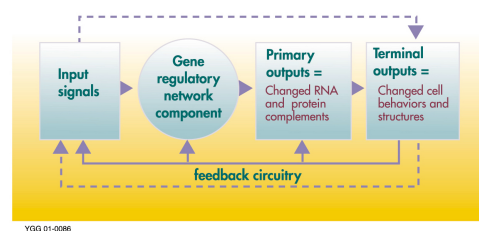


Figure 2.15: Processes in a Gene Regulatory Network

To sum all up, we can say that gene regulatory networks (GRNs) are interacting DNA-encoded regulatory subsystems in the *Genome* that have the function of coordinating the inputs from activators and repressors (*Transcription Factors*) during cell differentiation, development or even in response to environmental causes (natural or

not). GRNs function to specify expression of particular sets of genes for specific times, locations and conditions.

2.4 Expression Data

In order to sequence the human genome, methods for measuring the expression levels of single and all genes simultaneously in a genome had to be created. An existing challenge of Computational Biology is the analysis of thousands of measurements of one cell state in order to retrieve the useful information from it. To get these measures, Microarray Technology ² is applied. By using this technology, the researchers can observe the dependency of gene expression on different states of a cell and also on different environmental factors.

2.4.1 Data Measurement

The description of any model gets better as more relevant information becomes available. This seems trivial as huge amounts of information exist nowadays, but unfortunately for gene regulation that is not true, some of the data available is only for selected targets, which some of the times are not satisfactorily accessible and most of the data is very noisy. The advance in DNA-microarray technology permits to monitor thousands of genes in one experiment by measuring mRNA concentrations in a cell [31]. Each data point produced by a DNA microarray experiment represents the ratio of expression levels of a particular gene. The result, from an experiment with n genes on a single chip, is a series of n expression-level ratios. Typically, the numerator of each ratio is the expression level of the gene in the varying condition of interest, whereas the denominator is the expression level of the gene in some reference condition. The data from a series of m such experiments may be represented as a gene expression matrix, in which each of the n rows consists of an m -element expression vector for a single gene. The expression measurement is positive if the gene is induced (turned up) with respect to the reference state and negative if it is repressed (turned down). Every spot on the microarray has millions of copies of one probe in order

²An array is an orderly arrangement of samples where matching of known and unknown DNA samples is done based on base pairing rules. An array experiment makes use of common assay systems such as microplates or standard blotting membranes. The sample spot sizes are typically less than 200 microns in diameter usually contain thousands of spots

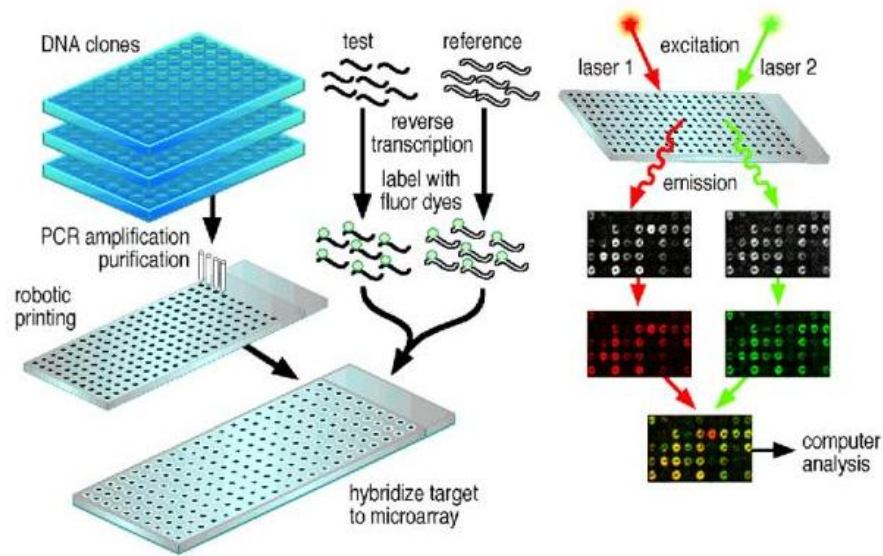


Figure 2.16: A general scheme for a microarray.

to measure not only the existence of specific mRNA in the test material, but also to measure its amount, making it interesting if time series are produced, such that the difference between two time points can give a hint for genes that are transcribed together or even a hint for some regulatory interactions.

Chapter 3

Related Work

*Knowing is not enough; we must
apply.*

Willing is not enough; we must do.

Johann Wolfgang von Goethe

Throughout time there has been a lot of research regarding gene regulatory networks, their interaction, modeling and prediction. A number of mathematical models have been created in order to capture the behavior of the system that is being modeled, and also to generate predictions that are then validated through experimental observation. In some cases, the created models made some accurate novel predictions that were validated afterwards, leading advances in the biological field. Sometimes, these new discoveries could have not been found if it was not for the mathematical model, as the scientists would have not considered doing the experiment in a laboratory environment.

The most common techniques applied to the models are differential equations, Boolean networks, Bayesian networks and graphical models. Logic-based modeling is seen as an approach lying midway between the complexity and precision of differential equations on one hand and data-driven regression approaches on the other [1]. In the next subsections we will have a look at some related work using those techniques.

Clustering algorithms have also been used since the first works in gene expression, they have been primarily used to group together genes with similar temporal expression patterns [32], some work using this type of algorithms are [33] [34] [35] [36] [37]. The motivation for using this type of algorithms is the idea that two genes that exhibit a similar expression pattern over time may be coregulated by a third gene or even regulate each other.

3.1 Differential equations models

Within the spectrum of modeling methods currently being applied to cellular biochemistry, models involving differential equations bear the closest relationship to the underlying biochemical rate laws, and thus they are one of the most important modeling formalisms in mathematical biology.

Sets of coupled ordinary differential equations (ODEs) can effectively represent chemical reactions when the number of molecules is large and mass action approximations are appropriate. Partial differential equations (PDEs) add the ability to represent spatial gradients, and stochastic methods make it possible to analyze systems in which the number of molecules is small. Networks of differential equations can model the temporal and spatial dynamics of biochemical processes in considerable detail, making it possible to study chemical mechanisms and to predict network dynamics

under various conditions. What makes this type of equations adequate for gene expression is that they can also model complex dynamic behavior like oscillations, cyclic patterns, multi-stationary and switch-like behavior [9]. However, the topology (how species interact, their patterns of interaction) of ODE- and PDE-based models must be specified in advance, and model output is strongly dependent on the values of free parameters (usually the initial protein concentrations and rate constants). This parameters estimation is a computationally intensive task requiring substantial data. As networks get larger, ODE modeling becomes more and more challenging, and models that attempt to capture real biological data are currently limited to a few dozen components.

When using these equations, the first step is to find the ones that are more adequate to the problem at hand. In order to do so, we need large amounts of data (to infer the unknown parameters) and also the processes in the system that we are studying. Knowing which gene regulates which one, the way of the regulation, the degradation and the maximal production rates of the associated proteins is also very valuable knowledge.

Describing a gene network in terms of differential equations has 2 advantages [38]:

- It describes gene interactions in an explicitly numerical form
- Due to the large amount of information in a system of differential equations, other networks can be derived from it

There has been some development using differential equations to create models for gene regulation. Chen *et al.* [39] built a Linear Transcription Model based on linear differential equations and two algorithms to solve the differential equations. For their model they consider *nRNA* as well as protein data and use an equation like $y = My$, where $y(t)$ contains the protein and *mRNA* concentrations at time point t and M is a constant matrix, describing the influence that the variables have on each others change of concentration. Their approach had limitations, the model does not consider time delays in transcription or translation leading to a very significant reduction of the problem complexity. Another significant limitation comes from ignorance of other regulators. Despite these limitations, the LT model is able to clearly capture more features of gene expression than other models. An approach with MWSLE (Minimum Weight Solutions to Linear Equations) was also used, but the actual number of gene regulators was much larger than expected and the solution would probably be computationally intractable.

Following Chen *et al.* [39] work, De Hoon *et al.* [40] built a model focusing only on *mRNA* values. In their work they proposed to infer the degree of sparseness of the gene regulatory network from the data they had by determining which coefficients are nonzero by using Akaike's Information Criterion for the task. The algorithm created by them estimates matrix M on the basis of maximum likelihood estimation and then uses AIK to estimate both position and number of nonzero parameters that the matrix contains. Their method allows for loops to be present in the network, these loops are only found if the measured data warrant them, the existence of them is not dictated. With this method they were able to present some interesting results on the transcription of *Bacillus Subtilis* [38]. The drawback of both models is that the matrix is assumed to be constant, leading to a failure to capture a lot of phenomenon in the dynamic behavior of a real organism.

Sakamoto *et al.* [41] developed a model that describes the expression change for the i -th gene as $X_i = f_i(x_1, \dots, x_n)$ ($i = 1, 2, \dots, n$) with possible nonlinear functions f_i , where X_i is the state variable and n is the number of components in the network. In order to identify the system of differential equations, they use Genetic Programming (GP) to evolve the right hand side of the equation from the observed time series of the gene's expression. They apply and combine two different methods of optimization, GP and Least Mean Square (LMS), the reason to do so is that GP is capable of finding a desirable structure effectively, but when one needs to optimize the constants or coefficients, ordinary GP is not always effective as it relies mainly on the combination of randomly generated constants. By using LMS one can explore the search space more effectively. Through using these methods Sakamoto *et al.* were able to successfully infer their network by several experiments, and were able to capture more behaviors than a linear model. One has to bear in mind that there may exist more than one solution for the target, that is, like many other models, a solution which fits the time series in a good way is not necessary the unique.

Gebert *et al.* [9] also built a model with differential equations in which they captured the most relevant regulating interactions and calculated the parameters for the model from time-series data. In order to do so they used piecewise linear differential equations, that are originated from a decomposition of the state space into cuboids. They base their model on the assumption that regulation between genes can be described using piecewise linear functions. On their model, every single cuboid, differential equation is turned into a ordinary linear equation that can be solved analytically. They did have a need to solve the problem of estimating the parameters for each linear equation, leading it to an optimization problem which was resolved restricting

the solution space, that is, they included biological knowledge about the regulatory network. Although their model gets good results it is not yet ready for all organisms, as it requires a lot of input data to determine all unknown parameters (still not available).

Also in the field of differential equations we have the work of Fearnhead *et al.* [30] in which they use Linear Noise Approximation (LNA) as a compromise between the ODE and stochastic differential equations (SDE) models. Usually inference is performed by approximating the dynamics through an ODE or a SDE, but we are presented with two known problems: ODEs ignore the stochasticity in the true model and this can lead to inaccurate inferences, on the other hand, SDEs are more accurate than the former but they are harder to implement due to the transition density if the SDE models being generally unknown. In their work they use LNA as an approximation for inference due to the fact that models based on ODEs are usually appropriate for very large systems in which the stochasticity in the evolution is small, and also because SDE models are more appropriate for medium-size systems and these lead to sensible estimates of the reaction rates [30]. Due to these reasons and also knowing that the inference for SDE models is not trivial they use an alternative approximation which first appeared on [42] and [43]. The LNA is obtained from two steps, first we must approximate the dynamics by a system of ODEs and second we must model the evolution of the state about the ODE deterministic solution using for this task a linear SDE. Although simulation suggests that the LNA approach has similar accuracy to the SDE, one must bear in mind that by using LNA the stochastic model for the states is a Gaussian process, which means that the mean and covariance of the transition densities can be found using a system of differential equations. Results presented by Fearnhead *et al.* [30] suggest that the usage of LNA gives more accuracy than approximating the underlying model using an ODE.

Akutsu *et al.* [44] present a model that can be considered as an intermediate model between differential equations and boolean networks (the algorithms are based on linear differential equations). In their model, regulation rules are embedded in network structures and are represented as quantitative rules. One of the algorithms presented by them can be applied to S-systems [45] [46] [47], these systems are based on a particular kind of nonlinear differential equations. Although their methods seem interesting they have some drawbacks, first it requires a lot of time series data from different sets of initial values (different conditions or even environments), this may not always be available, the second drawback is that complex enzymatic reactions can not be handled directly. In order to infer correctly they would have to focus only on a part of the network, as this would require less time series data.

In some other studies presented by di Bernardo *et al.* [48], Gardner *et al.* [49] and Bansal *et al.* [50] there has been a development of ODE-based algorithms that use a series of steady-state RNA expression measurements or even time-series measurements. There are also some other cases based on ODEs that can be found in the literature, some of them are the work by van Someren *et al.* [51], Tegner *et al.* [52], Bonneau *et al.* [53], D'haeseleer *et al.* [54] and de Jong *et al.* [32].

3.2 Logical models

Most often GRNs (Gene Regulatory Networks) are represented by using logical models. In these models, the gene-expression measurements are converted to discrete levels (*on/off*), but this process usually introduces inconsistencies into the data. It is believed that the reconstruction of a logical GRN that is able to minimize the errors is NP-complete, which suggests that an efficient algorithm to solve this problem may not exist. Even with these problems that have been several approaches that propose applying logical models. In deed boolean networks are used on many of the models present in the literature, we can have the state of a gene converted to a boolean variable that is in one of two possible states, active (1 , *on*) or inactive (0 , *off*), meaning that the gene products are present or not. The interactions between elements of the network can be represented by boolean functions that are used to calculate the resulting state of a gene due to the activation of other genes. All of this culminates in the creation of a boolean network, as we can see in Fig. 3.1. An approach that includes boolean

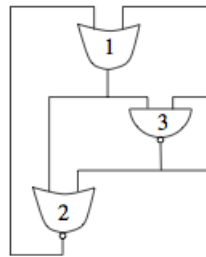


Figure 3.1: Example of a Boolean network

networks is the approach by Akutsu *et al.* [55], in their work they define a GRN through the boolean network $G = (V, F)$ where V is the set of nodes and the set $\{F = f_v | v \in V\}$ of boolean functions assigned to the nodes, where the network may have cycles. They also define a global state of G as being a mapping $\psi : V \rightarrow \{0, 1\}$ in which each global state must satisfy $\psi(x_i) = 1$ and $\psi(y_i) = 0$ under an experiment

$\langle x_1, \dots, x_p, \neg y_1, \dots, \neg y_q \rangle$. In order to identify a GRN from observed global states they investigate the number of experiments together with the cost of each experiment. This work lead to two important problems obtained from the identification of GRNs:

- *consistency* (checking if a network $G' = (V', F')$ coincides or not with the underlying GRN)
- *stability* (G is stable if there exists a global state consistent with all gene regulation rules) of the network

Also, in their work they have derived upper and lower bounds on the required perturbations for the boolean networks.

Another approach using boolean networks is the work developed by Ideker *et al.* [56]) in which they present two methods (*Predictor* and *Chooser*) for inferring a genetic network given gene expression measurements. The *Predictor* method is used to infer hypothetical boolean networks consistent with a profile that was generated previously by exposing the network of interest to a series of genetic or biological perturbations. After the inference of several networks, the *Predictor* method returns the ones that are most parsimonious (the ones having the fewest number of interactions). The next step is done by the *Chooser* method that adds an additional perturbation experiment in order to discriminate along the set of previously obtained hypothetical networks. These perturbations are added in a clever way by using a function that is based on entropy to optimally reduce the number of remaining hypothetical networks. One other feature of these two methods is that they can be used interactively and iteratively.

Standard boolean networks have a very salient limitation, their inherent determinism. One can look at this from two points of view, the conceptual point of view or the empirical point of view. From the former, one must bear in mind that is likely that the regularity of genetic function and interaction known to exist is not due to hard-wired logical rules, as for the latter, it takes in account that one logical rule per gene may to incorrect results when these rules are being inferred from gene expression measurements. To address these considerations Shmulevich *et al.* [57] introduced a new model class, the Probabilistic Boolean Networks (PBNs), which have the properties of boolean networks and can cope with uncertainty in the data and model selection. The basic idea of their work is to extend the boolean network in order to accommodate more than one possible function for each node. For this, they have a set $F_i = \{f_j^{(i)}\} j = 1, \dots, l(i)$ that corresponds to each node x_i and where each $f_j^{(i)}$ is a possible function that determines the value of gene x_i and $l(i)$ is the

number of possible functions for gene x_i . For inferring they use a method based on the Coefficient of Determination (COD) which produce a number of candidate predictors for each target gene. In this model, the approach was to probabilistically 'create' good predictors so that each predictor contribution is proportional to its determinative potential, this is close to our own approach.

Also in this field of logical models, but on a more formal approach, we have the work by Bernot *et al.* [58] where they provide a formal way to treat temporal properties of biological regulatory networks, expressed in computational tree logic leading to the possibility of building all the models that satisfy a set of given temporal properties. Their work allows biology to take advantage from all the available formal methods from computer science, temporal properties can be checked against models using Computation Tree Logic (CTL) and model checking.

Still in the formal approach, we have the work by Batt *et al.* [59] that validates models of GRNs by addressing the challenge matching model predictions and experimental data, taking also in account a reliable and efficient comparison between the observations and the predictions. The qualitative modeling and simulation method that they use is based in a refinement of previous work [60]. What is new about their work is that they use model-checking techniques to attend the problem that the state transition graphs that are generated by qualitative simulation may become very large in a way that it may be prohibitive for interesting biological networks.

An interesting work involving logical analysis it the one presented by Thomas *et al.* [61] where they present a logical method for the analysis of the complex dynamics of regulatory networks in terms of feedback circuits. The feature that distinguishes the most their work from others is that the logical method introduced by them is fully asynchronous, that is, current variables are discrete, but time is continuous. Besides this work presented by Thomas *et al.*, they also have published other interesting ones related to logical models, their analysis and creation [62] [63] [64][65].

In recent work by Handorf *et al.*, we are presented with a new method for an automated generation of boolean network models from curated mechanistic network databases. This straight forward method translates into a good gain that is even more important in the context of the fast growing amount of interactions available in these databases. Despite of this great advantage, there is also a drawback that is common to all boolean approaches, interactions strengths and concentrations can not be properly covered by *TRUE* and *FALSE* values.

Further work has been developed in the area of GRNs using logic models [66] [67] [68] and even logic tools to model and analyze signaling pathways [69].

3.3 Other approaches

At the other extreme, a very active field in computing graphical representations of biological networks [70] [71] through literature analysis or through identification of correlations in high-throughput data has emerged. In these graphs, termed protein interaction networks (PINs or interactomes) or protein signaling networks (PSNs), genes and proteins are represented by nodes and potential interactions by edges (links). The edges can be directional or not and signed (inhibitory/activating) or not and typically represent a wide range of interaction modes from direct physical binding to correlated gene expression or integrated database entries. Graphs are an attractive way to summarize diverse relationships among large numbers of biomolecules across multiple organisms, but they are not executable per se and cannot be used to compute input-output relationships. Moreover, network graphs rarely take into account dynamic changes in signaling activities, cell type-specific biochemistry, or context-dependent variations.

Despite the difficulty of deciphering genetic regulatory networks from microarray data, numerous approaches to the task have been quite successful. Friedman *et al.* [5] were the first to address the task of determining properties of the transcriptional program of *S. cerevisiae* (yeast) by using Bayesian networks (BNs) to analyze gene expression data. Pe'er *et al.* [72] followed up that work by using BNs to learn master regulator sets. Other graphical approaches are Tanay and Shamir [73], Chrisman *et al.* [74]). The former implemented a new software platform called *Genesis* to enable analysis of available transcription profile data sets and target pathways. The latter present a Bayesian framework which combines information from several different sources and makes the correct causal inferences with small sample sizes.

The methods above can represent the dependence between interacting genes, but they cannot capture causal relationships. Pe'er *et al.* [72] ingeniously proposed the use of microarray experiments in which specific genes have been deleted (*knockout*) in yeast to obtain causality. The use of perturbations such as gene deletion mutants can allow the BN learning algorithm to learn a directed edge that suggests direct causal influence. This approach of combining observational and interventional data delivered promising results. Unfortunately, a complete library of gene knockouts are

not yet available for organisms other than yeast. The advent of small interfering RNA (siRNA) can be used to reduce the expression of a specific gene in organisms other than yeast, however, siRNA does not guarantee complete silencing of the gene. Ong *et al.* [75], proposed that the analysis of time series gene expression microarray data using Dynamic Bayesian networks (DBNs) could allow to learn potential causal relationships. DBNs are usually based on discrete models [76] [77] [78] or on continuous time models [79].

Another approach, is the work by Perrin *et al.* [80]. In their work they used penalized likelihood maximization in EM-algorithms to learn the parameters for a DBN. Dojer *et al.* [81] also apply DBNs, but this time in the context of perturbation experiments. With the incorporation of this type of data they are able to check that the quality of inferred networks dramatically improves.

A different approach is presented by Yeung *et al.* [82] who propose a scheme to reverse-engineer gene networks using Singular Value Decomposition (SVD) to construct a set of candidate solutions and afterwards applies robust regression to identify the solution with the smallest number of connections.

Last, we refer the reader to Madeira *et al.* [83], a work where biclustering algorithms for biological data analysis are described and analyzed. Biclustering algorithms have been used for some time now, the first time the term was used in gene expression data analysis was by Cheng *et al.* [84]. The main difference between these type of algorithms and the simple clustering algorithms is that the second ones can be applied to columns or rows of the data matrix (each at a time), but the first algorithms can perform clustering on the two dimensions (rows and columns) at the same time - meaning that with biclustering produces a local model while clustering produces a global model. With this we can say that the goal of biclustering algorithms is to identify genes subgroups and conditions subgroups by performing clustering on the rows and columns of the gene expression data matrix at the same time. Contrary to the clustering algorithms, biclustering is able to find sets of genes that have similar activity under a specific set of conditions.

Chapter 4

Introduction to Logic Programming

Logic is not a body of doctrine, but a mirror-image of the world. Logic is transcendental.

Ludwig Wittgenstein

In this chapter we will talk about Propositional logic (also known as sentential logic, is that branch of logic that studies ways of combining or altering statements or propositions to form more complicated statements or propositions) as a small introduction prior to talking about Binary Decision Diagrams (data structures for representing the semantics of a formula in propositional logic) and Ordered Binary Decisions Diagrams of which we show three algorithms that can be performed on them (*Reduce*, *Apply*, *Restrict*). Next we talk about Problog, giving the fundamental ideas of how it works and how it is implemented.

4.1 Logic

The objective of propositional logic is to model *human thinking*. Starting from declarative phrases (propositions), which can be *true* (**T**) or *false* (**F**) we construct propositions using connectives such as *or* ($\vee$), *and* ($\wedge$), *not* ($\neg$), *if...then...* ($\rightarrow$).

Let us consider the following phrases, and an interpretation:

- Penguins are birds **T**
- Africa is a continent **T**
- $1 + 1 = 4$ **F**
- A triangle has 7 sides **F**
- $1 < 7$ **T**

From this we may deduce that:

- Penguins are birds *and* Africa is a continent **T**
as it is a conjunction of **T** propositions
- A triangle has 7 sides *or* $1 < 7$ **T**
as it is a disjunction of propositions in which one of them is **T**
- *not* $1 + 1 = 4$ **T**
as it is a negation of a **F** proposition

Connective	Symbols	Arity	Equivalent Symbols
Conjunction	$\wedge$	2	$\&$
Disjunction	$\vee$	2	$+$
Implies	$\rightarrow$	2	$\supset$
Negation	$\neg$	1	$!$

Table 4.1: Propositional logic symbols

In table 4.1 we can see some symbols of propositional logic:

Logic programming was created to help programming languages that were more readable and expressive. These programming languages borrow expressive power from mathematical logic. The most popular logical language is *Prolog* (the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel).

Semantics assign meaning to programs [85]. We have two types of semantics for logic programs:

- Declarative semantics - based on the standard model-theoretic semantics of first-order logic, it describes what we want to use logic programming for.
- Operational Semantics - is a way of describing procedurally the meaning of a program [85]. It represents a procedure to satisfy the list of objectives in the context of a given program. The output of this procedure is the *truth value* from the list with the objectives with their respective instantiation of variables. The Prolog procedure allows for the automatic return (*backtracking*) in order to examine new alternatives.

The operational semantics are based on Herbrand Model, Universe, Base and also Interpretation using an example in order to better explain the subjects.

- **Herbrand Universe**

Let $\mathbf{L}$ be a first order language. The Herbrand Universe $\mathbf{UL}$ for $\mathbf{L}$ is the set of all the basic terms that can be obtained from the constants and functions in $\mathbf{L}$. If $\mathbf{L}$ does not have any constants, we add a constant for the generation of basic terms.

ex. Let us consider the following logic program P_1 :

$$p(a).$$

$$\begin{aligned} & p(b) . \\ & q(X) : \neg p(X) . \end{aligned}$$

The first order language are program P_1 clauses. Therefore, P_1 's Herbrand Universe is $UL = \{a, b\}$.

- **Herbrand Base**

Let $\mathbf{L}$ be a first order language. Herbrand Base $\mathbf{BL}$ for $\mathbf{L}$ is the set of all the basic atoms which can be obtained from using the predicates of $\mathbf{L}$ with the basic terms of its corresponding Herbrand Universe as arguments.

ex. The Herbrand Base for our logic program P_1 is $BL = \{p(a), p(b), q(a), q(b)\}$.

- **Herbrand Model**

Let $\mathbf{L}$ be a first order language and $\mathbf{S}$ a set of closed formulas of $\mathbf{L}$. A Herbrand Model for $\mathbf{S}$ is the Herbrand Interpretation which is a model for $\mathbf{S}$.

ex. Let us consider our logic program P_1 and also S (the set formed by the program clauses). Rewriting the program we get:

$$\begin{aligned} & p(a) . \\ & q(b) . \\ & q(X) : \neg p(X) . \end{aligned}$$

The domain of Herbrand's pre-interpretation for this program is $\{a, b\}$, we do not have functions. If we consider p, q Herbrand's Interpretation predicates, this Interpretation we just built is a Herbrand Model for our Program P_1 . The reason for this is that all program clauses are *true* in this interpretation.

A logic program P is a set of clauses. If this program has a model then it will have a Herbrand Model.

- **Herbrand Interpretation**

Let $\mathbf{L}$ be a first order language. An interpretation of $\mathbf{L}$ is a Herbrand Interpretation, if the following rules are satisfied:

1. The interpretation domain is the Herbrand Universe, $\mathbf{UL}$
2. The constants in $\mathbf{L}$ are assigned to themselves in $\mathbf{UL}$
3. If f is a n -ary function in $\mathbf{L}$, then we assign f the $(\mathbf{UL}^n)$ mapping in $\mathbf{UL}$ defined by $(t_1, \dots, t_n) \rightarrow f(t_1, \dots, t_n)$

4.2 Binary Decision Diagrams

Boolean functions are an important descriptive formalism for many hardware and software systems, such as synchronous and asynchronous circuits, reactive systems and finite-state programs. Representing those systems in a computer in order to reason about them requires an efficient representation of boolean functions. A boolean function can be represented by an acyclic digraph with root that consists of decision nodes and 2 terminal nodes: 0 and 1. Each decision node is labeled by a propositional variable that has 2 children, whose edges (traced or solid) correspond to the possible values assigned to the variables (0 and 1).

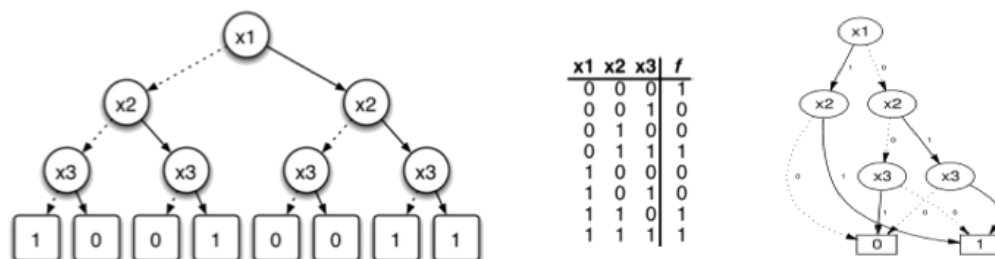


Figure 4.1: On the left a Decision tree; In the middle, a corresponding truth table; On the right a corresponding BDD

A BDD can be reduced if isomorphic subgraphs are identified and it does not have nodes whose child are isomorphic:

- *Redundant tests*: both edges of node n have the same destiny node; n can be eliminated.
- *Redundant decision nodes*: if they are roots of structural identical subBDDs; one of them can be eliminated. This guarantees that the tree has only two leaf nodes.

We may say that BDDs ensure an efficient representation thanks to the following reasons:

- *Compact Representation*: thanks to the reductions, BDDs can often be quite compact.

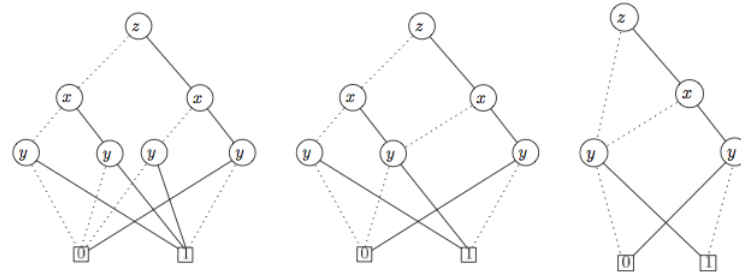


Figure 4.2: Reduction of a BDD

- Satisfiability: determine if exists a *consistent* path from the root that ends in 1. A *consistent* path is one which, for every variable, has only dashed edges or only solid edges leaving nodes labeled by that variable (we cannot assign a variable the values 0 and 1 simultaneously).
- Validity: no 0 terminal node can be reached by *consistent* paths.
- Conjunction: Given B_f and B_g representing two disjoint boolean functions f and g , a BDD representing $f \cdot g$ can be obtained by taking B_f and replacing each 1-nodes by B_g .
- Disjunction: Similar to the previous one (conjunction), but this time replacing all of B_f 0-nodes by B_g (BDD for $f + g$).
- Complementation: $B_{\bar{f}}$ is obtained from B_f by replacing the 0-terminal nodes by 1 and vice versa.

4.2.1 Ordered Binary Decision Diagrams

We have seen in the previous section that representing boolean function by using BDDs is often compact. However, BDDs that have multiple occurrences of a boolean variable along a path seem rather inefficient. Besides this, it does not seem to exist an easy way to test for equivalence of BDDs. Given these problems, one may improve the situation by imposing an ordering on the variables occurring along any path. This leads us to the ordered binary decision diagrams (OBDDs).

A BDD is *ordered* if the variables always appear in the same order along any path from the root. This induces an ordering in the set of variables. Let $[x_1, \dots, x_n]$ be an ordered list of variables, a BDD as order $[x_1, \dots, x_n]$ if for any occurrence of x_i and

x_j along a path in B we have $i < j$. A BDD is *ordered* if it has an order for its list of variables.

Theorem 1 *The reduced OBDD that represents a given function f is unique up to isomorphism.*

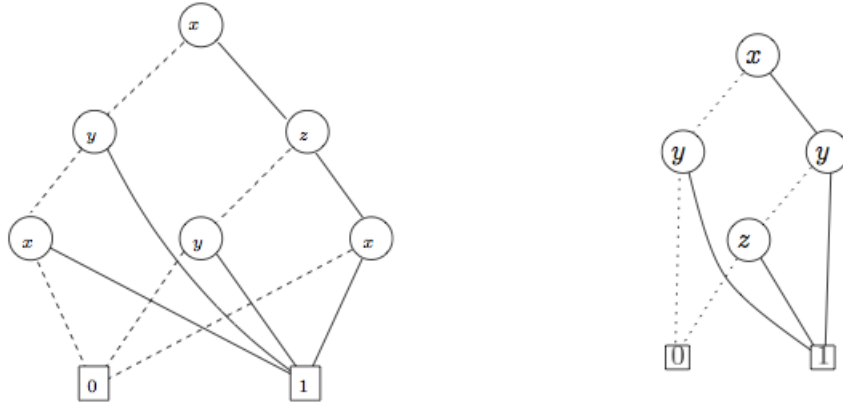


Figure 4.3: Both OBDDs are reduced, but they equivalent. Although only the left one is ordered $[x, y, z]$.

In Fig. 4.3 we can see that for the first BDD it is not possible to find an order: for the order to exist there can be no multiple occurrences of a variable along a path.

OBDDs have a *canonical form*, that is, their reduced OBDD. Most other representations do not have canonical forms. The importance of having this canonical form on OBDDs in conjunction with an efficient test that allows us to decide whether two reduced OBDDs are isomorphic should not be overestimated. The *canonical form* provides tests for:

- No redundant variables: from which function f does not depend.
- Equivalence: Allows one to test if two given functions f and g are equivalent provided that they have OBDDs with compatible orderings.
- Validity: If function f is a tautology, the reduced OBDD only has a node with value 1 (f is valid if, and only if, the reduced OBDD is B_1).
- Implication: In order to know if g is a semantic consequence of f , we must compute the reduced OBDD for $f \cdot \bar{g}$ and determine if the result is the OBDD with only one 0-node (B_0).

- **Satisfiability:** A boolean function f is satisfiable if, and only if, its reduced OBDD is different from B_0 .

Next we will have a look into three algorithms for OBDDs, more precisely the *reduce* algorithm, the *apply* algorithm and the *restrict* algorithm.

1. The *reduce* algorithm: $reduce(B_f)$

Each node n receives (bottom-up) a tag $i(n)$ so that two nodes have the same tag if, and only if, the respective sub-OBDDs represent the same boolean function:

- Assign tag $\#0$ to all the leaf nodes with value 0 and $\#1$ to all the leaf nodes with value 1.
- If $i(lo(n)) = i(hi(n))$, then $i(n)$ receives the same tag ($lo(n)$ is the node below with the dashed edge and $hi(n)$ the one with the solid edge), this leads to the possible elimination of node n , as it is redundant.
- If there exists another node m with the same variable x_i , so that $i(lo(n)) = i(lo(m))$ and $i(hi(n)) = i(hi(m))$, then $i(n) = i(m)$. this is because nodes n and m compute the same boolean function.
- If none of the two previous cases applies, we set n the next unused integer.

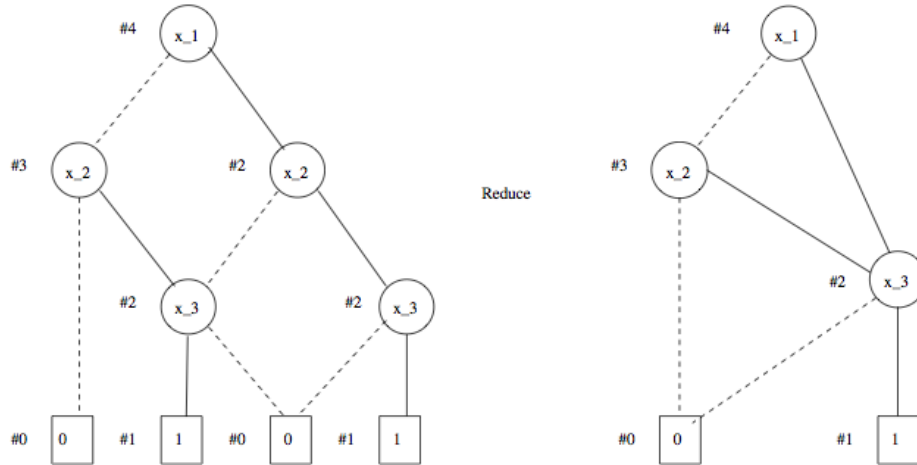


Figure 4.4: Execution of the *reduce* algorithm

2. The *apply* algorithm: $apply(op, B_f, B_g)$

Let r_f and r_g be the roots of B_f and B_g . To compute the B_{fopg} ¹ OBDD which

¹The reduced OBDD of the boolean formula B_{fopg} , where op denotes any function from $\{0,1\} \times \{0,1\}$ to $\{0,1\}$

in general is not in the reduced form, we have to apply the following steps:

- If both of them are leaves with values I_f and I_g , this implies $B_{fopg} = B0$ if $I_f op I_g = 0$, else $B_{fopg} = B1$.
- If both of them are x_i -nodes, we must create a x_i -node with a dashed edge for the $apply(op, lo(r_f), lo(r_g))$ OBDD and a solid edge for the $apply(op, hi(r_f), hi(r_g))$ OBDD.
- If r_f is a x_i -node and r_g a leaf or a x_j -node with $j > i$, we must create a x_i -node with a dashed edge for the $apply(op, lo(r_f), rg)$ OBDD, and a solid edge for the $apply(op, hi(r_f), rg)$ OBDD. The symmetric case is analogous.

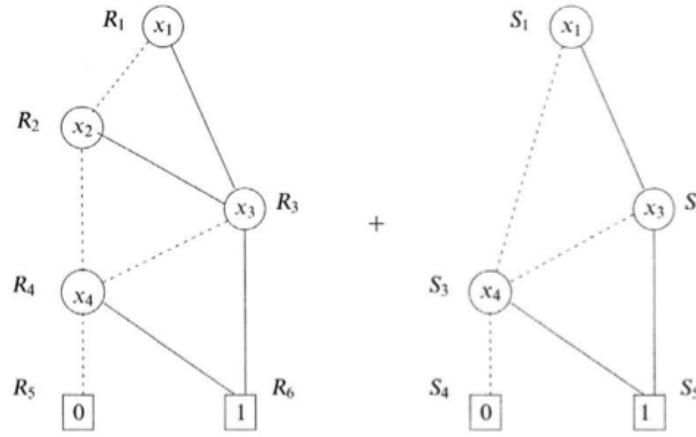


Figure 4.5: Two arguments for a call $apply(+, B_f, B_g)$.

In Fig.4.6 we can see the recursive descent control structure of $apply$ and Fig.4.7 shows the result after the call. The result of $apply(+, B_f, B_g)$ is B_f .

3. The *restrict* algorithm: $restrict(val, x, B_f)$

- To compute the $B_{f[0/x]}$ OBDD we have to redirect the edges that point to a x -node n to the $lo(n)$ node and remove node n . Then apply *Reduce*.
- To compute the $B_{f[1/x]}$ OBDD we have to redirect the edges that point to a x -node n to the $hi(n)$ node and remove node n . Then apply *Reduce*.

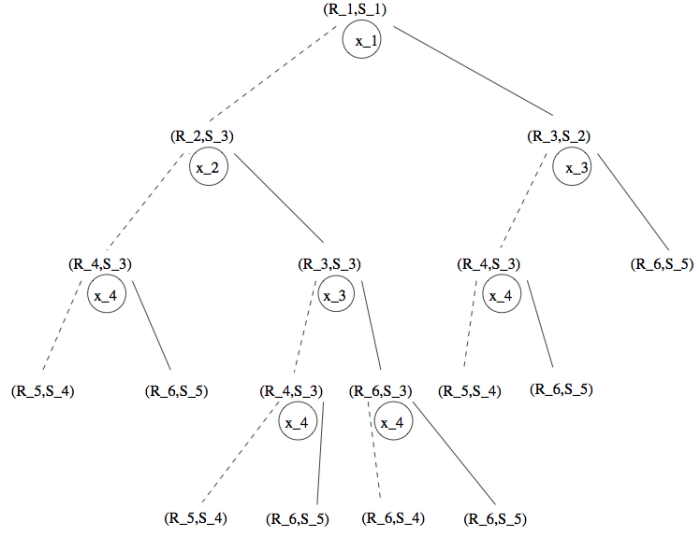


Figure 4.6: The recursive call structure for *apply* for example in Fig.4.5.

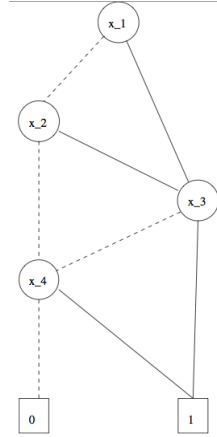


Figure 4.7: The result of $apply(+, B_f, B_g)$.

4.3 Problog

Statistical Relational Learning (SRL) [86] combines logical and probabilistic representations within the same framework. A large variety of languages and systems implement SRL concepts. Examples include PRISM [87], Probabilistic Relational Models (PRMs) [88], Stochastic Logic Programs [89], and Bayesian Logic Programs [90], [91]. A recently developed SRL language that nicely aligns with our task is Problog [6, 7]. We chose Problog because it was designed to represent a graph where there is structural uncertainty [92].

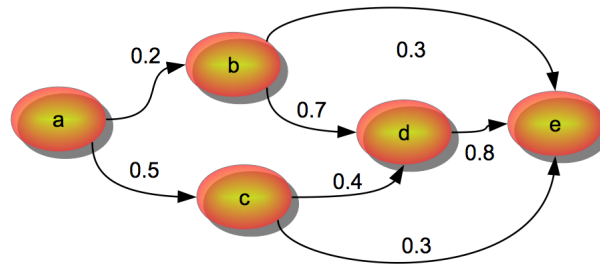


Figure 4.8: A simple directed graph, where each edge has a probability of being true.

As a straightforward example of ProbLog, consider the directed graph in Figure 4.8. Each edge has a probability of being true. ProbLog represents edges as *probabilistic facts*:

```

0.2::edge(a,b).
0.5::edge(a,c).
0.7::edge(b,c).
....

```

Notice that the probability of each edge being true is *independent* of all other edges.

ProbLog allows one to specify intensional *logic programs* that describe true relationships that hold in the program. As an example, the next two rules define a path relation between any two nodes:

```

path(N,N).
path(N,E) :-
    edge(N,M),
    path(M,E).

```

The first rule says that there is a path between node N and E if $N = E$. The second rule defines path recursively: there is a path from N to E , if from N one can reach M and from M there is a path to E .

Given a logic program and the set of probabilistic facts, a ProbLog engine allows one to reason about the probability of a relation, which is just the joint probability of all proofs of the relation. More precisely:

- The probability of a proof is the product of the probabilities of the facts appearing in the proof.

- The probability of a relation being true is given by the probability of the union of all proofs.

Imagine we want to find out $Pr(\text{path}(\mathbf{a}, \mathbf{d}))$ in the example above. One first proof is obtained by using the second clause, and calling $\text{edge}(\mathbf{a}, \mathbf{b})$ and $\text{path}(\mathbf{b}, \mathbf{d})$. The first call is a probabilistic fact. We assume it succeeds. The second call generates $\text{edge}(\mathbf{b}, \mathbf{d})$, another probabilistic fact, and $\text{path}(\mathbf{d}, \mathbf{d})$, that succeeds with its first clause. The proof is true if the probabilistic facts are true. Hence, the probability for this proof is the product of the probabilities of its (independent) probability facts: 0.2×0.7 . The second proof is obtained by considering the facts $\text{edge}(\mathbf{a}, \mathbf{c})$ and $\text{edge}(\mathbf{c}, \mathbf{b})$, and has probability 0.5×0.2 . The two proofs are disjoint, hence $Pr(\text{path}(\mathbf{a}, \mathbf{d})) = 0.2 \times 0.7 + 0.5 \times 0.2 = 0.24$.

Computing the total probability is more interesting if different paths have a common edge. As an example, consider $Pr(\text{path}(\mathbf{a}, \mathbf{e}))$. There are three proofs, that we write concisely as paths $\mathbf{abde}$, $\mathbf{acde}$, and $\mathbf{abe}$. Notice that the first proof, $\mathbf{abde}$, shares the edge $\mathbf{de}$ with the second proof, $\mathbf{acde}$, and the edge $\mathbf{ab}$ with $\mathbf{abe}$. Summing the probabilities of the three paths would count these two edges twice.

Kimmig and de Raedt showed that this reduces to the sum-product problem and proposed an effective solution. The idea is that probability can be computed as a sum if the paths do not share edges. This can be obtained by selecting an edge (or fact), and splitting into the case where the edge is true and the case where the edge is false. The process can be repeated recursively until we run out of facts to split.

This idea is the same one used to construct binary decision diagrams (BDDs) as shown before. Fig 4.9 shows a BDD that computes the total probability for the path $\mathbf{ae}$. The total probability is obtained by adding the two cases whether edge $\mathbf{ab}$ is true or not. The two cases are clearly disjoint, hence:

$$Pr(\beta_{\mathbf{ae}}) = Pr(\mathbf{ab}) \times Pr_l + Pr(\overline{\mathbf{ab}}) \times Pr_r$$

where Pr_l is the total probability for the left child $\beta_{\widehat{\mathbf{be}}}$, the BDD rooted at $\mathbf{be}$, and Pr_r is the total probability for the right child $\beta_{\widehat{\mathbf{ac}}}$, the BDD rooted at $\mathbf{ac}$. Notice that this gives rise to a recursive program but, fortunately, the program can be computed in linear time by using dynamic programming and proceeding bottom-up from the nodes 1 and 0:

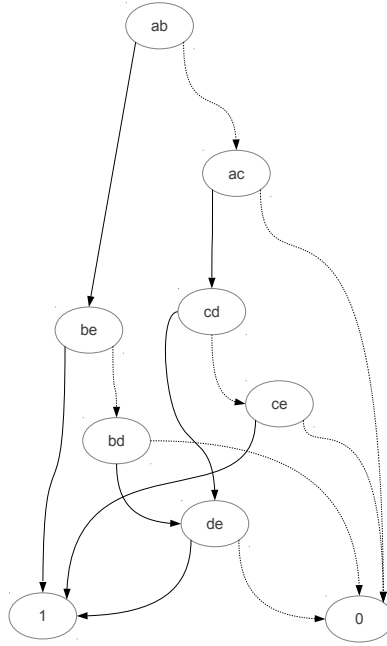


Figure 4.9: A BDD that computes the total probability of the path **ae**.

$$\begin{aligned}
 Pr(\beta_{\widehat{de}}) &= 0.8 \times 1 + 0.2 \times 0 \\
 Pr(\beta_{\widehat{bd}}) &= 0.7 \times Pr(\beta_{\widehat{de}}) + 0.3 \times 0 \\
 Pr(\beta_{\widehat{be}}) &= 0.3 \times 1 + 0.7 \times Pr(\beta_{\widehat{bd}}) \\
 Pr(\beta_{\widehat{ce}}) &= 0.3 \times 1 + 0.7 \times 0 \\
 Pr(\beta_{\widehat{cd}}) &= 0.4 \times Pr(\beta_{\widehat{de}}) + 0.6 \times Pr(\beta_{\widehat{ce}}) \\
 Pr(\beta_{\widehat{ac}}) &= 0.5 \times Pr(\beta_{\widehat{cd}}) + 0.5 \times Pr(\beta_{\widehat{bd}}) \\
 Pr(\beta_{\widehat{ab}}) &= 0.2 \times Pr(\beta_{\widehat{be}}) + 0.8 \times Pr(\beta_{\widehat{ac}})
 \end{aligned}$$

As expected, one can observe that the sub-tree $\beta_{\widehat{de}}$ is shared by the path using node **b** and node **c**. The BDD allows us to count this edge only once.

Binary decision diagrams provide a very efficient implementation for probability computation over small and medium graphs. Unfortunately, they do not scale to larger graphs with thousands of nodes. In this case, ProbLog implementations rely on approximated solutions, either Monte Carlo methods or often by approximating the total probability by the probability of the best k proofs [7].

4.3.1 Learning Problog Programs

Arguably, the most fundamental task in learning Problog programs is *parameter learning*, which aims at obtaining the best values for the fact probabilities given a program structure and examples. There are a number of approaches to this problem. LeProblog [8] provides a very natural and general algorithm, that is well suited for our task. The goal in parameter learning is to find the best set of probability parameters, Θ , given a set of examples E :

$$\max_{\arg} Pr(\Theta|E)$$

In our setting, we do not have a prior on the parameters and hence we can assume that all parameters have uniform probability. Applying Bayes theorem, we obtain:

$$Pr(\Theta|E) \propto Pr(E|\Theta)$$

LeProblog maximizes $Pr(E|\Theta)$ by using gradient ascent. To compute the gradient of a probability of an example $e \in E$ on a parameter $\theta_i \in \Theta$ we can simply differentiate the expression for total probability:

$$Pr = \theta_j \times Pr_l + (1 - \theta_j) \times Pr_r$$

If $i \neq j$, then the gradient $\frac{\delta Pr}{\delta \theta_j}$ can be obtained through the chain rule:

$$\frac{\delta Pr}{\delta \theta_j} = \theta_j \times \frac{\delta Pr_l}{\delta \theta_j} + (1 - \theta_j) \times \frac{\delta Pr_r}{\delta \theta_j}$$

If $i = j$, the expression is slightly more complex:

$$\frac{\delta Pr}{\delta \theta_j} = Pr_l + \theta_j \times \frac{\delta Pr_l}{\delta \theta_j} + (1 - \theta_j) \times \frac{\delta Pr_r}{\delta \theta_j} - Pr_r$$

We refer the reader to [8] for a complete discussion and detailed derivation assuming the parameters follow a sigmoidal function. In practice, this result means that to compute the gradient over some parameter θ_i we simply have to follow a bottom-up dynamic programming algorithm, in a similar style as what we had to do to compute the probability.

It should be noted that this result is obtained when we have the complete BDD. Often, we use k -proofs, where BDD is only a lower approximation to the correct probability, so we may not converge to local maximum. To address this problem, LeProbLog recomputes the best proofs every k steps during the gradient ascent process.

Chapter 5

Implemented Work

*If we knew what it was we were
doing, it would not be called
research, would it?*

Albert Einstein

5.1 Representing gene networks with Problog

Problog requires both logical and probabilistic inferencing. Logical inferencing is implemented though the YAP Prolog engine [93]; probabilistic inferencing relies on the CUDD BDD library for constructing the BDDs [94], and on the SimpleCUDD program to compute probabilities and derivatives. BDDs can grow very quickly and can be quite expensive to create, thus the Problog implementers decided to use text files to store all BDDs.

In practice, we have noticed that accessing BDDs as files and creating a new instance of SimpleCUDD for processing the derivatives associated with each example is expensive and severely limits performance, making processing of large problems almost impossible. On the other hand, in practice quite often we work with limited-size BDDs.

We address this problem by manipulating the BDDs as Prolog terms. A CUDD BDD is translated as a list of Prolog terms, where each term in the list is a node in the original BDD, and is as follows:

`node(Theta, Left, Right, Value)`

The `Left`, `Right`, and `Value` logical variables represent, respectively, the node's value, the value of the left child, and the value of the right child.

Figure 5.1 shows a three-node fragment of our graph's BDD. This BDD can be encoded as the following Prolog term:

`[node(THETA_BE,PBC,1,PBD), node(THETA_BD,PBD,PDE,0), node(THETA_DE,PDE,1,0)]`

By using this representation, we can implement probability computation and gradient computation in Prolog as a walk over this list of nodes (ordered bottom-up). More precisely, to compute probabilities, we run the following Prolog code at each node:

Value is $\theta \times \text{Left} + (1 - \theta) \times \text{Right}$

Notice that we do not need to explicitly propagate value upwards. Instead, Prolog unification does the propagation implicitly.

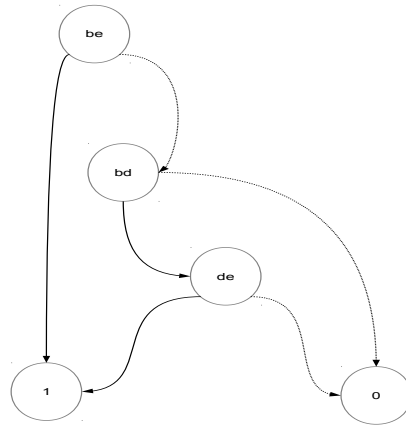


Figure 5.1: A simple BDD

Although Prolog code is much slower at performing arithmetic than the SimpleCUDD's C code, the lower interface cost and avoiding the need to create a process per new BDD makes the Prolog based approach a much faster solution.

5.2 Experimental Methodology

A cell responds to environmental changes by detecting molecules that bind to receptors on the surface of the cell and transmits this information to proteins within the cell by activating a cascade of molecular events. These signaling networks are typically studied by measuring molecular events after treatments that stimulate or perturb key elements in the network at the mRNA or gene level. Combining these measurements with phenotypic response enables the study of these treatments on the architecture and function of the underlying signaling networks as well as the relationship between the network behavior and phenotypic response [1]. To infer the architecture and function of the underlying signaling network of budding yeast, we decided to focus on the pathways activated by MAPK *Hog1* during osmotic stress response. According to previous results [2] this pathway interacts with the general stress (*Msn2/Msn4*) pathways, so we consider the genes that belong to both pathways our study.

We obtained time-series gene expression data from [95] for our analysis. The experiments followed the response of actively growing *Saccharomyces cerevisiae* subjected to an osmotic shock of 0.7 M NaCl. The dose of salt was chosen by the biologists to provide a robust physiological response but also allow high viability and eventual resumption of cell growth. Three biological replicate samples were collected before and

after NaCl treatment at 30, 60, 90, 120, and 240 min (measuring the peak transcript changes that occurs at or after 30 min) [96]. We focused our attention on the 270 genes of the **Hog1 Msn2/4** pathway from [2] for which we have expression data. The reason we limit our focus to the genes of the **Hog1 Msn2/4** pathway is due to the fact that this dataset, unlike that from [2] is much more limited. Furthermore, our main goal is to test the ability of ProbLog to learn pathways based on pairwise correlations or relationships that are computed from gene expression data.

We performed three types of analysis to estimate values for the edges, which are then used as training examples in ProbLog:

1. We first compared the three time-series to determine data quality and better understand the general patterns in the data.
2. Computed correlations to determine and quantify the main relationships in the pathways (correlations were mapped from *-1 to 1* into *0 to 1*).
3. Utilized temporal data to determine temporal relationships in the data (we used a sigmoid function to convert activity levels into *0 to 1*).

In the first and second analyses we computed correlations between normalized gene expression values. To do so, we use the *Pearson product-moment correlation coefficient*, also known as r :

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (5.1)$$

We first compute correlations between genes in the same experiment, where X and Y range over the temporal data in order to calculate the correlation (r) between each pair of genes. We experimented with both correlating X_t and Y_t (same time-step) and correlating X_t and Y_{t+1} (next time-step).

In our first analysis, we further compared the r -coefficient obtained between all correlations in the three biological replicates. Although all three replicates followed the same methodology, variations in initial conditions can significantly affect gene expression. We study this effect in order to understand the inherent variability existing in the data.

In our second analysis, we assume that if a pair of genes has high absolute correlation (a strong positive or negative linear dependence between the two variables) it also has a high probability of being connected in the pathway [97], resulting in two possible connected nodes in the gene interaction network. Correlation of genes based on expression

data may give insight to gene interactions or suggest pathways and associations. This information is then used as training examples for the ProbLog learner. The ProbLog background knowledge is of the form:

```

promoter(X,Y,S) :-
    hog1_promoter(X),
    hog1_promoter(Y),
    edge(X,Y,S).
promoter(X,Y,S) :-
    hog1_promoter(X),
    promoter(X,W,_),
    edge(W,Y,S).

```

The *Hog1_Promoter* genes are *Hog1*, *Msn2*, *Msn4*, *Hot1*, and *Sko1*. The *S* flag indicates whether the edge corresponds to activation or inhibition. The **Connected** relationship is defined either as:

```

edge(X,Y,S) :- connected(Y,X,S).
edge(X,Y,S) :- connected(X,Y,S).

```

in the case of learning an undirected graph (same time-step correlations), and as:

```

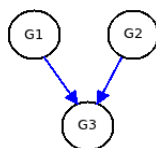
edge(X,Y,S) :- connected(Y,X,S).

```

for a directed graph (different time steps).

The third analysis aims for a more detailed picture of the learned network by using the temporal nature of the data. The output generated is a weighted, directed gene network, but nodes are connected as a gated network:

- **AND:** two promoter genes need to be active in order to activate a gene, as shown in the graph. We also show the ProbLog code for the temporal model:

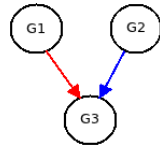


```

active(G3,T1,Z) :-
    next_step(T0,T1),
    and(G1,G2,G3),
    active(E,T0,G1),
    active(E,T0,G2).

```

- **OR:** either promoter gene needs to be active in order to activate a gene, as shown in the graph. We also show the corresponding ProbLog code for the temporal model.



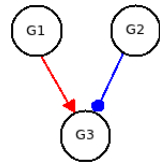
```

active(G3,T1,Z) :-
    next_step(T0,T1),
    or(G1,G2,G3),
    active(E,T0,G1).

active(G3,T1,Z) :-
    next_step(T0,T1),
    or(G1,G2,G3),
    active(E,T0,G2).

```

- **NEG:** one promoter gene needs to be active and one repressor gene needs to be inactive in order to activate a gene, as shown in the graph.



```

active(G3,T1,Z) :-
    next_step(T0,T1),
    neg(G1,G2,G3),
    active(E,T0,G1),
    not_active(E,T0,G2).

```

This is the only case where we allow the possibility of negative regulation.

- **SINGLE:** a unique promoter gene regulates the target gene.



```

active(G2,T1,Z) :-
    next_step(T0,T1),
    single(G1,G2),
    active(E,T0,G1).

```

We use two different forms of temporal data: expression level (E), and variation (Δ). We experimented with three different approaches:

- Level influences variation (LV).

$$\begin{aligned}
 E_t(G1) \wedge E_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 E_t(G1) \vee E_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 E_t(G1) \ominus E_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 E_t(G1) &\Rightarrow \Delta_{t+1}(G)
 \end{aligned}$$

- Variation influences variation (VV).

$$\begin{aligned}
 \Delta_t(G1) \wedge \Delta_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 \Delta_t(G1) \vee \Delta_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 \Delta_t(G1) \ominus \Delta_t(G2) &\Rightarrow \Delta_{t+1}(G) \\
 \Delta_t(G1) &\Rightarrow \Delta_{t+1}(G)
 \end{aligned}$$

- Level influences level (LL).

$$\begin{aligned}
E_t(G1) \wedge E_t(G2) &\Rightarrow E_{t+1}(G) \\
E_t(G1) \vee E_t(G2) &\Rightarrow E_{t+1}(G) \\
E_t(G1) \ominus E_t(G2) &\Rightarrow E_{t+1}(G) \\
E_t(G1) &\Rightarrow E_{t+1}(G)
\end{aligned}$$

One important advantage of our approach is that it allows us to implement *soft constraints* on the probability distribution. In other words, we can specify that satisfying some constraint has a very low probability. In the third analysis, we implemented constraints that a *gene activity must be explained by a single rule*. Two example constraints for **OR** are of the form: The next constraint says that there must be a single set of parents for a gene defined with the LV $\vee$ rule:

$$\begin{aligned}
&E_t(G_1) \vee E_t(G_2) \Rightarrow \Delta_{t+1}(G) \\
&\wedge \\
&E_t(G_3) \vee E_t(G_4) \Rightarrow \Delta_{t+1}(G) \\
\rightarrow \\
&G_1 = G_3 \wedge G_2 = G_4
\end{aligned}$$

The second constraint ensures that we cannot use two rules of different types at the same time:

$$\begin{aligned}
&\neg(\quad E_t(G_1) \vee E_t(G_2) \Rightarrow \Delta_{t+1}(G) \\
&\quad \wedge \\
&\quad E_t(G_3) \ominus E_t(G_4) \Rightarrow \Delta_{t+1}(G) \\
&\quad)
\end{aligned}$$

In practice, we must be careful not to flood the system with soft constraints. In this experiment we implemented one joint soft constraint per gene.

5.2.1 Implementation

Figure 5.2 shows the pipeline used in analysis two and three. We fed the expression data to a domain pre-processor that generates the E and Δ data, and computes correlations. We then use the ProbLog language, as implemented using YAP Prolog [93]

and CUDD [94] to generate an output graph and some statistics. The output graph is translated to graphviz format for visual inspection.

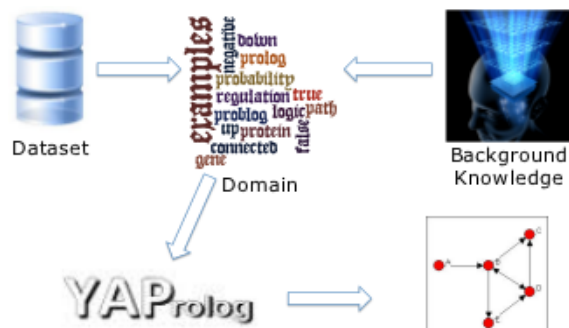


Figure 5.2: Implementation Method

5.3 Results

Our first analysis looked at data variation between the three experiments. To do so, we simply compute the correlation between gene expression level between every pair of experiments as the experiment proceeds across time. We compute this expression between all genes, and only between the genes in the Hog1/Msn2/4 network.

Time	Experiment Pair					
	All Genes			Hog1 Genes		
	1 – 2	1 – 3	2 – 3	1 – 2	1 – 3	2 – 3
30	0.84	0.88	0.91	0.83	0.89	0.90
60	0.64	0.77	0.80	0.84	0.88	0.89
90	0.71	0.79	0.79	0.87	0.88	0.87
120	0.71	0.75	0.77	0.84	0.84	0.83
240	0.52	0.65	0.53	0.80	0.80	0.73

Table 5.1: Variation in Correlation Between All Genes and Experiments Along Time

The experiments initially show a strong correlation, close to 90% after 30 min. The correlation decreases to 50% after 240 min. The decrease is noticeable in every experiment: no experiment seems different from the others. Of the set of genes in the Hog1 pathway, the genes remain correlated up until 240 min.

As a second step we analyze the main correlation between genes and their direct regulators by using the correlations between the genes as examples. Figure 5.3 visualizes the connections between the five promoter genes. This network is very similar to the network presented by Capaldi et al. [2]. The network is controlled by **Hog1** with **Msn2/4** as a separate connected network and the **Hot1/Sko1** promoters interacting with **Hog1**.

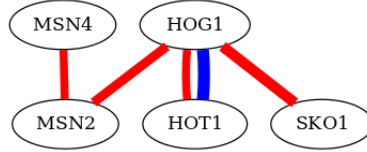


Figure 5.3: Same-step Correlation Generated **Hog1** Promoter Network

We then try to determine which promoter gene regulates **Hog1** or **Msn2/4**. Table 5.2 shows promoters that are proposed as parents, with the threshold ranging from $0.7 \leq P \leq 0.9$. They also show the major cases where we have multiple parents for a node. The results suggest that **Msn2/4** and **Sko1** have the most direct impact on gene expression. Moreover, as we lower the threshold it becomes harder to distinguish the influence of individual promoters.

Gene(s)	90%	80%	70%
Msn2	7	37	61
Msn4	1	22	68
Hog1	0	14	35
Hot1	0	8	38
Sko1	1	21	50
Msn2/Msn4	0	8	31
Msn2/Sko1	0	9	28
Msn4/Sko1	0	9	22
Msn2/Hog1	0	3	9
Total	9	102	153

Table 5.2: Analysis 2: proposed parents with threshold $0.7 \leq P \leq 0.9$

The approach followed in analysis two has a few drawbacks: it is hard to validate the model, and there is a clear interference between promoters.

Analysis three addresses this problem to some extent. We reserve the last point in the time-series as test data in order to verify whether the system achieves predictive

accuracy. Figure 5.4 shows the variation on test-set mean square error (MSE) as iterations proceed. For the LV setting ($L \Rightarrow \Delta$) MSE on the test data drops by approximately 30% in the first five hundred iterations. The VV setting initially gets worse but then improves to a value similar to LV. The LL setting converges to a much lower error, but it has an easier task as most genes have high expression.

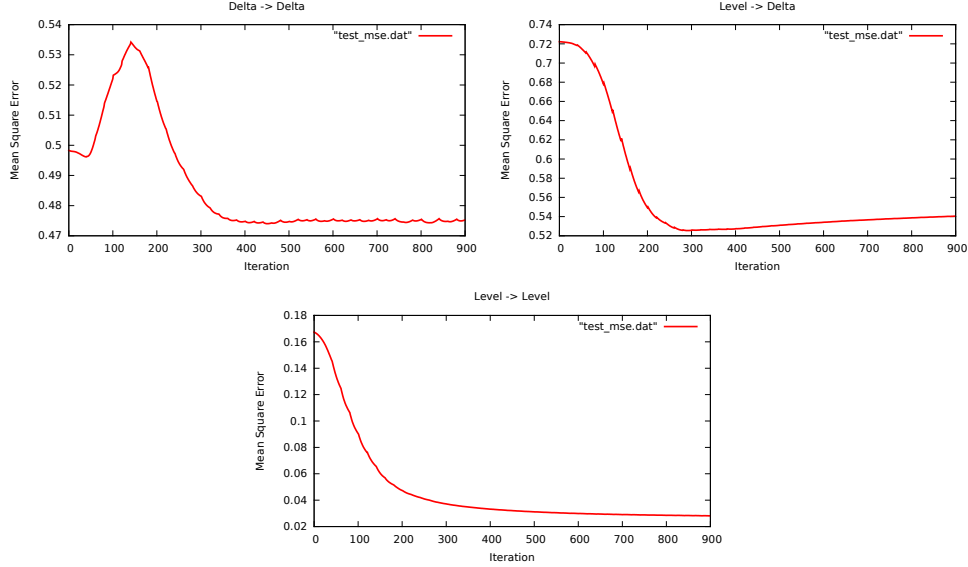


Figure 5.4: Learning Curves for Test Data Mean Square Error, resp for VV ($\Delta \Rightarrow \Delta$), VL ($\Delta \Rightarrow L$) and LL ($L \Rightarrow L$).

Next, we study performance for VV in more detail. We use the network configuration after 900 iterations, even if it does not have the best predictive performance. First, we observed much lower probabilities, than on the previous experiment, hence we use $P > 0.3$ to select candidate gates. The algorithm finds 107 genes with parents at $P > 0.3$. Of those 107, 9 have > 2 parents, indicating the algorithm has not converged yet. The distribution of gates over the remaining 98 is given in Table 5.3. Most target genes are controlled positively by **Msn2/4** or **Hot1**. It is interesting to observe repression from **Hog1/Sko1** [98].

We also study the performance for LV in more detail. We looked at the configuration when test-set accuracy was best. Again we use $P > 0.3$ to filter the candidate gates. The algorithm finds 99 genes with parents at $P > 0.3$. Of those 99, 10 have > 3 parents, indicating the algorithm has not converged yet. The distribution of gates over the remaining 89 is given in Table 5.4. Most genes are controlled positively by **Msn2** and **Msn4**, as expected, but the algorithm also indicates influence by **Hot1**.

Gene	$\wedge$	$\vee$	$\ominus+$	$\ominus-$
Msn2	22	4	2	
Msn4	16		23	
Hog1	9	7	4	18
Sko1	9	14		10
Hot1	29	10	2	4

Table 5.3: Experiment 3 (VV): proposed parents per gate. Notice that $\ominus+$ represents the positive parent, and $\ominus-$ the negative parent.

Gene	$\wedge$	$\vee$	$\ominus+$	$\ominus-$
Msn2	23	3	55	1
Msn4	30	3	52	
Hog1	7	1	1	
Hot1	1	3		107
Sko1		2		1

Table 5.4: Experiment 3 (LV): proposed parents per gate. Notice that $\ominus+$ represents the positive parent, and $\ominus-$ the negative parent.

5.3.1 Problog Performance

As we discussed before in section 5.1, there was a need to improve Problog code in order to face the performance problem created by the BDDs access and creation of a SimpleCUDD instance. After implementing these changes, we experimentally evaluated¹ how the new method performed. The following tables and graphs show running times of two versions of Problog (original and the new)- the domain (data) is equal for every experiment (the same data and examples), the results are in the following form - *hours : minutes : seconds* or *minutes : seconds*.

- **No BDDs rebuild**

Just for looking into Table 5.5 and the graph in Fig. 5.5 one can immediately conclude that the execution times using the two versions are completely different, we compressed the calculation times significantly, as we take from hours to minutes with the new

¹All of these tests were executed on a machine with an *Intel* Core i7 CPU 920 2.67GHz with 12 gigabytes of RAM, running Ubuntu (Release 11.04 - Kernel 2.6.38-16-generic).

Iterations	Original Version			New Version		
	<i>VV</i>	<i>LV</i>	<i>LL</i>	<i>VV</i>	<i>LV</i>	<i>LL</i>
50	1:14:48	1:07:50	1:20:02	1:00	1:07	1:36
100	2:26:54	1:58:05	2:47:07	1:53	2:06	2:52
300	8:35:55	6:41:32	9:30:39	5:25	5:55	8:12
400	12:47:46	9:56:08	13:54:20	7:13	7:53	10:56

Table 5.5: Calculation times differences between the two ProbLog versions with no BDDs rebuild

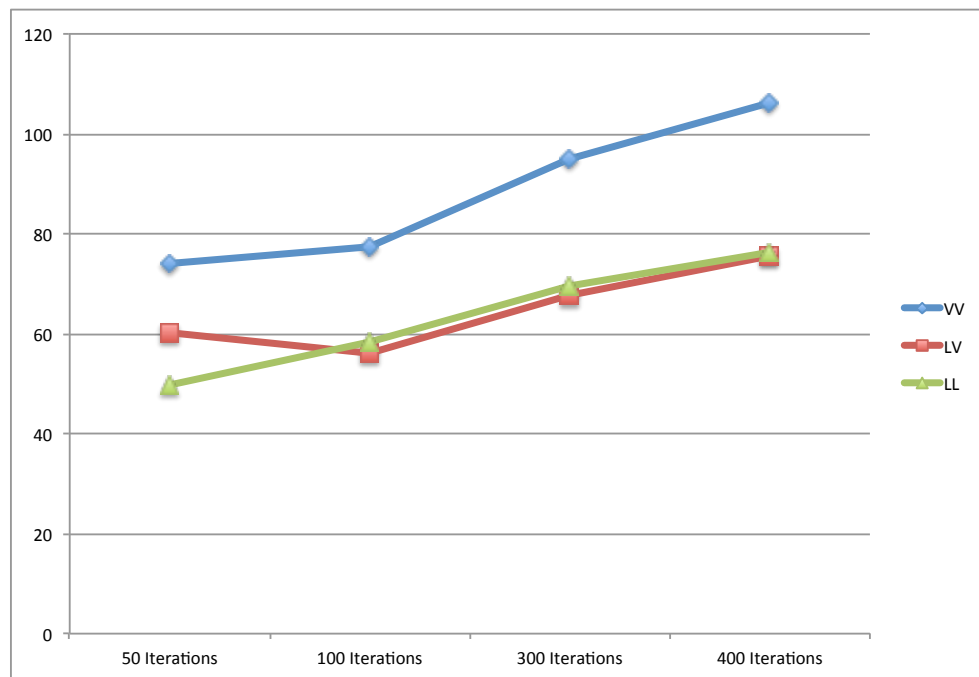


Figure 5.5: Speedup graph based on Table 5.5 results

version. Next we will see the comparison when rebuilding the BDDs after 20 or 45 iterations.

- **With BDDs rebuild**

From these results we can also see that even with the extra time needed for the BDDs rebuild, we were able to compress the calculations time significantly, again, we passed from several hours to minutes. This calculation time compressing in both situations (with or without rebuild) allows us to run more complex experiments.

Iterations	Original Version			New Version		
	<i>VV</i>	<i>LV</i>	<i>LL</i>	<i>VV</i>	<i>LV</i>	<i>LL</i>
50	1:26:42	1:06:23	1:38:24	1:20	1:26	2:08
100	2:47:49	2:23:11	3:19:33	2:27	2:41	3:54
300	10:32:23	8:07:54	12:23:01	7:22	7:59	11:45
400	15:16:36	11:13:17	16:56:45	9:56	10:47	15:42

Table 5.6: Calculation times differences between the two ProbLog versions with BDDs rebuild after 20 iterations

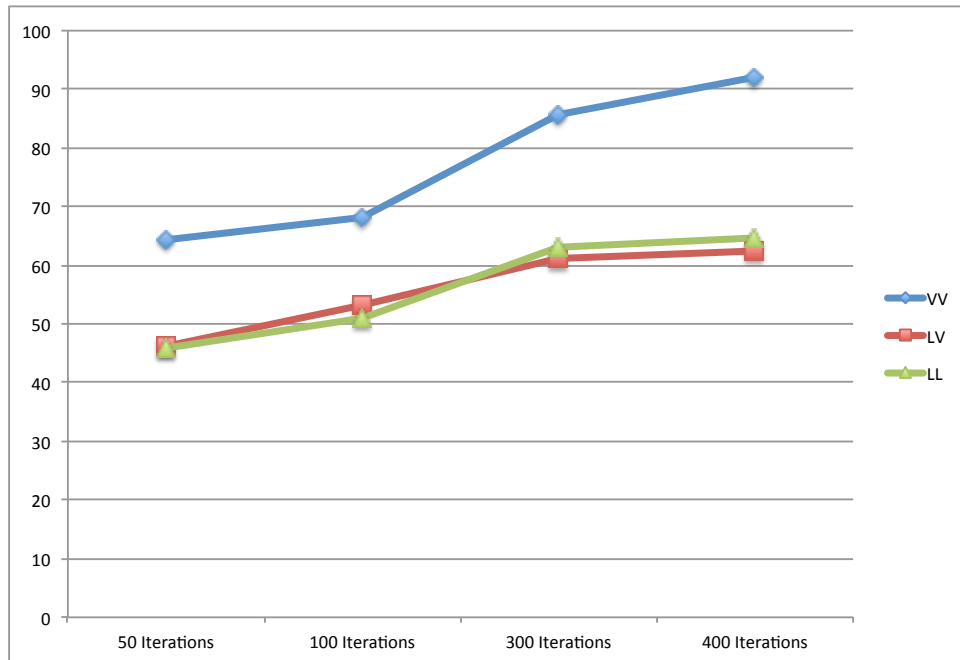


Figure 5.6: Speedup graph based on Table 5.6 results

Iterations	Original Version			New Version		
	<i>VV</i>	<i>LV</i>	<i>LL</i>	<i>VV</i>	<i>LV</i>	<i>LL</i>
50	1:21:42	1:03:40	1:27:37	1:14	1:19	1:57
100	2:55:03	2:19:30	3:14:02	2:15	2:24	3:30
300	9:57:14	7:23:38	11:05:50	6:25	6:52	9:51
400	14:11:28	10:49:00	15:04:43	8:21	9:12	13:10

Table 5.7: Calculation times differences between the two ProbLog versions with BDDs rebuild after 45 iterations

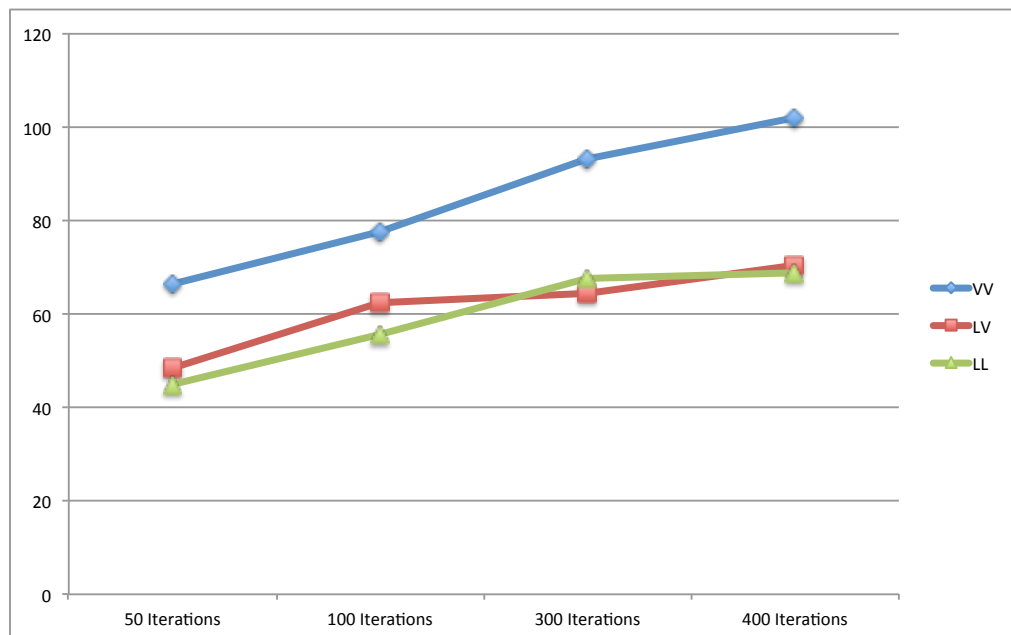


Figure 5.7: Speedup graph based on Table 5.7 results

Chapter 6

Conclusions

*The outcome of any serious research
can only be to make two questions
grow where only one grew before.*

Thorstein Veblen

6.1 Conclusions

Learning regulatory networks from gene expression is a hard problem. Data is noisy, there are often hidden/unmeasured data and relationships between genes are highly complex. We present a statistical relational approach to modeling pathways. Our approach allows us to design a coarser and more fine grained model, based on probabilistic gates. We show that the latter model has predictive performance on the time-series data, and recovers important relationships despite having limited data. We were able to recover some important relationships despite the lack of additional data such as knockout, ChIP-chip.

6.2 Future Work

We plan to continue improving the model quality and experiment with new data. Specifically, we would like to experiment with implementing a regression based approach, as it naturally fits our framework. In addition, we would like to experiment with different pathways and with proteomic data. Last, but not least, we would like to investigate how to reduce the number of parameters in the model by exploiting strong correlations between gene expression. Some further goals of this work will include:

- Expand our models to also learn from RNASeq data, and to compare RNASeq and microarray data.
- Generalize the model to include data from ncRNA expression levels, and to support protein data
- Combine probabilistic / logic models with differential equations based models.
- Include protein-to-protein interactions, promoters, phosphorylation data in our models.
- Proteomics data: how to include it in the model?

High-throughput sequencing is known to be an effective approach for transcriptome analysis. This methodology called RNA-seq [99], has been used to analyze unknown

transcript sequences, estimate gene expression levels and study single nucleotide polymorphisms. Recent RNA-Seq experiments [19] have shown great potential for transcriptome profiling. It is well known that sequencing increases the level of biological detail and also that integrative data analysis is also useful. These are some of the reasons arguing for including RNA-Seq data on the proposed work.

Non-coding RNA (ncRNA) is the next 'frontier' for the biological community, this has not been crossed yet, and it continues to present a lot of questions and doubts [11], [12], [13]. If the model is generalized to include data also from ncRNA, it would be a great step forward, and could lead to the discovery of connections between ncRNA's and other genes, and also the discovery of what they regulate or even co-regulate, which would be a significant contribution to the scientific community regarding the common comprehension and also advance on disease treatments.

Proteomic data is a potentially rich, but maybe under-exploited, data source for genome annotation. Peptide identifications from tandem mass spectrometry provide *prima facie* evidence for gene predictions and can also discriminate over a set of candidate gene models, making this type of data a good candidate for this work model.

Gene regulatory networks (GRN) models are difficult to deduce just by using experimental techniques, so, computational and mathematical methods are indispensable. As GRNs are nonlinear, nonlinear differential equation models can model much more complex GRN behavior. The identification of the nonlinear differential equation in these type of models is computationally more intensive and may require more data. Despite this, the range of nonlinear behaviors exhibited by GRNs can be comprehensively realized with nonlinear differential equations.

Appendix A

Code

```
#####
# Delta as a function of 2 other Delta      #
# Delta(A,t) AND Delta(B,t) -> Delta(C,t+1)  #
# Delta(A,t) OR Delta(B,t) -> Delta(C,t+1)   #
# Delta(A,t) -> Delta(C,t+1)                 #
# Delta(A,t) OR_NOT Delta(B,t) -> Delta(C,t+1) #
#####

next(30,60).
next(60,90).
next(90,120).
next(120,240).

delta_ge(E,T1,Z) :-
    and(X,Y,Z),
    \+ or(_,_,Z),
    next(T0,T1),
    delta(E,T0,X),
    delta(E,T0,Y).
delta_ge(E,T1,Z) :-
    or(X,_Y,Z),
    next(T0,T1),
    delta(E,T0,X).
delta_ge(E,T1,Z) :-
```

```

    or(_X,Y,Z),
    next(T0,T1),
    delta(E,T0,Y).
delta_ge(E,T1,Z) :-
    or_not(X,_Y,Z),
    next(T0,T1),
    delta(E,T0,X).
delta_ge(E,T1,Z) :-
    or_not(_X,Y,Z),
    next(T0,T1),
    not_delta(E,T0,Y).
delta_ge(E,T1,Z) :-
    single(X,Z),
    next(T0,T1),
    delta(E,T0,X).

delta_not_ge(E,T1,Z) :-
    and(X,_Y,Z),
    next(T0,T1),
    not_delta(E,T0,X).

delta_not_ge(E,T1,Z) :-
    and(_X,Y,Z),
    next(T0,T1),
    not_delta(E,T0,Y).

delta_not_ge(E,T1,Z) :-
    or(X,Y,Z),
    next(T0,T1),
    not_delta(E,T0,X),
    not_delta(E,T0,Y).

delta_not_ge(E,T1,Z) :-
    or_not(X,Y,Z),
    next(T0,T1),
    not_delta(E,T0,X),
    delta(E,T0,Y).

```

```
delta_not_ge(E,T1,Z) :-
    single(X,Z),
    next(T0,T1),
    not_delta(E,T0,X).

fal_and(Z) :- and(X,Y,Z), and(X1,Y1,Z), ( X1 \= X ; Y1 \= Y ).
fal_and(Z) :- and(_,_,Z), or(_,_,Z).
fal_and(Z) :- and(_,_,Z), or_not(_,_,Z).
fal_and(Z) :- and(_,_,Z), single(_,Z).
fal_or(Z) :- or(X,Y,Z), or(X1,Y1,Z), ( X1 \= X ; Y1 \= Y ).
fal_or(Z) :- or(_,_,Z), and(_,_,Z).
fal_or(Z) :- or(_,_,Z), or_not(_,_,Z).
fal_or(Z) :- or(_,_,Z), single(_,Z).
fal_or_not(Z) :- or_not(X,Y,Z), or_not(X1,Y1,Z), ( X1 \= X ; Y1 \= Y ).
fal_or_not(Z) :- or_not(_,_,Z), and(_,_,Z).
fal_or_not(Z) :- or_not(_,_,Z), or(_,_,Z).
fal_or_not(Z) :- or_not(_,_,Z), single(_,Z).
fal_single(Z) :- single(X1,Z), single(X,Z), X \= X1.
fal_single(Z) :- single(_,Z), and(_,_,Z).
fal_single(Z) :- single(_,Z), or(_,_,Z).
fal_single(Z) :- single(_,Z), or_not(_,_,Z).
```

```

#####
# Level as a function of 2 other Level      #
# Level(A,t) AND Level(B,t) -> Level(C,t+1)  #
# Level(A,t) OR Level(B,t) -> Level(C,t+1)   #
# Level(A,t) -> Level(C,t+1)                 #
# Level(A,t) OR_NOT Level(B,t) -> Level(C,t+1) #
#####

next(30,60).
next(60,90).
next(90,120).
next(120,240).

ge_ex(E,T1,Z) :-
    and(X,Y,Z),
    \+ or(_,_,Z),
    next(T0,T1),
    ge(E,T0,X),
    ge(E,T0,Y).
ge_ex(E,T1,Z) :-
    or(X,_Y,Z),
    next(T0,T1),
    ge(E,T0,X).
ge_ex(E,T1,Z) :-
    or(_X,Y,Z),
    next(T0,T1),
    ge(E,T0,Y).
ge_ex(E,T1,Z) :-
    or_not(X,_Y,Z),
    next(T0,T1),
    ge(E,T0,X).
ge_ex(E,T1,Z) :-
    or_not(_X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,Y).
ge_ex(E,T1,Z) :-
    single(X,Z),

```

```
    next(T0,T1),
    ge(E,T0,X).

not_ge_ex(E,T1,Z) :-
    and(X,_Y,Z),
    next(T0,T1),
    not_ge(E,T0,X).

not_ge_ex(E,T1,Z) :-
    and(_X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,Y).

not_ge_ex(E,T1,Z) :-
    or(X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,X),
    not_ge(E,T0,Y).

not_ge_ex(E,T1,Z) :-
    or_not(X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,X),
    ge(E,T0,Y).

not_ge_ex(E,T1,Z) :-
    single(X,Z),
    next(T0,T1),
    not_ge(E,T0,X).

fal_and(Z) :- and(X,Y,Z), and(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).
fal_and(Z) :- and(_,_Z), or(_,_Z).
fal_and(Z) :- and(_,_Z), or_not(_,_Z).
fal_and(Z) :- and(_,_Z), single(_Z).
fal_or(Z) :- or(X,Y,Z), or(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).
fal_or(Z) :- or(_,_Z), and(_,_Z).
fal_or(Z) :- or(_,_Z), or_not(_,_Z).
```

```
fal_or(Z) :- or(_,_,Z), single(_,Z).  
fal_or_not(Z) :- or_not(X,Y,Z), or_not(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).  
fal_or_not(Z) :- or_not(_,_,Z), and(_,_,Z).  
fal_or_not(Z) :- or_not(_,_,Z), or(_,_,Z).  
fal_or_not(Z) :- or_not(_,_,Z), single(_,Z).  
fal_single(Z) :- single(X1,Z), single(X,Z), X \= X1.  
fal_single(Z) :- single(_,Z), and(_,_,Z).  
fal_single(Z) :- single(_,Z), or(_,_,Z).  
fal_single(Z) :- single(_,Z), or_not(_,_,Z).
```

```
#####
# Delta as a function of 2 other Level      #
# Level(A,t) AND Level(B,t) -> Delta(C,t+1)  #
# Level(A,t) OR Level(B,t) -> Delta(C,t+1)   #
# Level(A,t) -> Delta(C,t+1)                 #
# Level(A,t) OR_NOT Level(B,t) -> Delta(C,t+1) #
#####

next(30,60).
next(60,90).
next(90,120).
next(120,240).

delta_ge(E,T1,Z) :-
    and(X,Y,Z),
    \+ or(_,_,Z),
    next(T0,T1),
    ge(E,T0,X),
    ge(E,T0,Y).
delta_ge(E,T1,Z) :-
    or(X,_Y,Z),
    next(T0,T1),
    ge(E,T0,X).
delta_ge(E,T1,Z) :-
    or(_X,Y,Z),
    next(T0,T1),
    ge(E,T0,Y).
delta_ge(E,T1,Z) :-
    or_not(X,_Y,Z),
    next(T0,T1),
    ge(E,T0,X).
delta_ge(E,T1,Z) :-
    or_not(_X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,Y).
delta_ge(E,T1,Z) :-
    single(X,Z),
```

```

next(T0,T1),
ge(E,T0,X).

```

```

delta_not_ge(E,T1,Z) :-
    and(X,_Y,Z),
    next(T0,T1),
    not_ge(E,T0,X).

```

```

delta_not_ge(E,T1,Z) :-
    and(_X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,Y).

```

```

delta_not_ge(E,T1,Z) :-
    or(X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,X),
    not_ge(E,T0,Y).

```

```

delta_not_ge(E,T1,Z) :-
    or_not(X,Y,Z),
    next(T0,T1),
    not_ge(E,T0,X),
    ge(E,T0,Y).

```

```

delta_not_ge(E,T1,Z) :-
    single(X,Z),
    next(T0,T1),
    not_ge(E,T0,X).

```

```

fal_and(Z) :- and(X,Y,Z), and(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).
fal_and(Z) :- and(_,_Z), or(_,_Z).
fal_and(Z) :- and(_,_Z), or_not(_,_Z).
fal_and(Z) :- and(_,_Z), single(_Z).
fal_or(Z) :- or(X,Y,Z), or(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).
fal_or(Z) :- or(_,_Z), and(_,_Z).
fal_or(Z) :- or(_,_Z), or_not(_,_Z).

```

```
fal_or(Z) :- or(_,_,Z), single(_,Z).  
fal_or_not(Z) :- or_not(X,Y,Z), or_not(X1,Y1,Z), ( X1 \= X ; Y1 \= Y).  
fal_or_not(Z) :- or_not(_,_,Z), and(_,_,Z).  
fal_or_not(Z) :- or_not(_,_,Z), or(_,_,Z).  
fal_or_not(Z) :- or_not(_,_,Z), single(_,Z).  
fal_single(Z) :- single(X1,Z), single(X,Z), X \= X1.  
fal_single(Z) :- single(_,Z), and(_,_,Z).  
fal_single(Z) :- single(_,Z), or(_,_,Z).  
fal_single(Z) :- single(_,Z), or_not(_,_,Z).
```


References

- [1] Melody Morris, Julio Saez-Rodriguez, Peter Sorger, and Douglas Lauffenburger. Logic-based models for the analysis of cell signaling networks. *Biochemistry*, 49:3216–3224, 2010.
- [2] Andrew Capaldi, Tommy Kaplan, Ying Liu, Naomi Habib, Aviv Regev, Nir Friedman, and Erin O’Shea. Structure and function of a transcriptional network activated by the mapk hog1. *Nature Genetics*, 40:1300–1306, 2008.
- [3] L. Glass and S.A. Kauffman. A logical analysis of continuous, non-linear biochemical control networks. *Journal of Theoretical Biology*, 39:103–129, 1973.
- [4] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1973.
- [5] Nir Friedman, Michal Linial, Iftach Nachman, and Dana Pe’er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3/4):601–620, 2000.
- [6] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2462–2467, 2007.
- [7] Angelika Kimmig, Vítor Santos Costa, Ricardo Rocha, Bart Demoen, and Luc De Raedt. On the Implementation of the Probabilistic Logic Programming Language ProbLog. *Theory and Practice of Logic Programming Systems*, 11:235–262, 2011.
- [8] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD-08*, volume LNCS 5211, pages 473–488, Antwerp, Belgium, September 15–19 2008. Springer.

- [9] J. Gebert, N. Radde, and G.-W. Weber. Modeling gene regulatory networks with piecewise linear differential equations. *European Journal of Operational Research*, 181(3):1148 – 1165, 2007.
- [10] Anirban Ghosh and Manju Bansal. A glossary of dna structures from a to z. *Acta Crystallographica Section D*, 59(4):620–626, Apr 2003.
- [11] Jeremy Berg. *Biochemistry*. W.H. Freeman, New York, 2002.
- [12] Wolfram Saenger. *Principles of nucleic acid structure*. Springer-Verlag, New York, 1984.
- [13] John Butler. *Forensic DNA typing : biology and technology behind STR markers*. Academic Press, San Diego, 2001.
- [14] J. D. Watson and F. H. C. Crick. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *nat*, 171:737–738, April 1953.
- [15] Peter Yakovchuk, Ekaterina Protozanova, and Maxim D. Frank-Kamenetskii. Base-stacking and base-pairing contributions into thermal stability of the dna double helix. *Nucleic Acids Research*, 34(2):564–574.
- [16] H. Clausenschaumann. Mechanical Stability of Single DNA Molecules. *Biophysical Journal*, 78:1997–2007, April 2000.
- [17] Qin Yu and Casey D. Morrow. Identification of critical elements in the trna acceptor stem and tc loop necessary for human immunodeficiency virus type 1 infectivity. *Journal of Virology*, 75(10):4902–4906, 2001.
- [18] Miguel Salazar, Oleg Y. Fedoroff, Julie M. Miller, N. Susan Ribeiro, and Brian R. Reid. The dna strand in dna.cntdot.rna hybrid duplexes is neither b-form nor a-form in solution. *Biochemistry*, 32(16):4207–4215, 1993. PMID: 7682844.
- [19] Satu Mikkola, Eeva Stenman, Kirsi Nurmi, Esmail Yousefi-Salakdeh, Roger Stromberg, and Harri Lonnberg. The mechanism of the metal ion promoted cleavage of rna phosphodiester bonds involves a general acid catalysis by the metal aquo ion on the departure of the leaving group. *J. Chem. Soc., Perkin Trans. 2*, pages 1619–1626, 1999.
- [20] Z. Tan. Salt Dependence of Nucleic Acid Hairpin Stability. *Biophysical Journal*, 95:738–752, July 2008.
- [21] Jonathan Pevsner. *Bioinformatics and functional genomics*. Wiley, 2003.

- [22] Sheetal A Mitra, Anirban P Mitra, and Timothy J Triche. A central role for long non-coding rna in cancer. *Frontiers in Genetics*, 3(00017), 2012.
- [23] Michael D Horwich, Chengjian Li, Christian Matranga, Vasily Vagin, Gwen Farley, Peng Wang, and Phillip D Zamore. The drosophila rna methyltransferase, dmhen1, modifies germline pirnas and single-stranded sirnas in risc. *Curr Biol*, 17(14):1265–72, 2007.
- [24] Mads Kaern, William J. Blake, and J. J. Collins. The engineering of gene regulatory networks. *Annual review of biomedical engineering*, 5:179–206, January 2003.
- [25] Gemma Bellí, Eloi Garí, Lidia Piedrafito, Martí Aldea, and Enrique Herrero. An activator/repressor dual system allows tight tetracycline-regulated gene expression in budding yeast. *Nucleic Acids Research*, 26(4):942–947, 1998.
- [26] M Gossen, S Freundlieb, G Bender, G Muller, W Hillen, and H Bujard. Transcriptional activation by tetracyclines in mammalian cells. *Science*, 268(5218):1766–1769, 1995.
- [27] S. Nagahashi, H. Nakayama, K. Hamada, H. Yang, M. Arisawa, and K. Kitada. Regulation by tetracycline of gene expression in *saccharomyces cerevisiae*. *Molecular and General Genetics MGG*, 255:372–375, 1997. 10.1007/s004380050508.
- [28] Barbara Ulmasov, John Capone, and William Folk*. Regulated expression of plant trna genes by the prokaryotic tet and lac repressors. *Plant Molecular Biology*, 35:417–424, 1997. 10.1023/A:1005819007549.
- [29] Damien Eveillard, Delphine Ropers, Hidde Jong, Christiane Branlant, and Alexander Bockmayr. Multiscale modeling of alternative splicing regulation. In Corrado Priami, editor, *Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 75–87. Springer Berlin Heidelberg, 2003.
- [30] A. M Walczak and G. Tkačik. Information transmission in genetic regulatory networks: a review. *ArXiv e-prints*, jan 2011.
- [31] M. Schena. *DNA microarrays: A practical approach*, volume 205 of *Practical Approach Series*. Oxford Univ. Press., Oxford, 1999.
- [32] Hidde De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9:67–103, 2002.

- [33] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.
- [34] Raymond J. Cho, Michael J. Campbell, Elizabeth A. Winzeler, Lars Steinmetz, Andrew Conway, Lisa Wodicka, Tyra G. Wolfsberg, Andrei E. Gabrielian, David Landsman, David J. Lockhart, and Ronald W. Davis. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell*, 2(1):65 – 73, 1998.
- [35] George S. Michaels, Daniel B. Carr, Stefanie Fuhrman, Xiling Wen, and Roland Somogyi. Cluster analysis and data visualization of largescale gene expression data. 1998.
- [36] X. Wen, S. Fuhrman, G. S. Michaels, D. B. Carr, S. Smith, J. L. Barker, and R. Somogyi. Large-scale temporal gene expression mapping of central nervous system development. *PNAS*, 95(1):334–339, 1998.
- [37] Paul T. Spellman, Gavin Sherlock, Michael Q. Zhang, Vishwanath R. Iyer, Kirk Anders, Michael B. Eisen, Patrick O. Brown, David Botstein, and Bruce Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9(12):3273–3297, 1998.
- [38] Michiel De Hoon, Seiya Imoto, and Satoru Miyano. Inferring gene regulatory networks from time-ordered gene expression data of *bacillus subtilis* using differential equations. In *Pac. Symp. Biocomput*, pages 17–28, 2003.
- [39] Ting Chen, Hongyu L. He, and George M. Church. Modeling gene expression with differential equations. In *Pac. Symp. Biocomput*, pages 29–40, 1999.
- [40] Michiel de Hoon, Seiya Imoto, and Satoru Miyano. Inferring gene regulatory networks from time-ordered gene expression data using differential equations. In Steffen Lange, Ken Satoh, and Carl Smith, editors, *Discovery Science*, volume 2534 of *Lecture Notes in Computer Science*, pages 283–288. Springer Berlin / Heidelberg, 2002.
- [41] Erina Sakamoto and Hitoshi Iba. Inferring a system of differential equations for a gene regulatory network by using genetic programming. In *Proc. Congress on Evolutionary Computation*, pages 720–726. IEEE Press, 2001.

- [42] Thomas G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):pp. 49–58, 1970.
- [43] T. G. Kurtz. Limit theorems for sequences of jump markov processes approximating ordinary differential processes. *Journal of Applied Probability*, 8(2):pp. 344–356, 1971.
- [44] Tatsuya Akutsu, Satoru Miyano, and Satoru Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, pages 727–734, 2000.
- [45] Michael A. Savageau and Eberhard O. Voit. Recasting nonlinear differential equations as s-systems: a canonical nonlinear form. *Mathematical Biosciences*, 87(1):83 – 115, 1987.
- [46] Eberhard O. Voit. Symmetries of s-systems. *Mathematical Biosciences*, 109(1):19 – 37, 1992.
- [47] Douglas H. Irvine and Michael A. Savageau. Efficient solution of nonlinear ordinary differential equations expressed in s-system canonical form. *SIAM Journal on Numerical Analysis*, 27(3):pp. 704–735, 1990.
- [48] Diego di Bernardo, Michael J Thompson, Timothy S Gardner, Sarah E Chobot, Erin L Eastwood, Andrew P Wojtovich, Sean J Elliott, Scott E Schaus, and James J Collins. Chemogenomic profiling on a genome-wide scale using reverse-engineered gene networks. *Nat Biotechnol*, 23(3):377–83, 2005.
- [49] T. S. Timothy, D. d. Diego, D. David, and J. J. James. Inferring genetic networks and identifying compound mode of action via expression profiling. *Science*, 301(5629):102–105, July 2003.
- [50] Mukesh Bansal, Giusy Della Gatta, and Diego di Bernardo. Inference of gene regulatory networks and compound mode of action from time course gene expression profiles. *Bioinformatics*, 22(7):815–822, 2006.
- [51] E. P. van Someren, B. L. T. Vaes, W. T. Steegenga, A. M. Sijbers, K. J. Dechering, and M. J. T. Reinders. Least absolute regression network analysis of the murine osteoblast differentiation network. *Bioinformatics*, 22(4):477–484, 2006.
- [52] Jesper Tegnér, M. K. Stephen Yeung, Jeff Hasty, and James J. Collins. Reverse engineering gene networks: Integrating genetic perturbations with dynamical

- modeling. *Proceedings of the National Academy of Sciences*, 100(10):5944–5949, 2003.
- [53] Richard Bonneau, David Reiss, Paul Shannon, Marc Facciotti, Leroy Hood, Nitin Baliga, and Vesteinn Thorsson. The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo. *Genome Biology*, 7(5):R36, 2006.
- [54] Patrik D’haeseleer, X. Wen, Stefanie Fuhrman, and Roland Somogyi. Linear modeling of mrna expression levels during cns development and injury. In *Pacific Symposium on Biocomputing’99*, pages 41–52, 1999.
- [55] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *Proc. the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 695–702, 1998.
- [56] T.E. Ideker, V. Thorsson, and R.M. Karp. Discovery of regulatory interactions through perturbation: Inference and experimental design. In *Pacific Symposium on Biocomputing*, pages 302–313, 2000.
- [57] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim, and Wei Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.
- [58] Gilles Bernot, Jean-Paul Comet, Adrien Richard, and Janine Guespin. Application of formal methods to biological regulatory networks: extending thomas’ asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339 – 347, 2004.
- [59] Grégory Batt, Delphine Ropers, Hidde De Jong, Johannes Geiselman, Radu Mateescu, and Dominique Schneider. Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in escherichia coli. *Bioinformatics*, 21:19–28, 2005.
- [60] Hidde de Jong, Johannes Geiselman, Céline Hernandez, and Michel Page. Genetic network analyzer: qualitative simulation of genetic regulatory networks. *Bioinformatics*, 19(3):336–344, 2003.
- [61] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory. ii. logical analysis of regulatory networks in terms of feedback

- circuits. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 11(1):180–195, 2001.
- [62] M. Kaufman, J. Urbain, and R. Thomas. Towards a logical analysis of the immune response. *Journal of Theoretical Biology*, 114(4):527 – 561, 1985.
- [63] René THOMAS, Anne-Marie GATHOYE, and Lucie LAMBERT. A complex control circuit. *European Journal of Biochemistry*, 71(1):211–227, 1976.
- [64] R. Thomas. Logical analysis of systems comprising feedback loops. *Journal of Theoretical Biology*, 73(4):631 – 656, 1978.
- [65] René Thomas, Denis Thieffry, and Marcelle Kaufman. Dynamical behaviour of biological regulatory networks—i. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Mathematical Biology*, 57:247–276, 1995. 10.1007/BF02460618.
- [66] J. Demongeot, D. Benaouda, and C. Jézéquel. “dynamical confinement” in neural networks and cell cycle. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 5(1):167–173, 1995.
- [67] Jacques Demongeot, Marcelle Kaufman, and René Thomas. Positive feedback circuits and memory. *Comptes Rendus de l’Académie des Sciences - Series III - Sciences de la Vie*, 323(1):69 – 79, 2000.
- [68] Jacques Demongeot, Julio Aracena, Florence Thuderoz, Thierry-Pascal Baum, and Olivier Cohen. Genetic regulation networks: circuits, regulons and attractors. *Comptes Rendus Biologies*, 326(2):171 – 188, 2003.
- [69] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, , and Carolyn Talcott. Pathway logic: Executable models of biological networks. In *Fourth International Workshop on Rewriting Logic and Its Applications (WRLA’2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.
- [70] Hiroaki Kitano, Akira Funahashi, Yukiko Matsuoka, and Kanae Oda. Using process diagrams for the graphical representation of biological networks. *Nat Biotech*, 23(8):961 – 966, 2005.
- [71] Hiroaki Kitano. A graphical notation for biochemical networks. *BIOSILICO*, 1(5):169 – 176, 2003.

- [72] D. Pe’er, A. Regev, and A. Tanay. Minreg: Inferring an active regulator set. In *Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology*, pages S258–S267. Oxford University Press, 2002.
- [73] A. Tanay and R. Shamir. Computational expansion of genetic networks. In *Bioinformatics*, volume 17, 2001.
- [74] L. Chrisman, P. Langley, S. Bay, and A. Pohorille. Incorporating biological knowledge into evaluation of causal regulatory hypotheses. In *Pacific Symposium on Biocomputing (PSB)*, January 2003.
- [75] I.M. Ong, J.D. Glasner, and D. Page. Modelling regulatory pathways in *Escherichia coli* from time series expression profiles. *Bioinformatics*, 18:S241–S248, 2002.
- [76] Nir Friedman, Kevin Murphy, and Stuart Russell. Learning the structure of dynamic probabilistic networks. pages 139–147. Morgan Kaufmann, 1998.
- [77] Anne Smith, Erich D. Jarvis, Alexander J. Hartemink, and Er J. Hartemink. Evaluating functional network inference using simulations of complex biological systems v., 2002.
- [78] V. Anne Smith, Erich D. Jarvis, and Alexander J. Hartemink. Influence of network topology and data collection on network inference. In *Pacific Symposium on Biocomputing’03*, pages 164–175, 2003.
- [79] Sunyong Kim, Seiya Imoto, and Satoru Miyano. Dynamic Bayesian network and nonparametric regression for nonlinear modeling of gene networks from time series gene expression data. *Bio Systems*, 75(1-3):57–65, July 2004.
- [80] Bruno-Edouard Perrin, Liva Ralaivola, Aurélien Mazurie, Samuele Bottani, Jacques Mallet, and Florence d’Alché-Buc. Gene networks inference using dynamic bayesian networks. *Bioinformatics*, 19(suppl 2):ii138–ii148, 2003.
- [81] Norbert Dojer, Anna Gambin, Andrzej Mizera, Bartek Wilczynski, and Jerzy Tiuryn. Applying dynamic bayesian networks to perturbed gene expression data. *BMC Bioinformatics*, 7(1):249, 2006.
- [82] M. K. Stephen Yeung, Jesper Tegnér, and James J. Collins. Reverse engineering gene networks using singular value decomposition and robust regression. 99(9):6163–6168, 2002.

- [83] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 1(1):24–45, January 2004.
- [84] Yizong Cheng and George M. Church. Biclustering of expression data, 2000.
- [85] Leon Sterling and Ehud Shapiro. *The art of Prolog (2nd ed.): advanced programming techniques*. MIT Press, Cambridge, MA, USA, 1994.
- [86] Ben Taskar and Lise Getoor. *Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [87] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.
- [88] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Relational Data Mining*, chapter 13, pages 307–335. Springer, 2001.
- [89] Stephen Muggleton. Stochastic logic programs. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 254–264. IOS Press, Amsterdam, 1996.
- [90] Kristian Kersting and Luc De Raedt. Basic principles of learning bayesian logic programs. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*, pages 189–221. Springer, 2008.
- [91] Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. CLP($\mathcal{BN}$): Constraint Logic Programming for Probabilistic Knowledge. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI03)*, pages 517–524, Acapulco, Mexico, August 2003.
- [92] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707, 2002.
- [93] Vítor Santos Costa, Luís Damas, and Ricardo Rocha. The yap prolog system. *Theory and Practice of Logic Programming*, 12(Special Issue 1-2):5–34, 2012.

- [94] Bwolen Yang, Randal E. Bryant, David R. O'Hallaron, Armin Biere, Olivier Coudert, Geert Janssen, Rajeev K. Ranjan, and Fabio Somenzi. A performance study of bdd-based model checking. In *FMCAD '98*, volume LNCS 1522, pages 255–289, 1998.
- [95] Violet Lee, Scott Topper, Shane Hubler, James Hose, Craig Wenger, Joshua Coon, and Audrey Gasch. A dynamic model of proteome changes reveals new roles for transcript alteration in yeast. *Molecular Systems Biology*, 7(514), 2011.
- [96] David B. Berry and Audrey P. Gasch. Stress-activated genomic expression changes serve a preparative role for impending stress in yeast. *Molecular Biology of the Cell*, 19(11):4580–4587, 2008.
- [97] Andrei Grigoriev. A relationship between gene expression and protein interactions on the proteome scale: analysis of the bacteriophage t7 and the yeast *saccharomyces cerevisiae*. *Nucleic Acids Research*, 29(17):3513–3519, 2001.
- [98] Sean M. O'Rourke and Ira Herskowitz. The *hog1* mapk prevents cross talk between the *hog* and pheromone response mapk pathways in *saccharomyces cerevisiae*. *Genes & Development*, 12(18):2874–2886, 1998.
- [99] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews. Genetics*, 10(1):57–63, January 2009.