COMPUTATIONAL TECHNIQUES FOR INFERRING REGULATORY NETWORKS

by

Irene M. Ong

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

To Mom, for making this dream possible,

Ian, for supporting and sharing it

and

Lillian for making it all worthwhile.

ABSTRACT

In this era where healthcare is one of the world's largest and fastest growing industries, there is great interest in understanding what is happening within our cells and organs at the molecular level. Fortunately, innovations and improvements in technology continue to spur the quantity and types of high-throughput (a process where large amounts of samples can be measured by a system at once) biological data that can be measured. Additionally, abundant information from many years of detailed research can be found in annotated or computationally extracted databases. These data sets, especially combined, have great potential for novel discoveries that can lead to advances in biology and medicine. The main focus of this thesis is the investigation of machine learning techniques for inferring gene regulatory networks from the combination of high-throughput time series gene expression array data and other data sources. A gene regulatory network is a collection of DNA segments in a cell which interact with each other (indirectly through their RNA and protein expression products) and with other substances in the cell, thereby governing the rates at which genes in the network are transcribed into mRNA.

The main contribution of this thesis is related to computational biology. We show how we exploit different types of data and temporal information to determine causality in gene regulatory networks using a probabilistic learning model, namely a dynamic Bayesian network. We further show how combining other sources of relational data with temporal data and using an inductive learning method known as inductive logic programming can help learn more interesting rules for predicting gene expression. We present experiments that show our methods are good at inferring regulatory relationships. Our use of temporal data provides us with more confidence that the relationships learned represent causality, which can be useful in interpreting underlying mechanisms from the inferred results. We further constructed a simplified theoretical model for the modeling

of time series gene expression data to guide experimental decisions. Our empirical evaluation on synthetic data agrees with the results of our theoretical analysis.

A minor contribution of this thesis is related to data mining. We show how we integrate an inductive logic programming (ILP) system, namely FOIL, with a database and utilize statistics to estimate counts to avoid expensive database operations in certain cases. Another contribution involves a different inductive learning system, Aleph, and the use of pathfinding, which finds paths that link relational entities, to help overcome plateaus in the search space by searching for paths within the lattice of a saturated bottom clause. These extensions in ILP systems can also be useful for computational biology as ILP is often applied to diverse biological data because of its ability to accomodate multiple tables, take into account background knowledge, and produce rules comprehensible to biologists.

ACKNOWLEDGMENTS

I am very grateful for having had so many wonderful influences throughout my career, especially throughout graduate school, which has been extremely fulfilling. It is with great pleasure that I extend my gratitude to the many people who have been a big part of my life.

First and foremost, I thank my advisor, David Page, who has in many ways shaped my development as a researcher. David has been a wonderful teacher and mentor, providing guidance when I needed it, yet allowing me the freedom to pursue my own ideas. He has also been very patient, understanding and supportive throughout the trials and tribulations that are part of graduate student life. I will never forget the encouragement he gave me that provided me with the confidence to give my first presentation to a crowd of over a thousand experts. I truly appreciate all that he has taught me.

Secondly, I thank the other members of my thesis committee: Mark Craven, Colin Dewey, Tim Donohue, and Jude Shavlik. They have been another wonderful source of guidance. Tim, early on taught me the importance of making my presentations accessible to people from other fields of study. Mark and Jude have been terrific teachers, not only in the courses that they teach but also in all the seminars that I have attended with them. They have also offered advice, providing different perspectives, which have been helpful for me.

Thirdly, I thank my collaborators: Joe Bockhorst, Jesse Davis, Inês Dutra, Jeremy Glasner, Vítor Santos Costa, Jan Struyf, and Scott Topper. They made our collaborations fruitful and enjoyable. In particular, I learned a lot from discussions with Vítor and Inês. A special thanks goes to Vítor who has been

Rich Maclin has also been another wonderful mentor to me. He has always provided good advice, and has been very supportive and encouraging of my endeavors. Rich has also been a source of humor and I thank him for making graduate life fun.

I thank my fellow students in machine learning (both past and present): David Andrzejewski, Joe Bockhorst, Rich Chang, Aaron Darling, Jesse Davis, Frank DiMaio, Tina Eliassi-Rad, Mark Goadrich, Eric Lantz, Sean McIlwain, Michael Molla, Keith Noto, Louis Oliphant, Andrew Palmer, Yue Pan, Maleeha Qazi, Beverly Seavey, Burr Settles, Adam Smith, Ameet Soni, Lisa Torrey, Michael Waddell, and Trevor Walker. I have learned a lot through seminars, practice talks, discussions and traveling with them. Joe Bockhorst, Mark Goadrich, Michael Molla, and Soumya Ray have been especially good friends as they started in the machine learning group around the same time as I did and we have gone through many of the same hurdles together.

I thank the following sources of funding, all of which have made it possible for me to pursue my own research: U.S. National Institutes of Health (NIH 5 T32 GM08349), U.S. National Library of Medicine (5T15LM005359), U.S. Air Force grant F30602-01-2-0571, and U.S. Department of Energy (DOE-GTL award DE-FG02-04ER25627).

I have met many friends in Madison and am afraid there are far too many to list. Many who started in the computer science graduate program in the same year as I did or a few years before or after, as have a few outside of the department, have become close friends. In particular, Matt Anderson, Drew and Jenn Bernat, Jen and Brad Beckmann, Ana and Pedro Bizarro, Trey Cain and Heather Swanson, Rich Chang, Mihai Christodorescu and Sondra Renly, Jesse Davis and Allison Holloway, Vuk Ercegovac and Magdalena Karakova, Mark and Laura Goadrich, Jaime Frey, Alexey Loginov and Wendy Sthokal, Sean McIlwain, Amy Millen, Christina and Richard Oberlin, Andrew Palmer, Maleeha Qazi, Dan Sorin, Vanitha Suresh, Corinna and Philip Wells. Jen Beckmann, Jenny Cotner, Drew Larsen, Kevin Moore, Florentina Popovici, Maleeha Qazi, Drew Repoza, Laurie Schubert, Karen Spach, Hao Wang and the members of the Oak Street Ramblers and their families have become especially cherished friends.

I also thank Bev, Fred and Jason Hartline, Jason Miller, Michel van der List, Sridhar Hannenhalli, Pangkaj Agarwal, and Jaime Duckworth for directly or indirectly encouraging me early on to pursue this dream.

I am grateful to my long time best friends, Kenneth Yap, and Min Zhong who despite being far away have always been there for me. Min, with her regular weekly calls has been very supportive throughout. Kenneth, whom I have known since my childhood days, has been a source of inspiration and confidence booster for as long as I've known him. We have shared many phases of the journey of our lives through many extremely long phone conversations among other experiences.

Many thanks to my extended family, who have been especially supportive and helpful despite living many miles away. My parents, Julie and Larry Beers, have provided support and encouragement, as have my brother, Ronnie Ong. My in-laws, Bernard and Jean Alderman, have been loving, and wonderful in keeping up with the progress of my work and have always celebrated each hurdle I passed.

Lastly I thank my daughter, Lillian, and husband, Ian Alderman. Lillian teaches me to put things in perspective and to enjoy the simple things in life. Ian has been a wonderful partner in all things. My life has been made so much richer for all that we share.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

COMPUTATIONAL TECHNIQUES FOR INFERRING REGULATORY NETWORKS

Irene M. Ong

Under the supervision of Associate Professor C. David Page

At the University of Wisconsin-Madison

In this era where healthcare is one of the world's largest and fastest growing industries, there is great interest in understanding cells at the molecular level. Fortunately, innovations in experimental technology continue to spur the quantity and types of high-throughput biological data that can be measured. Analyzing the combination of these data sets could lead to novel discoveries in biology and medicine. The main focus of this thesis is the investigation of machine learning techniques for inferring gene regulatory networks from the combination of high-throughput time series gene expression array data and other data sources.

The main contribution of this thesis is to computational biology. We exploit background knowledge and temporal information to determine causality in gene regulatory networks using a dynamic Bayesian network. We further combine other sources of relational data with temporal data and use an inductive learning method known as inductive logic programming (ILP) to learn rules for predicting gene expression. We also constructed a simplified theoretical model for the modeling of time series gene expression data to guide experimental decisions.

A minor contribution of this thesis is to data mining. We show how we integrate an ILP system, FOIL, with a database and utilize statistics to avoid expensive database operations in certain cases. Another contribution involves a different ILP system, Aleph, and the use of pathfinding to search for rules that link the entities of interest. These extensions in ILP can also be useful for computational biology as ILP is often applied to diverse biological data because of its ability to accomodate multiple tables, take into account background knowledge, and produce rules comprehensible to biologists.

C. David Page

Chapter 1

Introduction

Technological advances have always been a catalyst for change. The development of the personal computer and the internet have made computers ever more ubiquitous, and an integral part of science, business and society. When advancement in experimental technology made high-throughput biological data collection feasible, a new field of study, namely computational biology, emerged and changed the way biological research is done. In the past, most discoveries in biology involved years of research and expensive experiments in order to uncover few details about a specific gene, protein or process. The competition to be the first to complete the sequencing of the human genome propelled the field of computational biology as it brought together advances in sequencing technology and computational techniques. As a consequence, scientists today can use high-throughput methods to simultaneously measure the activity levels of thousands of genes or proteins in a population of cells and then relegate the task of uncovering the underlying process or pattern to computational methods.

The primary focus of this thesis is the development of computational techniques for inferring gene regulatory networks from time series measurements and other biological data. Inferring gene regulatory networks involves learning the interaction of genes or the circuitry that underlies the regulation of cells in response to internal and external stimuli. Gaining insight into the regulation of cells can lead to possible advances in medical treatment, as many diseases are caused by a breakdown of a particular component along a pathway. Understanding the design of these networks will enable the most effective molecular targets within the pathway to be chosen for pharmaceuticals and possibly further engineering methods [125]. In order to learn from biological data we use

machine learning methods to infer this network from a combination of dynamic and static information.

1.1 Machine Learning

Machine learning is a broad subfield of artificial intelligence concerned with the development of algorithms and techniques that allow computers to "learn" from observed facts. The task of learning is easy for humans, but when datasets are large or have many complex features, it is helpful to have techniques that can generalize from data. This ability to automatically generalize from data in any domain is a strength of machine learning. This strength has led to the use of machine learning in a wide spectrum of applications ranging from the sciences to the social sciences including health care, business and crime prevention. Fueling the trend are the digitization of information, ever faster and cheaper computing, and the explosion of online networks and data collection.

A common technique of machine learning is known as supervised learning, where a learning algorithm generates a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem, whereby a learner learns to approximate a function which maps a vector of features, into one of several classes by analyzing several input-output examples [100]. The kind of input-output examples that are required for a system that has to distinguish whether a new patient has breast cancer or not might include mammography images, expert mammographers' annotations of features in the mammography images, and the diagnoses of malignant or normal for each patient [20].

The process of learning typically involves training, validation and testing to determine the accuracy of a learned model. A training set of examples is analyzed and some representation of the newly learned knowledge is stored. Parameters of the learning algorithm may be adjusted by optimizing performance on a subset, known as a validation set, of the training set. After parameter adjustment and learning, the performance of the algorithm may be measured on a test set that is separate from the training set. Cross-validation is a way of partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while the other subset(s) are retained

for subsequent use in confirming and validating the initial analysis. Cross validation is important in guarding against testing hypotheses suggested by the data, which can be biased especially if there are few samples. Cross validation also provides an approach to estimate variance of the models learned from different training sets.

1.2 The Gene Regulatory Network Problem

Many years of painstaking, laborious research in biological systems have provided us with the understanding that a large number of functionally diverse and often multifunctional sets of molecules interact selectively over space and time in response to stimuli [86]. However, many of the details including the timing and interactions among DNA (genes), mRNA and proteins in regulatory networks (Figure 1.1) remain unknown. Over the past decade, high-throughput genomic and proteomic technologies have vastly transformed the approach to interpreting processes of cellular systems by ushering in the combination of experimental and computational techniques to study the inner workings of the cell [149].

Every cell contains a large number of *genes*, each of which codes for machinery that keeps the cell alive. Specifically, a *gene* is comprised of DNA bases that encode the information required for the construction of a specific *protein* via an intermediary molecule known as RNA. The *expression* of genes is the process whereby the DNA code is *transcribed* into RNA, which then in turn is *translated* into a protein. Individual genes are expressed at different levels and at different times depending on environmental factors (such as temperature change or starvation) and the stage of the cell in its life cycle.

At the molecular level, genes are transcribed when *RNA Polymerase* (RNAP), a protein complex that recognizes and binds to a particular code in the DNA sequence called a *promoter*, unravels the DNA and traverses the bases in a particular direction, simultaneously synthesizing mRNA from the DNA template. This proceeds until RNAP reaches a code in the DNA that causes the RNAP to fall off and release the mRNA product, ending transcription. Regulation of transcription is not dependent merely on the recognition of a promoter by RNAP. There are regions near the promoter

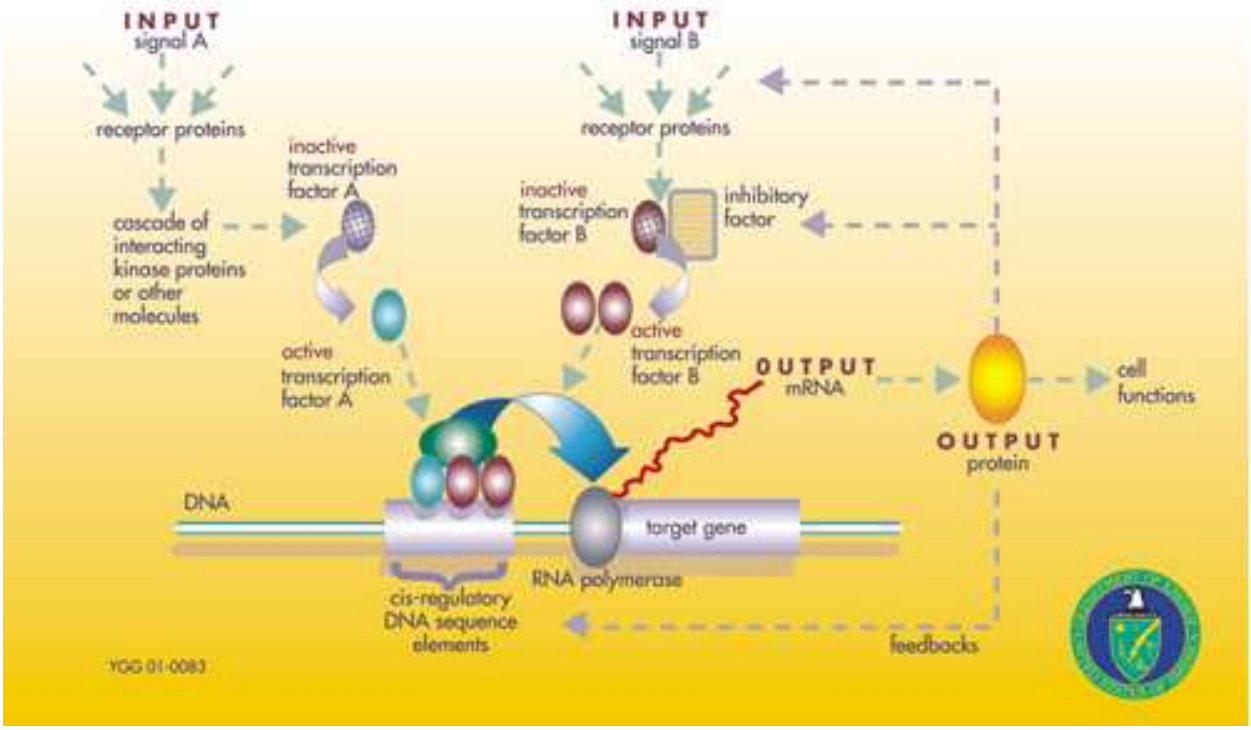


Figure 1.1 Gene regulatory network [53].

called *transcription factor binding sites*, which are recognized and bound by proteins called *transcription factors* that either enhance or inhibit the affinity of RNAP for certain promoters [69].

Usually, the RNA product then becomes involved in a process called translation and serves as a blueprint for synthesizing proteins in a process called translation. RNA binds to a large protein complex known as the ribosome, which matches each three-base codon of the mRNA to a specific form of transfer RNA (tRNA) containing the complementary three-base sequence. This tRNA, in turn, transfers a single amino acid to a growing protein chain. This sequence of events is known as the central dogma (Figure 1.2).

In some organisms such as bacteria, RNA molecules can arise from the transcription of more than a single gene. A set of genes transcribed into a single RNA molecule is known as an *operon*. The concept of an operon allows genes coding for proteins that work cooperatively to be "switched on" or "switched off" as a group.

In an effort to detect the expression of genes, molecular biologists have utilized the *complementarity* of DNA and RNA molecules. DNA bases are comprised of Adenine (A), Cytosine (C), Guanine (G) and Thymine (T), and RNA bases are similarly comprised of A, C and G along with Uracil (U) in place of T. Since each base strongly prefers to bind with its complement, A with T (or U) and G with C, a strand of DNA or RNA has a strong affinity for its *reverse complement*.

This property has been adopted to create probes (used as bait to detect complementary RNA molecules) attached onto a surface to detect the expression of hundreds of thousands of genes. Gene microarrays (Figure 1.3) allow the measurement of expression of a protein intermediate called mRNA [135], making it possible for scientists to measure the activity level of thousands of genes simultaneously, providing a more complete snapshot of processes happening within the cell. Time series expression experiments can provide even more information by giving a sense of the dynamics of the expression of thousands of genes.

Ideally, scientists would like to directly measure protein abundance and *phosphorylation* activity, but this is difficult with current technology since proteins are much more complex molecules. As proteins are synthesized from RNA, they fold into their lowest energy conformation and may have to further associate with other proteins to form a complex before they can carry out their tasks.

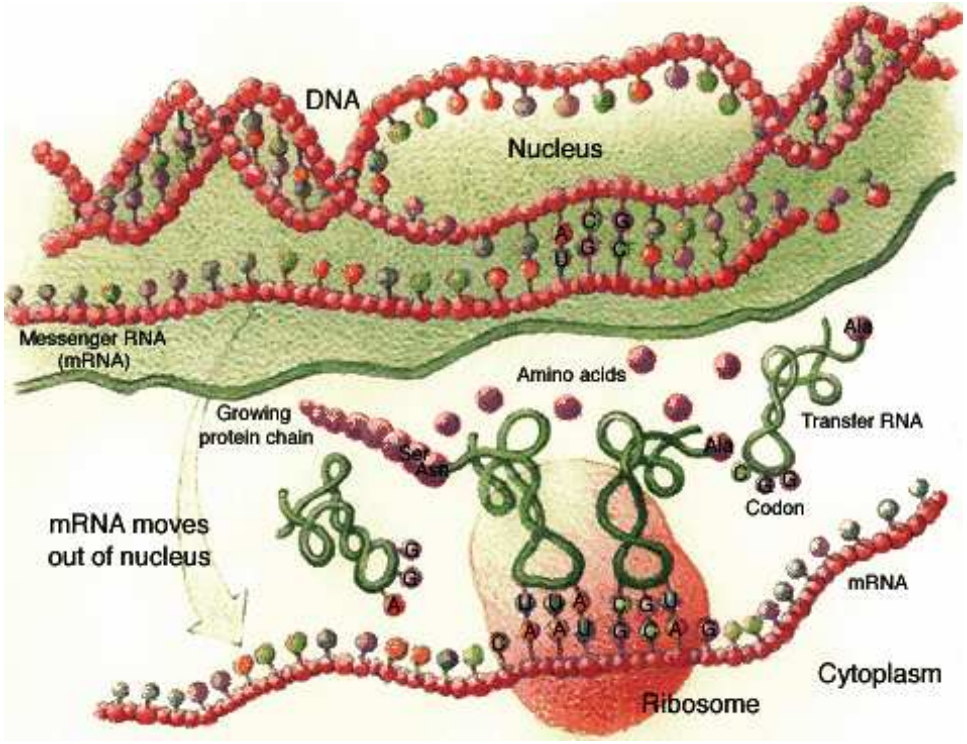


Figure 1.2 Central dogma [54].

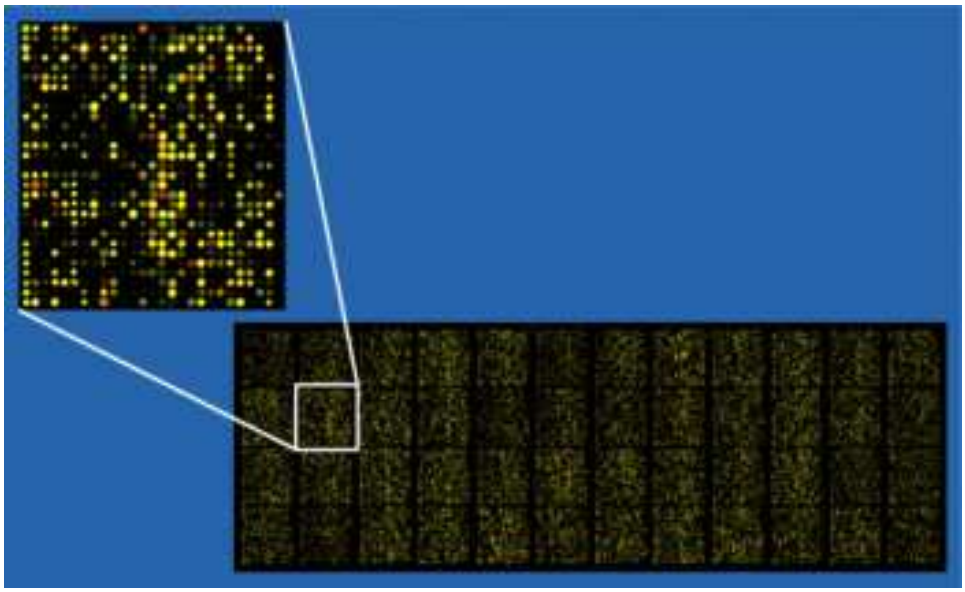


Figure 1.3 Example of an approximately 40,000 probe microarray with enlarged inset to show detail.

Additionally, some proteins may require post-translation modification such as *phosphorylation* in order to activate the protein. Furthermore, some proteins are mobile while others are embedded in membranes, making them static. There are currently ways to detect various proteins using 2-D gel electrophoresis, mass spectroscopy, or nuclear magnetic resonance; however, each has drawbacks. Post-translational modifications can also be detected in a high-throughput fashion by mass spectroscopy, but this technology is still being developed.

Decades of research to tease out specific details about genes and other aspects particular to each model organism exists in numerous databases. There are databases containing information ranging from gene related data to protein-protein interactions to transcription factor data that are freely accessible to the public. One of the biggest challenges in computational biology is to make sense of the vast amounts of information and data that have been collected by various laboratories on different samples, under different environments via different technologies [40].

Scientists are interested in learning the network of gene-gene interactions that signify genes activating other sets of genes. This task presents numerous challenges including, but not limited to:

- Training data contain missing values, there is noise in the data, and the process is only partially observable.
- Training data are heterogeneous.
- Data are measurements from a population of cells.
- Training data have different time steps between and within experiments.
- Data are assumed to be sampled at the appropriate intervals for experiments and genes.

We address a few of these issues in this thesis and propose ways to possibly address others in future work.

The task that motivates the work of this thesis is:

Given:

- Time series expression data for a subset of genes in a genome.
- Relational data for the particular organism such as known and predicted operon to gene maps, transcription factors of genes, protein-protein interactions and proteins that are known to be phosphorylated by particular kinases.

Do:

- Use machine learning methods to learn models or rules of the regulatory network from subsets of different sources of the data above.

1.3 Thesis Statement

We propose and evaluate machine learning algorithms for the difficult task of inferring regulatory networks. We hypothesize that current methods for inferring regulatory networks can be improved by: (i) exploiting prior knowledge, (ii) exploiting temporal information to determine causality, (iii) using theoretical results developed for the modeling of time series gene expression data to guide experimental decisions, (iv) learning small subnetworks using multiple sources of data, and (v) learning rules about regulatory networks.

1.4 Outline

The remainder of this thesis is outlined as follows:

- Chapter 2 describes the use of a probabilistic model for predicting regulatory networks from the combination of time series microarray data and known and predicted operon to gene maps.
- Chapter 3 describes the theoretical analysis of learning with dynamic Bayesian networks (DBNs) and using the computational model to guide the design of time series experiments.
- Chapter 4 presents a method for scaling FOIL for multi-relational data mining of large datasets.

- Chapter 5 extends the approach of searching for rules supported by the data by performing pathfinding within the lattice of saturated bottom clauses.
- Chapter 6 describes the use of inductive logic programming for learning rules about regulatory networks from time series expression data and relational data.
- Chapter 7 contains concluding remarks, including a review of the contributions of this thesis and possible future directions.

Chapter 2

Modeling Regulatory Pathways in *E. coli* from Time Series Expression Profiles

2.1 Introduction

Living cells contain thousands of genes, each of which codes for one or more proteins. Many of these proteins in turn regulate expression of genes through complex regulatory pathways to accommodate changes in their environment or carry out the organism's developmental program. The key to understanding living processes is uncovering this genome-wide circuitry that underlies the regulation of cells.

Genome-wide DNA microarrays are a powerful tool, providing a glimpse of the signals and interactions within regulatory pathways of the cell. They enable the simultaneous measurement of mRNA abundance of most if not all identified genes in a genome under normal conditions or under various treatments or perturbations. A drawback of the current technology of DNA microarrays is that low mRNA expression levels can be very hard to detect. Additionally, steady state or single time point expression profiles do not allow us to discover sequences of regulatory events. The difficulty in detecting low mRNA expression levels can be controlled in some genes by the use of biological knowledge in the analysis. The latter problem can be alleviated by performing time series experiments using DNA microarrays, which will provide a better picture of the signals and interactions over time[1].

Friedman *et al.* [45] were the first to address the task of determining properties of the transcriptional program of an organism (Baker's yeast) by using Bayesian networks to analyze gene

[1]Time series experiments will provide a better understanding of the interactions within the cell provided the time steps are taken within intervals appropriate for capturing important molecular activity.

expression data. Their method can represent the dependence between interacting genes, but it does not show how genes regulate each other over time in the complex workings of genetic networks. Analysis of time series data potentially allows us to determine regulatory pathways rather than just associating genes that are co-regulated together.

In certain organisms such as *Escherichia coli*, there are many sets of genes that are already known to be transcribed together and hence strongly co-regulated. These sequences of genes that are transcribed together into mRNA on their way to being expressed as proteins are known as operons. In this case, since we already know or can predict which genes are regulated together, it would be ideal to use this knowledge in the analysis technique rather than relearn it.

Using a Dynamic Bayesian network (DBN), a close relative of Bayesian network (BN), has several advantages. In addition to being well suited to handling time series data, this framework can handle missing data in a principled way as well as model stochasticity, prior knowledge and hidden variables [105]. To our knowledge Friedman *et al.* [46] and Murphy and Mian [105] are to be credited with first proposing the suitability of DBNs for modeling time series gene expression microarray data. The primary contribution of this work is to test this DBN approach on real time series microarray data. A secondary contribution is the incorporation of the results of a previous application of Bayesian inference (naïve Bayes) as background knowledge for this new application. This naïve Bayes approach used a variety of evidence sources, including earlier microarray data from the Blattner Laboratory at the University of Wisconsin, to predict the operons in *E. coli*. The goal of that work was to produce an accurate operon map that could later be used in the prediction of regulatory pathways in *E. coli*.

The present chapter describes a next step in this direction. The focus of this chapter is to address how one could approximately model the interactions of sets of genes automatically using prior biological knowledge and time series expression profiles. Does the use of prior knowledge or the use of time series expression profiles help determine broader correlations? Can we learn hierarchical connections between sets of co-regulated genes, and ultimately learn connections between multiple signal transduction pathways?

We introduce an approach to determining transcriptional regulatory pathways by applying Dynamic Bayesian network to time series gene expression data from DNA microarray hybridization experiments. Our approach involves building an initial DBN structure that exploits biological knowledge of operons and their associated genes. We further use a domain expert's best guess to initialize the probabilities of how the state of an operon might affect the genes within that operon or that of another operon.

We evaluate our approach on a study by Khodursky *et al.* [83], who performed time series experiments to analyze gene expression in response to physiological changes that affect tryptophan metabolism in *E. coli*.

2.2 Materials: Data and Software

To test our hypotheses, this chapter reports the analysis of time series gene expression data from Khodursky *et al.* [83]. This data set is used because it is focused on tryptophan metabolism, a well studied regulatory process, making it an excellent check for the reverse engineering of a genetic network. Our eventual goal is to develop a tool for analyzing larger collections of time series expression data on *E. coli*.

It should be noted that a common problem with current microarray expression data is a small number of data points and a large number of features. This is especially true of time series data. The present data set consists of 12 data points, from 4 time steps under tryptophan-rich conditions and 2 sets of 4 time steps under tryptophan-starved conditions. These data points consist of 169 genes that were selected based on their expression levels by Khodursky *et al.* [83], and were the only data made available at the start of the present study. Nevertheless it is hoped that discretization and reasonable priors will partially offset noise and permit useful results to be obtained. All the data for both conditions are used (except where indicated) in each of the experiments below to learn how the different environmental conditions affect the regulatory pathway.

The Bayes Net Toolbox [104] software package written by Kevin Murphy was used for the experiments in this chapter because it already provided the necessary functionalities for building Bayesian networks, as well as an implementation of Expectation Maximization (EM) for learning

the conditional probability tables. We constructed the initial BN structure and learned the parameters of the model using the methods provided in Bayes Net Toolbox. Within this framework we implemented the structure search described in section 2.3.4.

Our operon map includes both known operons from Salgado *et al.* [134] and predicted operons from Craven *et al.* [27]. The latter work maps every known and putative gene in the *E. coli* genome into its most probable operon. This map makes the simplifying assumption (rarely but occasionally violated in reality) that every gene appears in exactly one operon. The accuracy of the map (percentage of genes placed in the correct operon) is estimated at about 95% using 10-fold cross-validation.

We assume that this operon map is correct and use it to build our initial BN and DBN structures[2]. Furthermore, the initial probabilities used in our BN and DBN structures are Dirichlet priors obtained from a domain expert. The initial probabilities for a DBN could be dependent on the nature of the experiments and the amount of elapsed time between time points[3]. However, our only insight into these dependencies was largely gleaned from looking at the data, hence, we did not encode these insights into our DBN to avoid biasing our results.

The evidence variables in our BN and DBN are the discretized gene expression levels from the experiments with excess tryptophan and tryptophan starvation. We define up regulated ($\uparrow$) or down regulated ($\downarrow$) as the possible discrete values to avoid choosing arbitrary thresholds. In particular, we compare the expression levels between two consecutive time series measurements to determine whether there was an increase or decrease in expression levels. Note that we are determining the relative change in expression from one time step to another (even for the non time series BN model) rather than absolute absent or present calls.

[2]The full operon map, with an interactive graphical interface, is available online at http://apps.biostat.wisc.edu/~ml-group/GenomeViewerButton.html.

[3]If time points are not equally spaced, we may want to initialize the DBN with different probabilities across time points.

2.3 Dynamic Bayesian Network

We describe in detail how we use prior knowledge to build an initial dynamic Bayesian network structure and then use that initial structure to learn a regulatory model from microarray data.

2.3.1 Modeling Relationships Among Genes

The task of automatically discovering a model that represents relationships among genes from noisy expression data involves a significant amount of uncertainty. The entire experimental process allows for the introduction of uncountable variables as well as measurement errors. Also, the fact that we can only partially observe the happenings among a collection of cells makes it impossible to construct an accurate model from expression data. Therefore it is helpful to model this uncertainty. Instead of simply stating gene A and gene B are correlated, probability provides us with a way to express how certain we are about the correlation. If evidence strongly suggests that gene A and gene B are highly correlated, (i.e. the data shows that when expression level of gene A is up regulated (↑), then gene B is also up regulated) then the probability that gene B is ↑ given that gene A is ↑ would be close to 1, otherwise it would be closer to 0. This probability assignment can be denoted as P(gene B=↑ |gene A=↑) = 0.95.

A visual, intuitive and compact way of representing relationships between genes is via the use of graphical structures. We let genes A and B be represented by nodes and an arc from gene A to gene B denote that gene A influences gene B. We associate small probability tables with the nodes to summarize how gene B is affected by gene A (its parent) and how gene A is not affected by anything since it has no parent. Evidence or data for gene A and B are assigned to gene A and B's nodes respectively and are used to adjust the values in the local probability tables. This example of a Bayesian network is shown in Figure 2.1(a).

In order to find the relationships among genes in our dataset, we can use the BN model to represent all the genes in our dataset. Initial or prior probability settings for the local probability tables can be uniform ($\frac{1}{n}$ where n is the number of possible values) if no prior information is

known. The local probability tables can then be updated automatically based on actual counts of the data. Now we can perform a search for the most likely graph given the data.

We construct such a network, called BN_{reg}, and perform the structure search described in section 2.3.4 We later compare the results of this network to that of a Bayesian network with an explicit model of operons. By performing this comparison we will be able to determine whether the latter model is better at learning useful correlations among genes than the straightforward approach of BN_{reg}.

2.3.2 Incorporating Prior Knowledge or Environmental Factors

There are three important reasons to incorporate explicit operon nodes into the BN model even though operon transcription levels are not observed. First, if we use nodes for genes only, and allow the learning algorithm to induce arcs between genes, it will induce many "useless" arcs between genes in the same operon. For example, if gene A and gene B are both in operon O, then we would expect the expression level of gene A to be an excellent predictor of the expression level of gene B, but this would provide no new insight. Second, incorporation of operons in the model can help combat problems due to noise. For example, the operon that codes for tryptophan, *trpLEDCBA* (also known as *trp*), contains a leader, *trpL*, whose expression did not meet the selection of gene-specific thresholds in the microarray data, and five genes: *trpA*, *trpB*, *trpC*, *trpD*, and *trpE*. Because of noise in microarray experiments, the measured expression level for *trpC* might be low. But the five different gene expression measurements give us essentially five independent indicators of *trp* transcription, reducing the effect of noise in the measurement of *trpC* expression. Third, by searching for interactions among operons rather than genes, we reduce the space of possible models.

Let us assume that we know gene A and gene B are co-transcribed genes in an operon, operon O, and that operon O is transcribed into mRNA when it is initiated by a molecule called an activator, activator O. The fact that gene A and gene B are in the same operon explains the high correlation between the two genes. Hence we can restructure the graph to represent this knowledge in Figure 2.1(b). Because we cannot directly measure the operon or activator's expression level, we regard

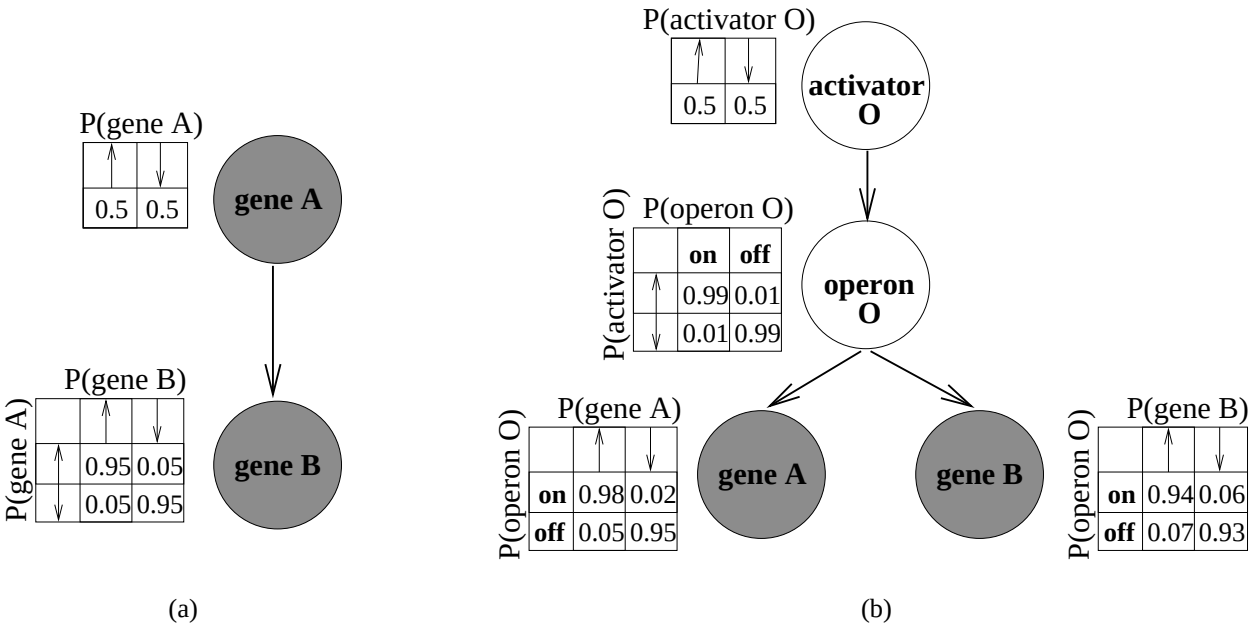


Figure 2.1 (a) Example of a Bayesian network structure. ↑ represents up regulation and ↓ represents down regulation. (b) Nodes that are not shaded are hidden, i.e. nodes without observable data, and shaded nodes indicate nodes that have observable data. Hence the operon and activator nodes are hidden and the gene nodes are observed nodes.

them as hidden nodes since we do not have any evidence for those nodes. If technology permitted us to obtain a measure of the amount of activator molecule in a cell or if we know the amount of activator molecule present in the environment, we would model the activator node as an observed node just like genes A and B. The values in the local probability tables for the hidden nodes are estimated from the graph structure, observed data and initial or prior probabilities associated with the hidden nodes.

Note that the local probability tables for gene A and gene B have changed from Figure 2.1(a). Gene A and gene B now depend on their parent, operon O. While gene A and gene B are correlated, once we know the value of operon O (that it is activated), gene A becomes independent of gene B. Additionally, genes A and B are independent of activator O given its parent, operon O. This is known as conditional independence, which is a key property of Bayesian networks. The lack of arcs implies conditional independence, i.e. a node is independent of all non-descendent nodes in the graph given its parents. A fundamental property of BNs is that given an acyclic graph and the set of local probabilities associated with it (also known as conditional probability tables since the probabilities are conditional on the parent's values) we can uniquely determine the joint probability distribution.

We build our initial BN structure with the model of operons as described above for all the genes in our dataset from our operon map. Since an operon's transcription level affects the expression levels of the genes in that operon, we show this causality with arcs from the operon to its associated genes. Uniform priors are used for the operons, and a domain expert's best guess is used to set the informative priors for how the effect on an operon would affect the genes within that operon. The latter initial probability values, the same as that in our initial DBN model, are shown abstractly in Figure 2.4. The search results for this BN model with operons, BN_{op}, are compared with that of BN_{reg} in section 2.4.

Two other experiments using the BN_{op} structure are performed to determine whether separating the data based on the treatment of the cells would allow us to learn a finer structure. BN_{op_excess} uses the data under tryptophan-rich conditions, whereas BN_{op_starve} uses data under tryptophan-starved conditions.

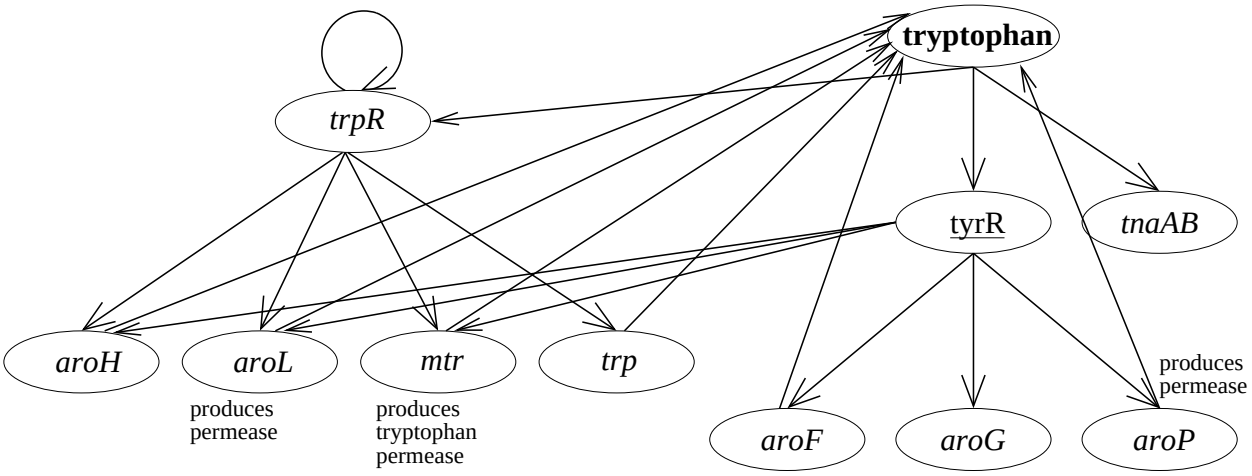


Figure 2.2 A model (not a BN graph) of how the 9 key operons (italicized) in the tryptophan regulon, groups of operons that are co-regulated, influence each other. This structure was constructed by the authors based on Khodursky *et al.* [83]. Operon names are abbreviated. The **tryptophan** node represents the molecule. tyrR is not part of the tryptophan regulon but it influences key operons within the regulon. The **tryptophan** and tyrR nodes serve to connect the interactions between the 9 key operons.

2.3.3 Modeling the Concept of Time

Time series expression data can provide insight into causality[4] and the regulation of cells as they change over time. Dynamic Bayesian networks gracefully scale up BNs to handle the analysis of time series data. In addition, DBNs can also model feedback loops, which are not possible for BNs due to the acyclicity constraint. To see why this is an important feature for modeling regulatory pathways, see Figure 2.2, which shows how some operons rely on a feedback mechanism to regulate transcription.

The Dynamic Bayesian network relies on the same properties as that of Bayesian networks with the addition of modeling genes or operons as they evolve over time. Using a simplified version of the model from Figure 2.1(b) as an example, we show in Figure 2.3 how the concept of time can be modeled by simply replicating the structure for each time step. The structure at time t=0 is not the same as the other time slices because the purpose of operon O at t=0 is to model the effect it has on operon O at t=1 (to start off the chain).

[4]Causality can also be inferred by using the method proposed by Pe'er *et al.* [116] if cells with deletions or mutations of specific genes are available.

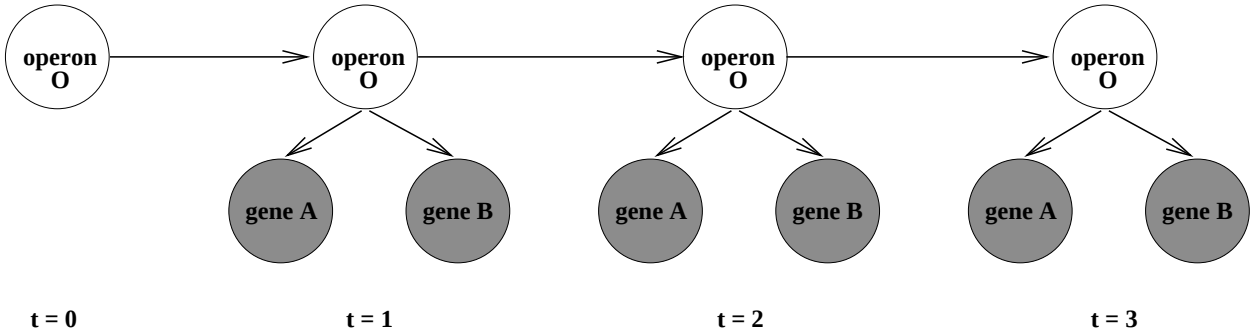


Figure 2.3 An example of a Dynamic Bayesian network structure. Time slices are represented by t=0, t=1, ..., t=n, where n is the number of time step experiments for which there are observable data.

Since each time slice in a DBN is identical in structure to the next, we can just replicate the structure of BN_{op} four times (our data has four time steps) to build our initial DBN structure modeling operons. This leaves the arcs from the hidden variables of one time step to the hidden variables of the next time step undetermined. Since an operon's expression level from one time step typically reflects its expression level at the next time step, we add these arcs as shown in the detailed DBN structure in Figure 2.5.

Note that the operons at time t+1 (operon(i)$_{t+1}$, where i indicates the ith operon) are independent of operon(i)$_{t-1}$ given operon(i)$_t$ because of the conditional independence assumptions. This states that the future is independent of the past given the present. We can work around this assumption by adding an additional arcs from operon$_{t-2}$ to operon$_t$ to indicate that the present also depends on the events from two time steps ago. However, the computational costs increase exponentially.

As before, the domain expert's best guess is used to set the initial probability values for all the 142 hidden nodes and 169 evidence nodes. The abstract structure of our initial DBN model, DBN_{op}, along with the prior probabilities are shown in Figure 2.4. Any additional arc among hidden nodes, as well as all posterior CPT probabilities, must be inferred from time series microarray data for *E. coli*.

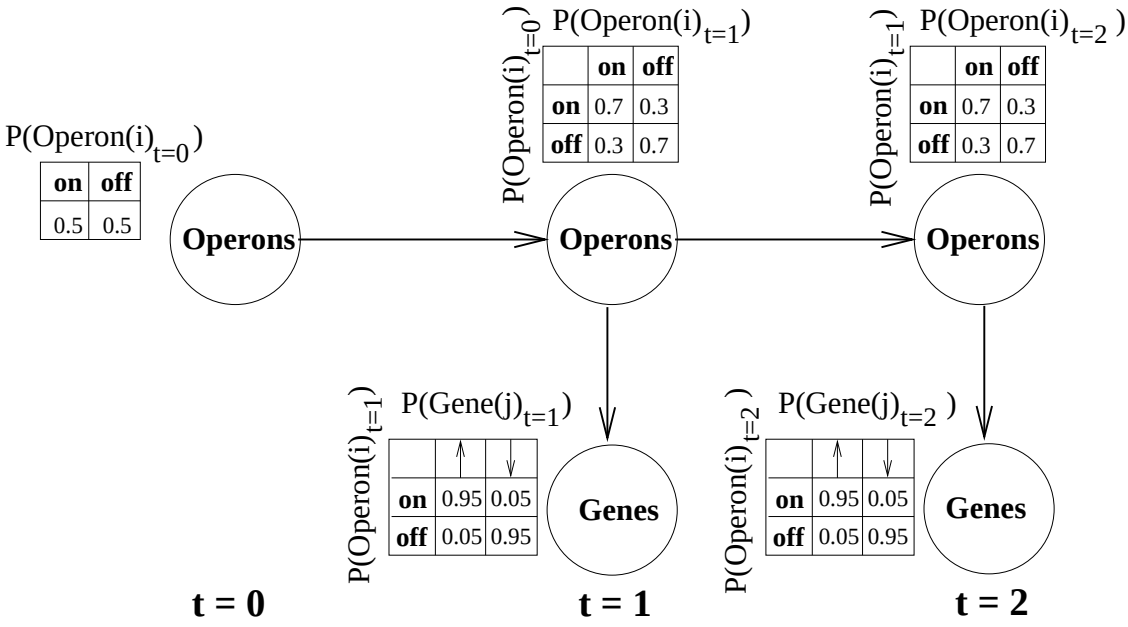


Figure 2.4 Abstract Dynamic Bayesian network structure with conditional probability tables (CPTs) for each arc in the model. Time slices three and four (not shown) are identical to time slice one and two.

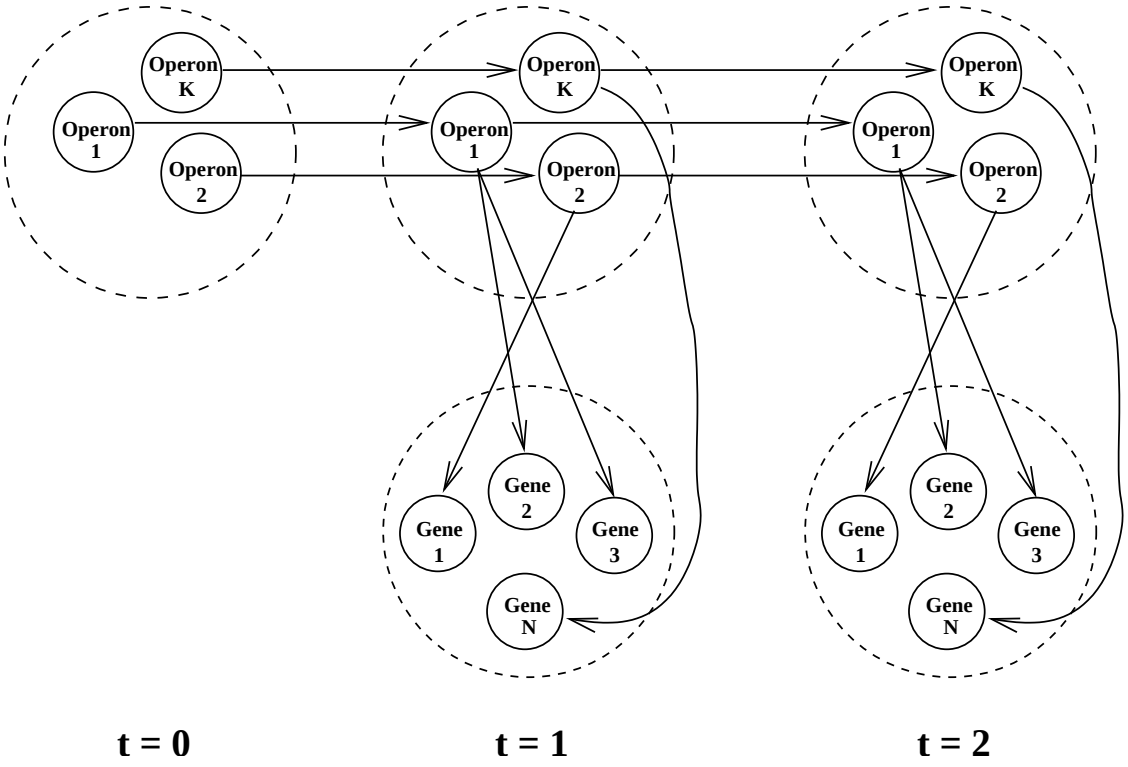


Figure 2.5 Detailed view of our initial Dynamic Bayesian network structure showing intra and inter time slice connections

2.3.4 Structure Learning

To perform structure learning for our BN models, we focused on key genes or operons known to be affected by the absence or presence of tryptophan in the environment. For BN_{reg}, there are 15 such key genes. Similarly, for BN_{op} we focused on the corresponding 9 key operons. For each of these genes (operons) in BN_{reg} (BN_{op}), we consider all the other genes (operons) as possible parents. At each step, we use the Expectation Maximization (EM) algorithm to update all CPTs in the model to give a (local) maximum log likelihood. EM will infer values for the hidden variables as well as missing observations. The log likelihood score from the previous step is used as the scoring measure to select the 20 most probable parents. This is done because EM is not guaranteed to find the maximally probable parent and because a large number of structures have the same or very close scores.

Because of limited data, we consider only simple DBN structural models in which each operon has at most two incoming arcs, from (1) the same operon at the previous time step, and (2) one other operon from the previous time step. Each operon begins with one parent—the same operon at the previous time step. In our full algorithm, for each operon we consider adding a different operon from the previous time step as a second parent. Each potential parent is considered. For each such potential second parent, the EM algorithm is employed. If any choice of second parent increases the log likelihood, then the choice that provides the highest log likelihood is selected.

In general, the preceding cycle through all the operons may need to be repeated several times for convergence to a locally optimal structure. Despite our structural restrictions the run time would take over 9 months using the junction tree algorithm as implemented in BN Toolbox. This is because of the large number of nodes (142 operons, 169 genes, for four time steps) in our DBN structure. For the long-term, we are experimenting with approximate approaches to speed up the computations. For the short-term, we focus the algorithm on the 9 key operons; the algorithm cycles once through these only, but *all* the other 141 operons are considered as potential parents. For each operon we record the best 20 choices for the second parent.

2.4 Results

The results from the BN_{reg} and BN_{op} experiments indicate that modeling operons in the BN structure provide a more comprehensive view of the tryptophan regulon, groups of operons that are co-regulated. Without the operon structure, BN_{reg} found some correlations between genes in the same operon but missed many correlations between genes in different operons.

The BN_{op} structure was able to identify correlations with direct siblings (other children of a node's parent), parents, or children for 4 of the 9 operons. *trp*'s parent, *trpR*, and siblings, *aroH* and *mtr*, were among *trp*'s 20 most probable parents. Similarly, each of *trpR*, *mtr* and *aroH* identified correlations with each other and *trpR*. The other 3 operons showed correlations with 2 of the 9 key operons.

The BN_{reg} results showed that all 15 key genes correlated with a subset of these six genes: *trpR*, *trp*, *aroH*, *mtr*, *aroF* and *aroP*. BN_{reg} found correlations between some but missed other genes within the same operon. All five genes in the *trp* operon were found to be probable second parents when any of the 5 were present. However, *tnaA* and *aroF* were not always found to be correlated with *tnaB* and *tyrA* respectively.

It was interesting that the results for BN_{op_excess} showed that the operon *hisGDCBHAFI* influenced 5 of the 9 key operons. These histidine genes might be correlated to operons in the tryptophan regulon or alternatively, the cells may have consumed all the histidine in the media resulting in histidine biosynthesis. BN_{op_excess} also showed that many of the 9 key operons were influenced by the excess tryptophan condition as all of the key operons (except *trpR*) have between 2 to 6 (which includes *trpR*) of the 9 operons as a possible 2nd parent.

We were surprised by the results from BN_{op_starve}. BN_{op_starve} showed that known operons, *hybGFEDCBA*, *artPIQMJ*, *rplK-rplA*, *fecIRABCDE*, and predicted operons, *yciGFE*, *yafDE*, *yi21-yi22*, were influenced by all of the 9 key operons. Khodursky *et al.* [83] note that in their cluster analysis genes *yciF* and *yciG* form a tight cluster with *trpR* and related operons. They also noticed that arginine biosynthetic operons were sensitive to tryptophan changes. The *artPIQMJ* operon codes for proteins involved in the arginine transport system.

Are the other correlations meaningful? It turns out that the *rplK* ribosomal protein is involved in regulating the response to starvation for amino acids [162]. The *fecIRABCDE* operon is actually 2 distinct operons *fecABCDE* and *fecIR* (probably an error in the database of Salgado *et al.* [134]). *fecABCDE* is induced under iron limiting conditions or in the presence of ferric citrate and is under the control of the regulator FUR. We are not sure why it would be involved in the tryptophan starvation response, but *aroH* is known to be more active in the presence of iron [126]. The *hybGFEDCBA* operon encodes hydrogenase-2, which is usually used under conditions of anaerobiosis (low oxygen). We do not know of a reason why the culture conditions would lead to low oxygen levels, although constant vigorous shaking of the culture during growth is important for maintaining good aerobic growth conditions. The results could be related to unknown factors in the experimental methodology of Khodursky *et al.* [83] such as oxygen levels. *yci21* and *yci22* proteins are encoded by IS2, an insertion sequence in *E. coli*. These insertion sequences are mobile DNA elements that are often induced by growth of cells under stressful conditions. The roles of the *yafD* and *yafE* proteins are not clear as they are hypothetical proteins.

BN_{op_starve} also showed that *trpR* was identified to be correlated with almost all of the nine key operons under the tryptophan starvation condition. *mtr* and *trp* were also among the 20 most probable parents for two of the nine operons. No possible correlation was found for any of the other nine operons.

A summary of the results from the search of the 20 most probable parents in the DBN structure are listed in Table 2.1. Seven of the operons that were found in the 20 best parents would correctly model causality if they were selected as the second best parent to be added. The probability of at least one of the nine operons plus *tyrR* being in the top 20 best parents for each of the nine operons is quite low, at 0.059[5]. While further experimentation is required, these results provide some initial evidence supporting the use of time series data to learn causality.

[5]The probability of picking an operon that is not one of the nine key operons or *tyrR* (if all genes are equally likely) is $\frac{132}{141}$. The probability of picking all 20 operons that are not one of the 10 is $(\frac{132}{141})^{20} = 0.27$. Thus, the probability of getting at least one of the nine key operons is 1 - 0.27 = 0.73. However, the probability of doing this for all nine operons is $(0.73)^9 = 0.059$.

Table 2.1 Summary from the results of the DBN model. The operons listed on the right are one of the 9 key operons or *tyrR* that appeared as one of the 20 most probable parents of the operons on the left.

Key operons in tryptophan regulatory pathway	**Operons known to be involved in the tryptophan regulatory pathway that appeared as one of the 20 most probable parents of the operon on the left**
aroF-tyrA	**tyrR**, **trpR**
aroG	**aroF-tyrA**, **aroP**, **aroH**, **tyrR**
aroH	**tyrR**
aroL-yaiA-aroM	**trpR**, **tyrR**
aroP	**tyrR**
mtr	**tyrR**
tnaLAB	**aroF-tyrA**
trpLEDCBA	**aroH**, **tyrR**
trpR	**aroG**

2.5 Related Work

Methodologies such as clustering or correlation-based techniques [6, 10, 37] and probabilistic graphical models [4, 45, 61, 88, 164] have been used to learn various aspects of the cellular system such as inferring gene, metabolic and signaling networks. These techniques form hypotheses about the underlying mechanism based on patterns extracted from experimental data. Whereas Arkin *et al.* [9], Palsson [113], You *et al.* [165], Schoeberl *et al.* [136] and Yamada *et al.* [161] have instead used simulation to exemplify what is known of the dynamics of the system in order to test the validity of underlying assumptions and interpret behaviors observed in cell cultures.

One of the earliest and still widely used approach is clustering or correlation-based techniques [6, 10, 37, 71]. Extensions have included identifying clusters with transcription factor binding sites to further associate groups of functionally related genes [131]. Ideker *et al.* [71] used prior knowledge of a specific metabolic pathway as well as mRNA and protein abundance measurements of genetically and environmentally perturbed yeast to compensate for some of the shortcomings of correlation-based techniques. Stuart *et al.* [153] presented a way to cluster genes across different organisms utilizing evolutionary conservation to identify gene interactions.

Probabilistic graphical models are an intuitive and now widely used approach to represent regulatory relationships. Theoretical and experimental results have been proven about Boolean networks (BoolNs) by Akutsu *et al.* [3–5], Shmulevich *et al.* [143, 144] and Lahdesmaki *et al.* [88]. Bayesian networks [45, 61, 117] and its variants, module networks [138, 139] and hierarchical BN [59], dynamic Bayesian networks [70, 84, 108, 118], relational Markov networks (RMNs) [141] and probabilistic relational models (PRMs) [140] all concisely describe relationships and probability distributions over the observations. Noto and Craven [106] have used prior biological knowledge to structure the BN so that complex mechanisms may be inferred. Segal *et al.* [137, 139–141] have used RMNs, PRMs and module networks to model expression data and simultaneously (jointly) discover patterns in multiple types of data.

The shift towards systems biology [71] and the use of multiple data sources [73, 141, 150] have come about because current theories still have various shortcomings in providing a predictive

description of the cellular system as a whole. Learning network motifs [99, 142, 163] has also been important for understanding the underlying mechanisms [160].

Learning Bayesian networks is a very difficult problem and has been shown to be NP-complete [23]. However, this has not deterred researchers as there has been a great deal of research focused on learning the structure of BNs [1, 23, 41, 42, 44, 62, 63]. The most common approach to discovering structure is to use learning with model selection to find a high-scoring model [64]. However, in many domains there are exponentially many structures that have similarly good scores given the data, especially if the data is small relative to the size of the model. The sparse candidate algorithm [43] is a specific method for structure learning from sparse datasets.

2.6 Chapter Summary

This chapter presents the first application (to our knowledge) of Dynamic Bayesian networks to time series gene expression microarray data. It also shows how background knowledge about an organism's genome (in this case, an operon map) can be used to construct the initial, core structure of the DBN. This background knowledge can be taken from the scientific literature or can itself be the output of another modeling system. In this case, the operon map consisted partially of each type of knowledge. We also provided some evidence that the results of such an application of DBNs can give additional insights into the organism's regulatory network and that such an application has the potential to reveal hierarchical connections between signal transduction pathways. We further demonstrate that our DBN approach has the potential for inducing direct causal links, that is, direct arcs in the regulatory network. The approach proposed by Pe'er *et al.* [116] can be used in conjunction with our approach if knockout experiments are available. Further experiments will provide insight into whether causality can really be learned from time series data.

This work was originally published in the journal Bioinformatics in 2002 [108].

Chapter 3

Computational Model to Guide the Design of Time Series Experiments

Time series data, and the use of dynamic Bayesian networks (DBNs) as a method for analysis are becoming more widely used as an approach to learn gene regulatory networks [70,84,108,118]. A common question that have been asked regarding the collection of future time series experiments for DBN analysis is: given fixed resources to run r microarrays, is it better to run many short time series or a few long time series? Another question regarding a design issue for our learning algorithms is: given a specific number of microarrays r that will be run, and a given amount of time in which a DBN must be produced from this data, should we place a limit on the number of parents a node can have in the DBN and, if so, what should this limit be? One way to answer these questions is to perform many runs with many time series data sets having different properties; unfortunately, at present few such data sets are available, and the cost of producing such a data set requires design insight now, before additional data sets are available. Thus, we chose to construct a formal model of the learning task, making the model as realistic as possible though necessarily making some simplifying assumptions to gain insight to this problem.

3.1 Related Work

We limit our focus to DBNs whose variables are Boolean and thus can be viewed as deterministic or probabilistic Boolean networks; however, our results extend naturally to non-Boolean discrete variables. Dasgupta [28] proved the sample complexity of learning fixed-structure BNs and early work by Akutsu *et al.* [3] formalized the task of constructing Boolean networks from gene

expression microarray data. Further papers [4, 5, 29, 88, 143, 144] extended or improved the initial results of Akutsu *et al.* [3]. Several of those papers provide formal results on the task of finding a consistent or best-fit Boolean network for the data. Nevertheless, the results do not give guarantees about the accuracy of the learned network on new or unseen data, or the amount of data required to achieve a given level of accuracy. We propose to address the question of polynomial-time learnability using the PAC-learning framework [159] and its extension to probabilistic concepts [79]. As we propose to use a PAC-framework, its provision of (probabilistic) accuracy guarantees are based on data set size.

DBNs have hidden variables and can be represented as *hidden Markov models* (HMMs) [105]. Abe and Warmuth [2] have found that fixed state size *probabilistic automata* (PAs), which are closely related to HMMs[1] (HMMs), are trainable in polynomial time. Ron *et al.* [130] further proved results on learning PAs with variable memory length using *probabilistic finite suffix automata* (PFSAs).

Kearns and Li [80], Angluin and Laird [8], Goldman [57] and Decatur and Gennaro [34] have presented models for learning from noisy or incomplete data. Several papers on learning models using prior knowledge have been put forth by Pitt [119] and Campi and Vidyasagar [21]. Angluin [7] also presented models on *queries and concept learning*.

3.2 Definitions and terminology

Definition 3.2.1. *A Boolean* dynamic Bayesian network *(DBN) is defined over the Boolean variables*

$$X_{1,1}, X_{2,1}, \ldots, X_{n,1},$$
$$X_{1,2}, X_{2,2}, \ldots, X_{n,2},$$
$$\ldots,$$
$$X_{1,T}, X_{2,T}, \ldots, X_{n,T}$$

[1] HMMs are similar to PAs, except that outputs in HMM are associated with the states rather than the transitions, and thus transitions are unlabeled state to state pairs [2].

where $X_{i,t}$ denotes variable X_i at time t. For each $1 \leq i \leq n$ and $1 < t \leq T$ the value of variable $X_{i,t}$ is $f_i(X_{1,t-1}, \ldots, X_{n,t-1})$, where f_i is some (possibly stochastic) Boolean function.

Definition 3.2.2. *We denote by $\mathcal{DBN}(\mathcal{C}_n)$ the class of Boolean DBNs for which each function f_i comes from Boolean concept class $\mathcal{C}_n$.*

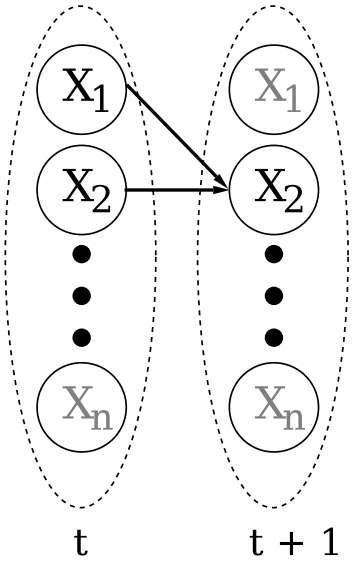
Any particular Boolean DBN in $\mathcal{DBN}(\mathcal{C}_n)$ is a set of functions $f_i(X_{1,t-1}, \ldots, X_{n,t-1})$, one for each variable X_i $1 \leq i \leq n$. Note that the function f_i does not change with time. Note also that in practice, a DBN model may contain some hidden variables, however, we presently do not consider these or missing data in our analyses.

For example, the Boolean concept class $\mathcal{C}_n$ might be all of the stochastic functions of at most k variables. This class of functions corresponds to all possible CPTs in a DBN for a node with at most k parents. An example of such a CPT is given in Figure 3.1. Or if the DBN is in fact deterministic, $\mathcal{C}_n$ might be the set of all (non-stochastic) functions of at most k variables, that is, all truth tables over k variables. For such a CPT, each row in Figure 3.1 would instead have one of the probabilities set to 1 and the other set to 0. A generalization of this class, allowing more than k parents in a still limited fashion would be to have as $\mathcal{C}_n$ the set of all functions that can be represented by a k disjunctive normal form (kDNF) expression. The set of kDNF expressions is the set of all DNF expressions where each disjunct (conjunction) contains at most k literals.

3.3 Boolean DBN from 2-slice data

The cost of obtaining r microarray experiments (measuring the expression of each gene in r samples of mRNA) is roughly the same regardless of whether the r samples are all part of a single, long time series or many different time series. Therefore, we treat our number of data points as the number of microarray experiments rather than the number of time series.

In the ordinary PAC-learning model, one assumes each data point is drawn randomly and independently according to some probability distribution D. Our models cannot assume this, because in a time series each data point (after the first) depends on the previous data point. The most faithful we can remain to the original PAC-learning model is to specify that the first data point in each



X_1	X_2	$P(X_{2,t+1})$	
		1	0
0	0	0.15	0.85
0	1	0.35	0.65
1	0	0.70	0.30
1	1	0.60	0.40

Figure 3.1 Example of probabilistic CPTs as part of a DBN.

time series is drawn randomly according to some probability distribution D, and that the first data points in different time series are drawn independently of one another.

We begin with a formal model of DBN learning that resembles the PAC-learning model as much as possible, by restricting consideration to deterministic concepts. We later extend the definition to permit probabilistic concepts. Given a deterministic DBN and a specific (input) time slice, the next (output) time slice is fixed according to the DBN. A DBN model and a target DBN disagree with one another on an input time slice if and only if, given the input time slice, the two DBNs produce different outputs. A DBN model is $(1-\epsilon)$-accurate with respect to a target model if and only if the sum of the probabilities, according to D, of input time slices on which the two DBNs disagree is at most ϵ. As is standard with the PAC model, we take $|T|$ to denote the size of the target concept (DBN model) $T \in \mathcal{DBN}(\mathcal{C}_n)$ in a "reasonable" encoding scheme. For concreteness, we specify $|T|$ as the number of bits required to encode, for each variable X_i, its parents and its function f_i. Given these preliminaries, the following definition is an application of the PAC-learning model to DBNs.

Definition 3.3.1. *An algorithm PAC-learns a deterministic $\mathcal{DBN}(\mathcal{C}_n)$ if and only if there exist polynomials poly$_1$(_, _, _, _) and poly$_2$(_) such that for any target DBN T in $\mathcal{DBN}(\mathcal{C}_n)$, any $0 < \epsilon < 1$ and $0 < \delta < 1$, and any probability distribution D over initial data points for time series given any $r \geq$ poly$_1(n, |T|, \frac{1}{\epsilon}, \frac{1}{\delta})$ data points, the algorithm runs in time poly$_2(rn)$ and with probability at least $1 - \delta$ outputs a model that is $(1 - \epsilon)$-accurate with respect to T.*

Theorem 3.3.2. *For any fixed $k \in \mathbb{N}$ the class of $\mathcal{DBN}(k$DNF$)$ is PAC-learnable from 2-slice data.*

Proof. Our algorithm A learns one kDNF formula to predict each of the n variables at time slice 2 from the values of the n variables at time slice 1. Each 2-slice time series (input and output) is used to generate one example for each $X_{i,2}$. For each $1 \leq i \leq n$ the output (class) is $X_{i,2}$ and input features are $X_{1,1}, \ldots, X_{n,1}$. Given a PAC learning algorithm L for kDNF expressions [81], we can run L on n feature vectors to find a concept in $\mathcal{C}_n$ that is consistent with our data.

Algorithm A iterates as follows: for each variable $X_{i,2}$, $1 \leq i \leq n$, we make a call to kDNF learning algorithm L with $\frac{\delta}{n}$ as the maximum probability of failure (i.e., with desired confidence of $1 - \frac{\delta}{n}$) and with $\frac{\epsilon}{n}$ as the maximum error (i.e., with desired accuracy of $1 - \frac{\epsilon}{n}$). Algorithm A's final model is the set of functions $f_i(X_{1,1}, \ldots, X_{n,1})$ returned by L, one per output variable $X_{i,2}$.

Algorithm A runs in polynomial time since $n*\text{poly}_1(n,|T|,\frac{n}{\epsilon},\frac{n}{\delta})$ yields a polynomial, and each call to L runs in time polynomial in the size of its input. It remains only to show that with probability $1-\delta$ the error is bounded by ϵ. The definition of union bound states that if A and B are any two events (that is, subsets of a probability space), then $\Pr(A \cup B) \leq \Pr(A)+\Pr(B)$ [82]. Since each call to L fails to achieve the desired accuracy with probability only $\frac{\delta}{n}$, by the union bound the probability that there exists *any* of the n calls to L that fails to achieve the desired accuracy is at most δ. If each call to L has a desired error bound of $\frac{\epsilon}{n}$, then the error of the model (probability according to D of drawing an input time slice on which the learned model and target will disagree for some variable $X_{i,2}$, $1 \leq i \leq n$) is the union of all n error expressions from L. By the definition of $\mathcal{DBN}(\mathcal{C}_n)$ this gives us $\Pr(Error_1 \cup Error_2 \cup \ldots \cup Error_n) = \Pr(Error_1)+\Pr(Error_2)+\ldots+\Pr(Error_n) = \frac{\epsilon}{n} + \frac{\epsilon}{n} + \ldots + \frac{\epsilon}{n} \leq \epsilon$, by the union bound. □

kDNF is a richer representation than one usually uses in a DBN. Typically, each variable is a function (represented as a CPT) of up to k parents. We denote the class of such DBNs by $\mathcal{DBN}$(k-parents). While PAC-learnability of a more restricted class does not automatically follow from PAC-learnability of a more general class (because the space of allowed hypotheses is smaller), in this case arguments very similar to those just given show that, for any fixed $k \in \mathbb{N}$, the class of deterministic $\mathcal{DBN}$(k-parents) is PAC-learnable from 2-slice data.

3.4 Boolean DBN from r-slice data

It is equally common in practice for time series measurements to yield one long time series instead of multiple time series of length 2, or to fall between these two extremes. To gain some theoretical insight into whether a single, long time series might be more useful than many short time series, we now ask whether the class $\mathcal{DBN}$(k-parents) is PAC-learnable from a single time

series, and if so, whether the total number of microarrays required might be fewer. Unfortunately, it is trivial to prove that no algorithm PAC-learns this class when all the data points are in a single time series; the algorithm simply cannot learn enough about the distribution D according to which the start of each time series is drawn. But such a trivial negative result in unsatisfying. In practice, if we subject an organism to an experimental condition and run a long time series of microarray expression measurements, it is because we wish to learn an accurate model of how the organism responds to *that particular condition*. Therefore, we next consider a natural variant of our first learning model, where this variant is tailored to data points in a single time series. A positive result for single time series will be easier to obtain in this variant than in the original model.

Definition 3.4.1. *An algorithm learns a deterministic class $\mathcal{DBN}(\mathcal{C}_n)$ from a* single time series *if and only if there exist polynomials poly$_1$(_, _, _, _) and poly$_2$(_) such that for any target DBN T in $\mathcal{DBN}(\mathcal{C}_n)$, any $0 < \epsilon < 1$ and $0 < \delta < 1$, and any starting point for the time series given a time series of any length $r \geq$ poly$_1$(n,$|T|$,$\frac{1}{\epsilon}$,$\frac{1}{\delta}$), the algorithm runs in time poly$_2$(rn) and with probability at least $1 - \delta$ outputs a model that with probability at least $(1 - \epsilon)$ correctly predicts time slice $r + 1$.*

Notice that we do not require that the learning algorithm is capable of performing well for most starting points, but only for the one given. For deterministic DBN models, which are all we are considering thus far, after some m time slices the time series must return to a previous state, from which point on the time series will cycle with some period length at most m. If for some class of DBN models m is only polynomial in the number of variables n then it will be possible to PAC-learn this class of models from a single time series, within the definition just given. Unfortunately, even for the simple class of deterministic $\mathcal{DBN}(k$-parents), the period is superpolynomial in n and the size of the target model, leading to the following negative result.

Theorem 3.4.2. *For any $k \geq 2$ the class of $\mathcal{DBN}$(k-parents) is not learnable from a single time series.*

Proof. Assume there exists a learning algorithm L for $\mathcal{DBN}(k$-parents). Then for any k-parent target DBN T, any $0 < \epsilon < 1$ and any $0 < \delta < 1$, given a time series of any length $r \geq$

poly$_1(n,|T|,\frac{1}{\epsilon},\frac{1}{\delta})$, L will run in time polynomial in the size of its input and with probability at least $1-\delta$ will output a model that will correctly predict time slice $r+1$ with probability at least $1-\epsilon$. Because any 2-parent DBN can be represented in a number of bits that is polynomial in n, we can simplify poly$_1(n,|T|,\frac{1}{\epsilon},\frac{1}{\delta})$ to poly$_1(n,\frac{1}{\epsilon},\frac{1}{\delta})$.

We consider a time series that starts from a point in which every variable is set to 0. In this case, Lemma 3.4.3 shows that for a suitable choice of n (any n such that $n-1$ is divisible by 3) we can build two 2-parent deterministic DBNs T_1 and T_2 over variables $X_1, ..., X_n$ with the following properties when started from a time slice with variables set to 0: in both T_1 and T_2, X_n remains 0 for $r \geq 2^{\frac{n-1}{3}}$ steps and then X_n goes to 1 at step $r+1$; in T_1 once X_n goes to 1 it remains 1; in T_2 when X_n goes to 1 it then reverts to 0 on the next step.

We choose $\epsilon = \delta = \frac{1}{4}$ and large enough n such that $2^{\frac{n-1}{3}} >$ poly$_1(n,\frac{1}{\epsilon},\frac{1}{\delta})$. We present the algorithm L with a time series generated by T_1, of length r as specified in the previous paragraph, starting from the time slice in which all variables are set to 0. Then L must, with probability at least $\frac{3}{4}$, return a model that will correctly predict time slice $r+1$. Therefore, with probability at least $(\frac{3}{4})(\frac{3}{4}) > \frac{1}{2}$, L's output model predicts the value of X_n to be 1. Consider what happens when we give L exactly the same learning task, except that the target is T_2 instead of T_1. The time series of length r that L sees is identical to the previous one, so L will with probability greater than $\frac{1}{2}$ incorrectly predict the value of X_n at time slice $r+1$. Hence L will *not* produce, with probability at least $1-\delta$, a model that will predict time slice $r+1$ with accuracy at least $1-\epsilon$.

With this construction we can generate a cycle that looks like the cycle in Figure 3.2. □

Lemma 3.4.3. *There exists a 2-parent deterministic DBN over j variables with a period of $2^{\frac{j}{3}}$, for any positive integer j divisible by 3. Moreover, based on our specific construction we can build two 2-parent deterministic DBNs T_1 and T_2 over any variables $X_1, ..., X_n$, $n = j+1$ for some j divisible by 3, with the following properties when started from a time slice with all variables set to 0. In both T_1 and T_2, X_n remains 0 for $r \geq 2^{\frac{n-1}{3}}$ steps and then X_n goes to 1 at step $r+1$; in T_1 once X_n goes to 1 it remains 1; in T_2 when X_n goes to 1 it then reverts to 0 on the next step.*

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	time
1	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	2
1	0	1	0	0	0	0	0	0	0	0	0	3
0	1	0	1	0	0	0	0	0	0	0	0	4
1	0	1	1	1	0	0	0	0	0	0	0	5
0	1	0	0	1	0	0	0	0	0	0	0	6
1	0	1	0	0	1	0	0	0	0	0	0	7
0	1	0	1	0	0	1	0	0	0	0	0	8
1	0	1	1	1	0	1	1	0	0	0	0	9
0	1	0	0	1	0	1	1	0	0	0	0	10
1	0	1	0	0	1	1	1	0	0	0	0	11
0	1	0	1	0	0	0	1	0	0	0	0	12
1	0	1	1	1	0	0	0	1	0	0	0	13
0	1	0	0	1	0	0	0	0	1	0	0	14
1	0	1	0	0	1	0	0	0	1	1	0	15
0	1	0	1	0	0	1	0	0	1	1	0	16
1	0	1	1	1	0	1	1	0	1	1	0	17
0	1	0	0	1	0	1	1	0	1	1	0	18
1	0	1	0	0	1	1	1	0	1	1	0	19
0	1	0	1	0	0	0	1	0	1	1	0	20
1	0	1	1	1	0	0	0	1	1	1	0	21
0	1	0	0	1	0	0	0	0	0	1	0	22
1	0	1	0	0	1	0	0	0	0	0	1	23
0	1	0	1	0	0	1	0	0	0	0	0	24
1	0	1	1	1	0	1	1	0	0	0	0	25
0	1	0	0	1	0	1	1	0	0	0	0	26
1	0	1	0	0	1	1	1	0	0	0	0	27
0	1	0	1	0	0	0	1	0	0	0	0	28
1	0	1	1	1	0	0	0	1	0	0	0	29
0	1	0	0	1	0	0	0	0	1	0	0	30
1	0	1	0	0	1	0	0	0	1	1	0	31
0	1	0	1	0	0	1	0	0	1	1	0	32
1	0	1	1	1	0	1	1	0	1	1	0	33
0	1	0	0	1	0	1	1	0	1	1	0	34
1	0	1	0	0	1	1	1	0	1	1	0	35
0	1	0	1	0	0	0	1	0	1	1	0	36
1	0	1	1	1	0	0	0	1	1	1	0	37
0	1	0	0	1	0	0	0	0	0	1	0	38
1	0	1	0	0	1	0	0	0	0	0	1	39

Figure 3.2 Cycle with period $2^{\frac{n}{3}}$

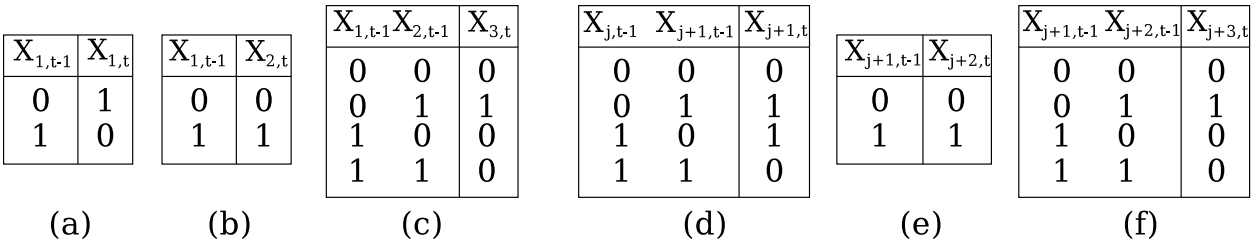


Figure 3.3 Base case construction (a,b and c) and inductive case construction (d,e and f) of inductive proof of Lemma 3.4.3. Construction includes bit counter (a,d), previous bit memory (b,e) and 0-to-1 flag (c,f).

Proof. We begin with the first statement of the lemma. If we were allowed up to 3 parents per variable, we could represent a two-input (e.g., R-S or J-K) flip-flop (one parent for current state and two for inputs, to determine new state) with a single variable. We could then directly implement a standard up-counter (binary counter is one which counts binary sequence) from computer architecture with just two variables per bit. In our proof we still essentially implement an up-counter. But because we want to show a superpolynomial period can be obtained with just two parents per variable, we require three variables per bit of the up-counter.

Inductive hypothesis: Given j variables, a deterministic DBN exists with a period of $2^{\frac{j}{3}}$.

Base case: When $j = 3$, the proof is trivial as this can be done with 1 bit. Nevertheless, we employ the construction in Figure 3.3a to 3.3c so that every set of 3 variables is analogous to the others. The first variable is a 1-bit counter. The second variable, $X_{2,t}$, serves as a memory of the previous bit at the previous time slice, $X_{1,t-1}$. $X_{3,t}$ works as a flag, taking on the value 1 to indicate when X_1 has made the transition from 0 back to 1 on the previous time slice.

Inductive case: We prove that given $n = j + 3$ variables, a period doubling the period of $2^{\frac{j}{3}}$ can be generated. We first generate a counter that is based on the flag, $X_{j,t-1}$. If $X_{j,t-1}$ is 0, then $X_{j+1,t}$ takes on the value of $X_{j+1,t-1}$. Otherwise, when $X_{j,t-1}$ is 1, $X_{j+1,t}$ acts as a bit counter. $X_{j+2,t}$, serves as a memory of the previous bit at the previous time slice, $X_{j+1,t-1}$, mimicking its value in the previous time slice. Finally, $X_{j+3,t}$ works as a flag, taking on the value 1, to indicate when $X_{j+1,t-1}$ is 0 and $X_{j+2,t-1}$ is 1; and $X_{j+3,t}$ is 0 otherwise.

For the second statement of the lemma, m is equal to $j + 1$, where j is divisible by 3. We construct a network as in the proof of the first statement of the lemma, such that X_j is 0 for the first $r \geq 2^{\frac{n-1}{3}}$ time steps, then becomes 1, and then immediately reverts to 0; that is, X_j is 1 just once every $2^{\frac{n}{3}}$ time steps. For T_1 we have a function for X_m such that $X_{m,t}$ is 1 if and only if $X_{j,t-1}$ (that is, $X_{m-1,t-1}$) is 1 *or* $X_{m,t-1}$ is 1. For T_2 we have a function for X_m such that $X_{m,t}$ is 1 if and only $X_{j,t-1}$ (that is, $X_{m-1,t-1}$) is 1.

□

3.5 Stochastic model of Boolean DBN from 2-slice data

In our first model of learning Boolean DBN models from 2-slice data, we made a simplifying assumption that the DBN models were deterministic, in keeping with the standard assumption for target concepts in the PAC model. In reality, DBNs are probabilistic models, with deterministic DBNs as simply a special case. When we learn a Boolean DBN model, we are not only interested in learning the correct Boolean functions but also inferring a good model of probability with respect to the target distribution. We therefore now extend our theoretical framework and to bring our model closer to practice. The foundation of this extension consists of the notions of *p-concept* (probabilistic concept) and (ϵ, γ)-*good models of probability*, defined by Kearns and Schapire [79]. In these definitions X is the domain of possible examples and D is a probability distribution over X.

Definition 3.5.1. *A* probabilistic concept (p-concept) *is a real-valued function* $c : X \rightarrow [0, 1]$. *When learning the p-concept* c*, the value* $c(x)$ *is interpreted as the probability that* x *exemplifies the concept being learned. A p-concept class* $\mathcal{C}_p$ *is a family of p-concepts. A learning algorithm for* $\mathcal{C}_p$ *attempts to learn a distinguished target p-concept* $c \in \mathcal{C}_p$ *with respect to a fixed but unknown and arbitrary target distribution* $\mathcal{D}$ *over the instance space* X *[79].*

Given this definition, it is easy to see that a function f_i in a (not necessarily deterministic) Boolean DBN, which gives the probability distribution over possible values for X_i at time $t + 1$

conditional on the values of the n variables at time t, is a p-concept. Therefore, a Boolean DBN as defined earlier is completely specified by a set of n p-concepts, one for each variable.

Definition 3.5.2. *A p-concept h is an* (ϵ, γ)-good model of probability *of a target p-concept c with respect to D if and only if* $\mathbf{Pr}_{x \in D}[|h(x) - c(x)| > \gamma] \leq \epsilon$.

We generalize this definition to apply to DBNs as follows. Given an input time slice, a DBN model defines a probability distribution over output time slices. We say that two DBNs M and T γ-disagree on an input time slice if they disagree by more than γ on the probability of an output time slice given that input. A learned DBN M is an (ϵ, γ)-*good model of probability* of a target DBN T with respect to a probability distribution D over input time slices if and only if the sum of the probabilities of input models on which M and T γ-disagree is at most ϵ.

The learning model we present next is a straightforward application of Kearns and Schapire's notion of *polynomially learnable with a model of probability* to DBNs, analogous to our earlier application of the PAC model to deterministic DBNs. In the following definition we take $\mathcal{C}_n$ to be any p-concept class. Thus for example $\mathcal{DBN}$(k-parents) is the set of DBNs in which each variable has at most k-parents; the p-concept class used here is the class of p-concepts of k-relevant variables, or the class of p-concepts representable by CPTs conditional on k parents.

Definition 3.5.3. *Where $\mathcal{C}_n$ is a p-concept class, we say that an algorithm learns $\mathcal{DBN}(\mathcal{C}_n)$ with a model of probability if and only if there exist polynomials poly$_1$(_, _, _, _, _) and poly$_2$(_) such that for any target concept $T \in \mathcal{DBN}(\mathcal{C}_n)$, any $0 < \epsilon < 1$, $0 < \delta < 1$, and $0 < \gamma < 1$, and any probability distribution D over initial data points for time series: given $r \geq$ poly$_2(n, |T|, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{\gamma})$ data points, the algorithm runs in time poly$_2(rn)$ and with probability at least $1 - \delta$ outputs an (ϵ, γ)-good model of probability of the target.*

Theorem 3.5.4. *For any fixed $k \in \mathbb{N}$ the class $\mathcal{DBN}$(k-parents) is learnable with a model of probability from 2-slice time series data.*

Proof. We describe a learning algorithm B that is analogous to the algorithm A of Theorem 3.3.2, and the correctness proof is analogous as well. Each 2-slice time series (input and output) is used

to generate one example for each $X_{i,2}$. For each $1 \leq i \leq n$ the output (class) is $X_{i,2}$ and input features are $X_{1,1}, \ldots, X_{n,1}$. In place of the kDNF learning algorithm used by the earlier algorithm A, algorithm B uses an algorithm P that with probability $1 - \delta$ learns an (ϵ, γ)-good model of probability for any p-concept with at most k relevant variables [79]. The learned p-concept for variable X_i can be expressed as a set of at most k parents for variable X_i and a CPT for variable X_i given those parents.

More specifically, where δ, ϵ and γ are the parameters provided to algorithm B, algorithm B calls algorithm P using instead $\frac{\delta}{n}$, $\frac{\epsilon}{n}$ and $\frac{\gamma}{2n}$. Algorithm B iterates as follows: for each variable $X_{i,2}$, $1 \leq i \leq n$, algorithm B makes a call to algorithm P with the examples and parameters as specified. Algorithm B's final model of probability for each X_i, $1 \leq i \leq n$, is Pr($X_{i,t}|X_{1,t-1}, \ldots, X_{n,t-1}$) = Pr($X_{i,t}|$Pa($X_i$)$_{t-1}$), where Pa($X_i$)$_{t-1}$ denotes the (at most k) parents of X_i from the previous time step, as determined by algorithm P, and Pr($X_{i,t}|$Pa(X_i)$_{t-1}$) denotes the specific function (representable as a CPT) learned by algorithm P.

Algorithm B runs in polynomial time since $n*$poly$_1(n,|T|,\frac{n}{\epsilon},\frac{n}{\delta},\frac{2n}{\gamma})$ yields a polynomial, and each call to P runs in time polynomial in the size of its input. The remainder of the reasoning is analogous to that in the proof of Theorem 3.3.2, except that we must also note the following. If the learned DBN and target DBN agree within $\frac{\gamma}{2n}$ on the probability for a given setting for each variable X_i, $1 \leq i \leq n$, then they agree within γ on the probability of the entire setting. It follows that since *for any given variable* X_i the learned DBN with probability $1 - \frac{\delta}{n}$ has an $(\frac{\epsilon}{n}, \frac{\gamma}{2n})$-good model of probability compared with the target DBN, then with probability $1 - \delta$ the learned DBN is an (ϵ, γ)-*good model of probability* of the target DBN. □

We can also extend our model of learning from a *single time series* to apply to probabilistic concepts in a similar fashion. But since deterministic DBNs are a special case of probabilistic ones, it follows that the result for learning $\mathcal{DBN}$(2-parents) with a model of probability, from a single, long time series is a negative one.

3.6 Experiments using Synthetic Data

We used a kinetic model that modeled the EGF-receptor system using classic enzyme kinetics to generate synthetic data. This simulator, known as RAKS [96], can generate short and long time series data according to the Ras-signaling pathways model. There are 22 variables, each with 0 to 3 parents per variable in the target model. We used the generated data to try to recover the target model using the REVEAL [90] algorithm. Then we compared the learned models to the known target model as the number of samples, n, grows (see learning curves in Figure 3.4 and Figure 3.5). We count missing parents or incorrect assignment of parents as errors.

3.7 Lessons and limitations

The results proven show that, for natural definitions of learnability, DBNs are learnable from 2-slice time series data but not from a single, long time series. If we adopt a compromise, with k-slice time series for fixed k greater than two, we can again get positive results but the total number of time slices, e.g., microarrays to be run, increases linearly with k. Hence the results in this chapter imply that while time series are desirable, they should be kept as short as possible.

Because PAC bounds are worst-case, the number of examples they indicate are required, while polynomial in the relevant parameters, can be much greater than typically are required in reality. Nevertheless, we can gain some insight from these numbers into which factors most affect sample size required for a given degree of accuracy. The sample sizes required by the algorithms in this chapter follow directly from those required by the learning algorithms they employ as subroutines. The sample sizes for those algorithms grow linearly with the number of variables n, the target concept size, and $\frac{1}{\epsilon}$ (and $\frac{1}{\gamma}$ where relevant), and logarithmically with $\frac{1}{\delta}$. But note that the sizes of our target concepts in $\mathcal{DBN}$(kDNF) and $\mathcal{DBN}$(k-parents) are at least $O(n^k n)$, because we must specify the choice of k out of n possible parents for each of n variables. Therefore, by far the most important factor in sample size is k, and the next most important is n. Because current costs limit microarray data set sizes to around one thousand microarrays (in fact, we currently know of no data sets quite that large), a value of three for k seems the largest reasonable value, with $k = 2$

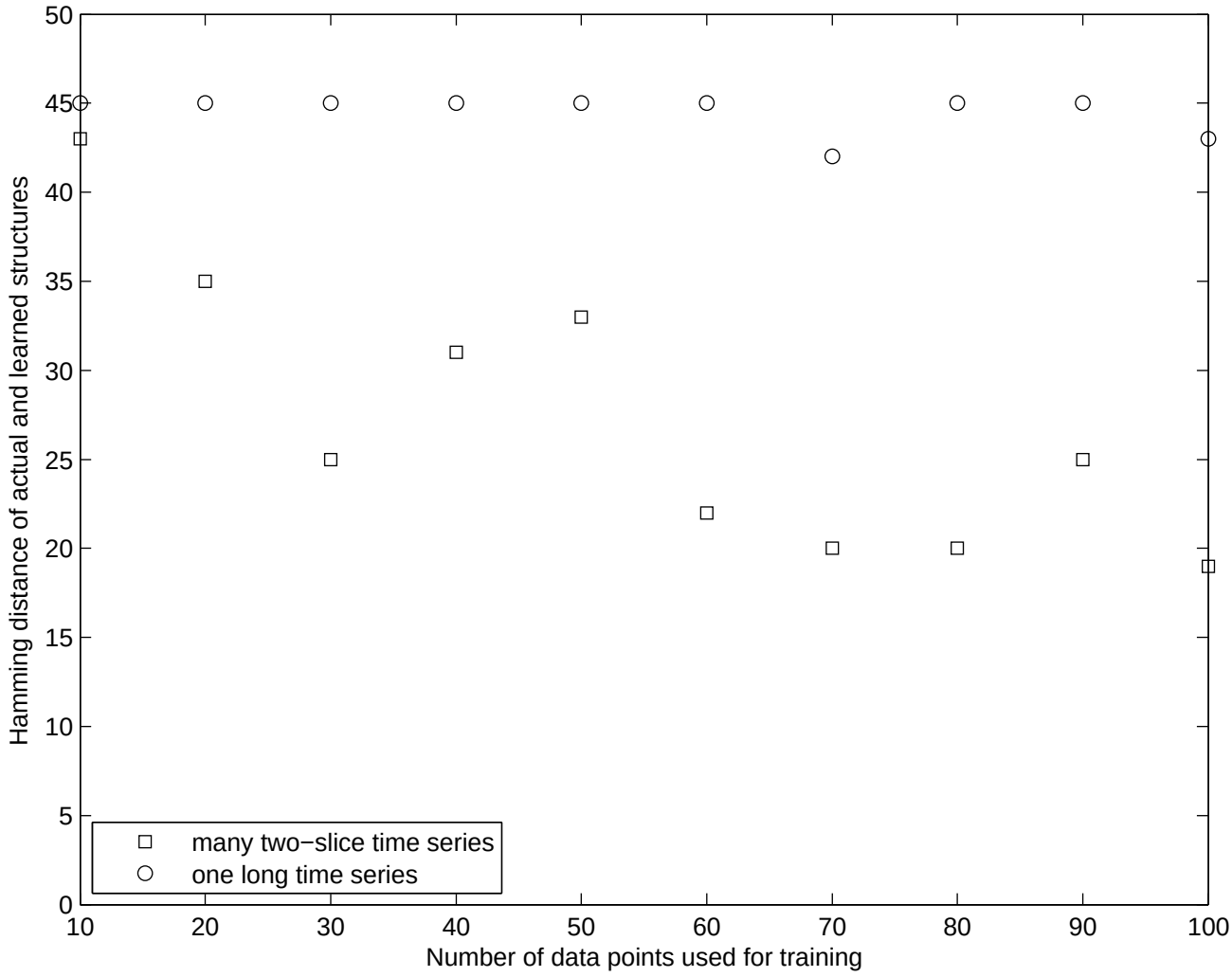


Figure 3.4 Errors as the number of samples, n varies: low n

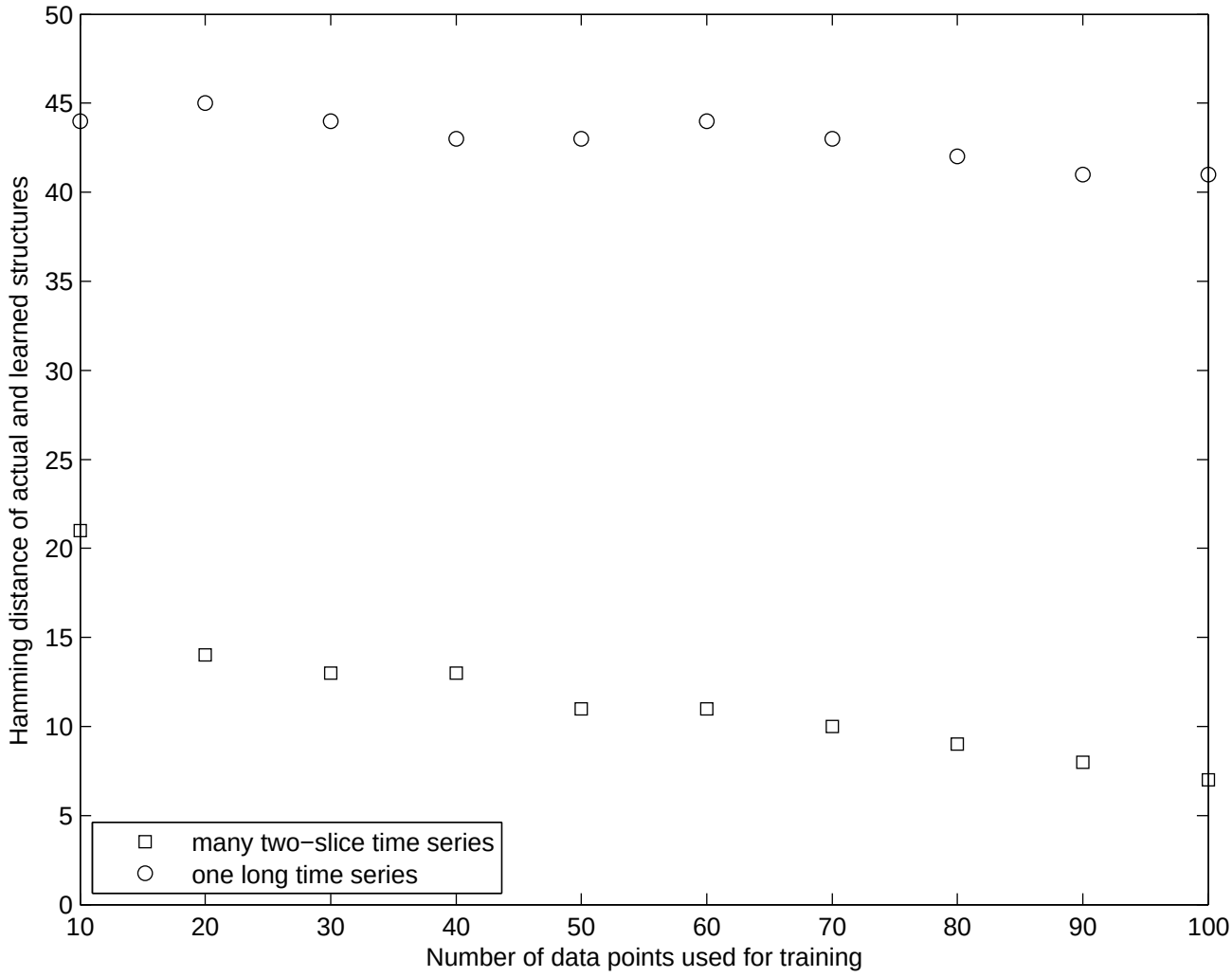


Figure 3.5 Errors as the number of samples, n varies: high n

probably more sensible. The size of the target concept can be further contained if we limit our models to trying to predict only a small subset of variables in terms of another small subset, based on background knowledge about particular regulatory pathways in which we are most interested.

The learning models defined are assumed to be reasonable as they are a natural application of existing PAC models to DBN learning. Nevertheless, several assumptions are inherited in this application—some from PAC modeling and some from DBNs—and several additional assumptions have been made.

Inherent to the use of PAC modeling are the assumptions that (1) we must perform well on *all* target concepts, and (2) examples are drawn randomly, independently according to some probability distribution. Regarding assumption (1), numerous regulatory motifs have been identified to date, including logic gates and memory elements [95,125,160], giving credence to the simple DBN representation of difficult target concepts such as counters. For some real biological pathways that have very short periods, perhaps single, long time series will be more effective than our results imply. Regarding assumption (2), it seems plausible that an organism's environment imposes some probability distribution over states in which its regulatory machinery may find itself, and to which it will need to respond. Nevertheless, perhaps through careful selection of experimental conditions, active learning approaches may arise that will benefit more from a few long time series than from many short ones.

Inherent in the use of DBNs are several notable assumptions as well. First, the DBN framework assumes we are modeling a *stationary process*; while the state of an organism is not static (e.g., mRNA levels may change over time), the organism's regulatory network itself does not change over time. This assumption appears reasonable for the application to learning regulatory pathways. But more specific assumptions include the assumption of discrete time steps—that an organism, like a computer, operates on a fixed clock that governs when changes occur—and a first-order Markov assumption, that the organism's next state is a function of only its previous state inputs. These assumptions clearly are violated to some extent, and those violations present caveats to our lessons. For example, perhaps collecting longer time series, with a very fast sampling rate, could allow our algorithms to try different sampling rates (by skipping some of the time steps), to find

optimal rates for providing insight into certain processes. Inappropriate sampling rates have been noted as a potential source of error when modeling time series microarray data [13]. Our results do not speak to uses such as this for longer time series.

We have made additional simplifying assumptions beyond those of the PAC framework or DBNs. Specifically, we have assumed all variables are Boolean and that there are no missing values or hidden variables. While discretization is common with microarray data, one may discretize to more than two values or may use the continuous values originally reported in a microarray experiment. We see no obvious reason why using such values should change the lessons in this chapter, but such a change is possible. Missing values are rare in microarray data, but one might wish to include hidden variables for unmeasured environmental factors or for other players in a regulatory pathway, such as certain proteins that may act as transcription factors or for other types of signaling.

This work was originally published in the Proceedings of the Pacific Symposium on Biocomputing in 2006 [110].

Chapter 4

Scaling FOIL for Multi-Relational Learning on Large Datasets

4.1 Introduction

There are many large biological databases available [78, 94, 133] with large amounts of information, including protein-DNA and protein-protein interactions. This information can be used to learn relational information regarding regulatory "motifs" [99, 142] especially since large portions of these networks appear to have evolved through extensive duplication of transcription factors even among genomes [12]. Rules regarding regulation could be learned directly from information already in these databases or combined with the results of an inferred regulatory network via a DBN. Inductive Logic Programming (ILP) [89] algorithms learn first-order logical rules from multi-relational data and thus are well suited to this task.

However, in order for ILP to learn from large datasets such as the above, two improvements to ILP must be made. First, ILP systems must deal with a limited amount of physical memory. Most ILP implementations, such as FOIL [121], tacitly assume the whole database fits into main memory. If it does not fit, these programs either crash or grind to an effective halt as they rely on the operating system to manage moving data between physical memory and disk [120]. Second, ILP systems need to be faster. The search space of ILP systems is very large and even heuristic methods are slow. Moreover, since the time needed to score an operator typically depends on the database size, direct application of most ILP systems to very large datasets is impractical.

4.2 Related Work

Interest in scaling up ILP for relational data mining on large datasets has been growing as numerous researchers [16,30,35,101,120,156] have published surveys and methods in this direction. The idea of incorporating ILP with RDBMSs is not new [14, 39], in fact Stonebraker and Kemnitz [151] developed a way to implement server-side logic that allows the application developer to modify the "query tree" of an incoming query, Fu and Han [48] used meta-rules as a guidance at finding multiple-level association rules in large relational databases, Lindner and Morik [91] and Brockhausen and Morik [18] have directly interfaced a relational data-mining algorithm, RDT, with RDBMSs and Lisi and Malerba [92] have shown that $\mathcal{AL}$-log is suitable for inducing multi-level association rules from multiple relations.

Hulten *et al.* [67, 68] have proposed methods for scaling-up any induction algorithm based on discrete search so that the running time becomes independent of the size of the database. DiMaio and Shavlik [35, 36] have also focused on speeding up scoring for candidate hypotheses.

Bryant *et al.* [19], Reiser *et al.* [127] and King *et al.* [85] have successfully been able to generate hypotheses about yeast gene functions from examples and background knowledge and use these hypotheses to select trials for actively learning the function of genes in yeast.

4.3 FOIL

FOIL [121,122] is a popular ILP algorithm that learns function-free first order rules for a target relation given a set of *extensionally* defined relations. By extensionally we mean each input relation is defined by a listing of its tuples rather than *intensionally* as a set of logical rules. This closely parallels the organization of data in relational databases [14]. We designate FOIL's *initial* positive and negative tuple sets as POS and NEG respectively. POS is the target relation and NEG is of the same arity as POS but contains tuples that do not belong to the target relation. The other relations $B1, ..., BN$ serve as background knowledge.

Given its input, FOIL learns rules of the form

$$POS(X_1, ..., X_m) \leftarrow L_1 \wedge L_2 \wedge ... \wedge L_r$$

where each L_i is a (possibly negated) literal and $X_1, ..., X_m$ are distinct variables. FOIL is a covering algorithm whose goal is to discover a set of rules that entails all of the tuples in POS and none of the tuples in NEG.

Table 4.1 presents the FOIL algorithm [100]. On each iteration of its outer loop, which starts at line 4[1] of the FOIL algorithm, adds a single clause to its learned rule set that logically entails (covers) some of the previously uncovered tuples in POS and none of the tuples in NEG. FOIL's method of building a single rule, shown in the LEARNONERULE procedure of Table 4.1, begins with the general rule $POS \leftarrow$ true, which covers all positive and negative tuples. This rule is then specialized by greedily conjoining literals one at a time to the body until the new rule covers no negative tuples. When deciding which literal to append to the body of the new rule, FOIL considers only the *current* positive and negative tuple sets, POS_{curr} and NEG_{curr} in Table 4.1. From one iteration to the next, these sets may shrink in size, grow in size, increase in arity or some combination thereof.

At the start of the LEARNONERULE procedure POS_{curr} is initialized to the tuples of POS that are not covered by any rule learned so far (POS_{unc}) and NEG_{curr} is set to all the tuples in NEG. Following the addition of the chosen literal (Line 17), FOIL updates POS_{curr} and NEG_{curr}. Let $L(N_1, ..., N_a, O_1, ..., O_b)$ be the chosen literal where N_i is a *new* variable that does not appear anywhere else in the new rule and O_i is an *old* variable that appears either in the head or a previously introduced literal. If L is unnegated, the arity of tuples in the updated POS_{curr} and NEG_{curr} sets increases by a, the number of new variables. A tuple t in POS_{curr} (or NEG_{curr}) gives rise to a (possibly expanded) tuple in the updated set for each tuple in the relation associated with the chosen literal that matches t on the arguments indicated by the old variables.

FOIL uses an information theoretic heuristic to determine which literal to append to the body of a growing rule. On each iteration of its inner loop, which starts on Line 15, FOIL chooses the literal that has maximum gain where it defines the gain of literal L_i as

$$GAIN(L_i) = p^{++} \times \left\{ \log(\frac{p'}{p'+n'}) - \log(\frac{p}{p+n}) \right\}.$$

[1]This and all subsequent references to line numbers refer to those in Table 4.1.

Table 4.1 The FOIL algorithm. Given a set of tuples, POS, in the target relation, a set of tuples, NEG, not in the target relation and sets for tuples $B_1, ..., B_M$ that define M background relations, FOIL returns a set of first-order rules for the target relation. The symbol $\Leftarrow$ indicates variable assignment and the symbol $\leftarrow$ indicates logical implication.

```
1: procedure FOIL(POS, NEG, B_1, ..., B_N )
2:   learnedRules ⇐ {}
3:   POS_unc ⇐ POS                          ▷ start with all positive tuples uncovered
4:   repeat                                  ▷ begin FOIL's outer loop
5:     newRule ⇐ LEARNONERULE()
6:     update POS_unc                        ▷ remove tuples in POS_unc covered by newRule
7:     add newRule to learnedRules
8:   until |POS_unc| == 0
9:   return learnedRules
10: end procedure

11: procedure LEARNONERULE()
12:   newRuleBody ⇐ {}                      ▷ start with general rule POS ← true
13:   POS_curr ⇐ POS_unc                    ▷ all positive tuples are covered by newrule
14:   NEG_curr ⇐ NEG                        ▷ all negative tuples are covered by newrule
15:   repeat                                ▷ begin FOIL's inner loop
16:     bestLit ⇐ getBestLiteral(candidateLiterals())   ▷ select the literal with maximum gain
17:     conjoin bestLit to newRuleBody      ▷ add literal to growing rule
18:     update POS_curr                     ▷ arity of POS_curr may increase
19:     update NEG_curr                     ▷ arity of NEG_curr may increase
20:   until |NEG_curr| == 0
21:   return POS ← newRuleBody
22: end procedure

23: procedure GETBESTLITERAL(C)             ▷ C is set of candidate literals
24:   maxGain ⇐ −∞
25:   for all candLit ∈ C do
26:     p' ⇐ |POS_curr| if candLit is added to newrule
27:     n' ⇐ |NEG_curr| if candLit is added to newrule
28:     p^++ ⇐number of tuples in POS_curr covered by candLit
29:     gain ⇐ GAIN(p', n', p^++)
30:     if gain > maxGain then
31:       bestLiteral ⇐ literal
32:       maxGain ⇐ gain
33:     end if
34:   end for
35:   return bestLiteral
36: end procedure
```

Here p and n are the sizes of POS_{curr} and NEG_{curr}, p' and n' are the sizes of the updated POS_{curr} and NEG_{curr} if L_i were added and p^{++} is the number of tuples in POS_{curr} that are covered by L_i. The main computational cost of FOIL comes from the evaluation of all the candidate literals every time a new literal is added to a growing clause. If the input data set cannot fit in main memory, management of the current positive and negative tuple becomes more complicated.

4.4 FOIL-D

Our implementation of FOIL with a direct interface to a RDBMS is called FOIL-D. FOIL-D assumes that the relational data may not fit in main memory and thus utilizes database operations, in terms of SQL statements. FOIL-D does, however, assume that other operations, such as generating and storing the candidate literals, fit in main memory. In our current implementation, FOIL-D only considers unnegated literals whereas FOIL considers both unnegated and negated literals. Note that there is nothing fundamental that prevents FOIL-D from considering negated literals though, and we plan to add support for them in the future.

Let the tuples defining the input relations be in database tables named POS, NEG and background relations B1, ..., BN where the mapping between tables and the relations in Table 4.1 is the obvious one. We store the uncovered positive tuples and the current positive and negative examples in database tables named POS_UNC, POS_CURR and NEG_CURR respectively. Due to the dynamics of the training sets, the number of columns of POS_CURR and NEG_CURR may change during the course of the algorithm. At any time though there is a one-to-one correspondence between the columns of POS_CURR (and NEG_CURR) and the distinct variables that have been introduced by either the head or a literal in the body of the growing clause.

The left-hand column of Table 4.2 lists the line numbers from Table 4.1 where FOIL-D issues database queries. The right-hand column gives the SQL statements[2] for the corresponding line. We are essentially performing *join* operations between two tables to obtain tuples that satisfy the conditions.

[2]These statements are compatible with version 4.1 of MySQL (http://www.mysql.com) and they can be adapted to other RDBMSs

Table 4.2 SQL statements used by FOIL-D. The left-hand-column indicates line numbers from Table 4.1 where FOIL-D issues database queries. The right-hand-column lists the corresponding SQL statements.

Line 3	CREATE TABLE POS_UNC LIKE POS
	INSERT INTO POS_UNC SELECT * FROM POS
Line 6	ALTER TABLE POS_UNC RENAME OLD_POS_UNC
	CREATE TABLE POS_UNC LIKE POS
	INSERT INTO POS_UNC SELECT $cols$ FROM OLD_POS_UNC
	LEFT JOIN POS_CURR ON $cond$ WHERE $cond'$
	DROP TABLE OLD_POS_UNC
Line 13	CREATE TABLE POS_CURR LIKE POS
	INSERT INTO POS_CURR SELECT * FROM POS_UNC
Line 14	CREATE TABLE NEG_CURR LIKE NEG
	INSERT INTO NEG_CURR SELECT * FROM NEG
Line 18	ALTER TABLE POS_CURR RENAME OLD_POS_CURR
	CREATE TABLE POS_CURR (...)
	INSERT INTO POS_CURR SELECT *cols*
	FROM OLD_POS_CURR, $rel(bestLit)$ WHERE *cond*
	DROP TABLE OLD_POS_CURR
Line 19	ALTER TABLE NEG_CURR RENAME OLD_NEG_CURR
	CREATE TABLE NEG_CURR (...)
	INSERT INTO NEG_CURR SELECT *cols*
	FROM OLD_NEG_CURR, $rel(bestLit)$ WHERE *cond*
	DROP TABLE OLD_NEG_CURR
Line 26	SELECT COUNT (*) FROM POS_CURR, $rel(candLit)$ WHERE $cond$
Line 27	SELECT COUNT (*) FROM NEG_CURR, $rel(candLit)$ WHERE $cond$
Line 28	SELECT DISTINCT COUNT (*) FROM POS_CURR, $rel(candLit)$ WHERE $cond$

4.5 Computational cost of FOIL-D

The primary computational cost of running FOIL-D on large databases comes from the *join* operations used to execute the six conditional `SELECT` statements (Lines 6, 18, 19, 26, 27 and 28). A join operation ($r \bowtie s$) between tables r and s selects a subset of the tuples in the cross product $r \times s$ that matches a specified set of constraints [123]. The cost of a join depends on a number of properties of the join such as the number and types of constraints (e.g., $>, <, =$), the presence of any database indexes on the join columns, and the number of tables. In FOIL-D all joins involve only equality constraints (*equi-joins*) between two tables. FOIL-D does perform joins with $k > 1$ equality constraints (k-column joins). We are agnostic about the implementation of join but measure the computational cost by the total number of joins performed to learn a theory.

An inspection of Tables 4.1 and 4.2 reveals that the number of joins needed to learn a rule set of R rules with L total literals where C total candidates are considered is

$$\text{\# of join operations} = R + 2L + 3C$$

The number of rules in a learned theory R is typically small, often less than five, and the number of literals L is also manageable, in the ten's at most. The number of candidates C, however, is often much larger and thus C dominates the other factors. The number of candidate literals considered at *each* step depends most strongly on the arity of the maximum arity relation and the number of old variables introduced so far [115]. For example, the number of candidate literals considered to append to a rule with 5 old variables and max arity of 3 is 136 [115]. FOIL-D uses type constraints on the arguments of the relations (or the columns in the database) which reduces the total number of candidate literals somewhat but not so much that it does not dominate in the above expression. Next, we describe extensions to FOIL-D that reduce the total number of join operations needed to learn a theory.

4.6 FOIL-DH: Estimating the size of join tables

To get the counts p', n' and p^{++} needed to compute the gain of a candidate literal we only need the *number* of tuples in the result sets of the SQL statements listed in Table 4.2 that we execute

on Lines 26-28. Thus, one way to reduce the total number of joins is to compute p', n' and p^{++} by more efficient means. The approach we take is to use histograms, an estimation method that have been used in query-optimization in databases [74, 75], to construct probabilistic models to estimate the number of tuples in each table. We call this system "FOIL-D with histograms" or simply FOIL-DH.

We maintain a histogram for each column of POS_UNC, NEG, B1, ..., BN, POS_CURR and NEG_CURR. Let $h^{r.c}$ be the histogram for the column $r.c$ of table r. The domain of $h^{r.c}$ is the same as the domain of the type of column $r.c$. The count $h^{r.c}(v)$ for value v is the number of tuples in r for which the value of column $r.c = v$.

Given the column histograms and an independence assumption we can estimate quickly the size of any k-column equi-join, such as the ones to get p' and n'. The probability $p^{r.c}(v)$ that a randomly selected tuple in table r has value v in column c is

$$p^{r.c}(v) = h^{r.c}(v)/|r|.$$

For an equi-join between columns c of table r and c' of table s, the probability that a randomly selected tuple from the cross product $r \times s$ satisfies the equality constraint, and thus is included in the result set, is

$$p(r.c, s.c') = \sum_v p^{r.c}(v) \times p^{s.c'}(v)$$

where the sum is over all values in the type of column $r.c$ (and $s.c$). The number of tuples in the result set is exactly given by

$$size(r \bowtie_1 s) = |r||s| \times p(r.c, s.c')$$

where $\bowtie_1$ indicates a 1-column join between r and s.

This is an exact calculation because the nature of the cross product guarantees that for a randomly selected tuple in $r \times s$, the value in a column from r is statistically independent of the value of a column from s. For multi-column joins the summary statistics in the histograms are not sufficient to exactly compute the size of the result set. If we assume, however, that the values of all columns in the join from the *same* table are statistically independent, we can estimate the size of a

$k-$column join as

$$\hat{size}(r \bowtie_k s) = |r||s| \prod_{i=1}^{k} p(r.i, s.i) \quad [4.1]$$

where, without loss of generality, we assume column $r.i$ is joined with $s.i$ for $1 \leq i \leq k$.

Our estimates of p' and n' for candidate literal $candLit$ then are

$$\hat{p}' = \hat{size}(\text{POS\_CURR} \bowtie_b rel(candLit))$$

and

$$\hat{n}' = \hat{size}(\text{NEG\_CURR} \bowtie_b rel(candLit))$$

where $rel(candLit)$ is the relation of $candLit$ and b is the number of old variables in $candLit$.

In order to compute the gain of $candLit$, in addition to p' and n', we need p^{++}, the number of tuples in POS_CURR (pre-update) still covered by the new rule if we accept $candLit$. Thus, we need to estimate the number of tuples in $r =$ POS_CURR that give rise to at least one tuple in $r \bowtie_k s$ where s is $rel(candLit)$.

To estimate p^{++}, we first compute q, the probability a randomly selected tuple in $r \times s$ matches on join columns 2 through k given the independence assumptions as

$$q = \prod_{i=2}^{k} p(r.i, s.i)$$

where again we assume column $r.i$ is joined with $s.i$ for $1 \leq i \leq k$. If $k = 1$ we set q to 1.0. If we pick randomly, with replacement, j tuples from $r \times s$, the probability that at least one matches on join columns 2 through k is

$$m(j) = (1.0 - (1.0 - q)^j).$$

A tuple in r with value v in the first join column has $h^{s.1}(v)$ tuples in the cross product that match in the first column and this many "chances" to match on the remaining $k - 1$ columns. So, we estimate the probability that the tuple will have at least one match in the result set as $m(h^{s.1}(v))$. Summing over all values and multiplying by $h^{r.1}(v)$ gives our estimation of p^{++}

$$\hat{p}^{++} = \sum_{v} h^{r.1}(v) * m(h^{s.1}(v)).$$

Our choice of doing this final sum over values of the first join column is arbitrary. We could have chosen any i of the k join columns as the one to do the final sum over; however, we would then calculate q as the probability of a match on the $k-1$ columns excluding join column i. Due to errors introduced by the sampling with replacement assumption of $m(j)$, in general the estimate $\hat{p}^{++}$ will be different for each choice of final join column i. In practice however, we have found $\hat{p}^{++}$ to be close for any choice of i.

As with the expression for estimating p' and n', our estimation $\hat{p}^{++}$ is exact for 1-column joins. Thus, we can exactly compute the gain for candidate literals with one old variable given the column histograms.

4.7 FOIL-DH(F): Estimating the highest gain literal

We speed up FOIL-D by using estimations of p', n' and p^{++} to estimate the highest scoring candidate literal in two ways. The first method simply estimates the gain of every candidate literal directly with $\hat{p}', \hat{n}'$ and $\hat{p}^{++}$ and chooses the one with highest estimated gain to append to the growing clause. This method eliminates the $3C$ joins needed to estimate the gain of the C candidate literals thereby reducing the number of joins performed to learn a theory with R rules and L literals to

$$\text{\# of join operations} = R + 2L.$$

A problem with this method is that the independence assumption often causes the estimated gains of the highest-scoring candidate literals with 2 or more old variables to be less than their true gains. In cases where the candidate literal with highest gain has multiple old variables, this procedure often erroneously estimates the highest scoring literal to be one with only 1 old variable.

However we have found empirically that the relative ranks of the candidate literals with multiple old variables, especially the highest scoring ones, is relatively stable following the estimation procedure. This leads to our other use of $\hat{p}', \hat{n}'$ and $\hat{p}^{++}$.

We also use the estimates of the candidate literal gains as a filter to reduce the number of candidate literals whose gains are computed exactly by performing actual join operations. Given filter size F we pick a candidate literal to conjoin to the growing clause with this method as follows.

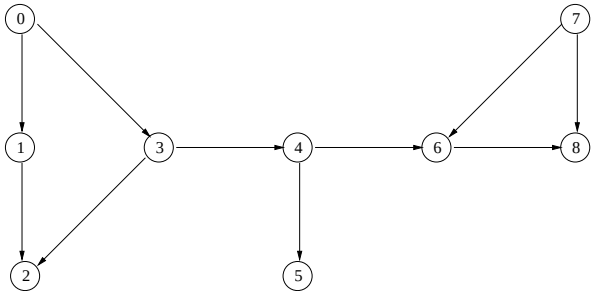


Figure 4.1 A small network, where ⟵ indicates linked-to [121]

First, we compute the estimated gains of the entire candidate set using our histograms as described above. Next, we rank the candidate literals with two or more old variables by estimated gain and compute the exact gain for the top F. Finally, we choose that the literal with the highest true gain among these F and the top scoring candidate literal with a single old variable. We call this method FOIL-DH(F).

The number of joins performed to learn a theory with FOIL-DH(F) is

$$\text{\# of join operations} = R + 2L + 3FL$$

which is still much smaller than $R + 2L + 3C$ for small F.

4.8 Experimental results

We conducted experiments using FOIL-D, FOIL-DH and FOIL-DH(F) on three learning tasks taken from machine learning literature, which were used by Quinlan [121] to illustrate the power of FOIL. The aim of the experiments performed is to determine whether the cost of obtaining accurate hypotheses, with respect to FOIL, can be significantly reduced by using probabilistic models (FOIL-DH) and filtering (FOIL-DH(F)) to estimate the scores of literals.

Our first learning task involved learning the definition of can-reach in the network [121] shown in Figure 4.1. Extensional definitions of can-reach(N,N), not-can-reach(N,N) as well as linked-to(N,N) were given as positive examples, negative examples and background knowledge respectively. Variables in this relation consists of one type, N (Node). The goal is to learn a general

Table 4.3 Rules learned for target relation can-reach

FOIL type	Rules learned
FOIL-D, FOIL-DH(1-6)	can-reach(X0, X1) ← linked-to(X0, X1) can-reach(X0, X1) ← linked-to(X0, X2), can-reach(X2, X1)
FOIL-DH(0)	can-reach(X0, X1) ← linked-to(X0, X2), linked-to(X3, X1), linked-to(X2, X4), can-reach(X2, X1) can-reach(X0, X1) ← linked-to(X0, X2), linked-to(X3, X1), linked-to(X2, X4), linked-to(X4, X5), linked-to(X6, X3)

Table 4.4 Rules learned for target relation eastbound

FOIL type	Rules learned
FOIL-D, FOIL-DH(0-6)	eastbound(X0) ← has-car(X0, X1), closed-top(X1), short(X1)

definition for can-reach. Table 4.3 shows the rules learned by FOIL-D and FOIL-DH(0-6), which are using filter sizes set to 0 through 6.

The second learning task was from the INDUCE system by Michalski *et al.* [97]. Trains in Figure 4.2 have different numbers of cars, with various shapes and tops, carrying loads of various number and shape. The task is to distinguish between eastbound and westbound trains. Clauses learned by FOIL-D and FOIL-DH(0-6) are shown in Table 4.4.

The third and final task we considered was that of learning family relationships from Hinton [65]. Figure 4.3 shows two isomorphic families of twelve members each. There are twelve relationship types to be learned: mother, father, wife, husband, son, daughter, sister, brother, aunt, uncle, niece and nephew. The rules learned by FOIL-DH(1) for this task are identical to those learned by FOIL-D on all twelve relations. Tables 4.5 and 4.6 show the definitions learned by FOIL-D and FOIL-DH(0-6) for uncle and mother respectively. Since all uncles happen to also be married in these families, the second literal in both uncle clauses serves the purpose of asserting

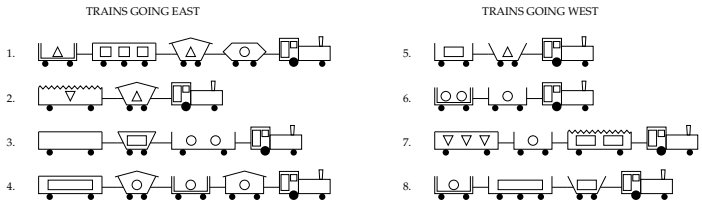


Figure 4.2 Eastbound and westbound trains from Michalski et al. [97]

Table 4.5 Rules learned for target relation uncle

FOIL type	Rules learned
FOIL-D, FOIL-DH(1-6)	uncle(X0, X1) ← niece(X1, X0), husband(X0, X2)
	uncle(X0, X1) ← nephew(X1, X0), husband(X0, X2)
FOIL-DH(0)	No rules learned

that person X0 is a man. The rules learned by FOIL-D and FOIL-DH(1-6) for other relations are similar in structure. FOIL-DH(0) fails to learn any rules[3]

When learning the relations can-reach and the twelve familial relationships, FOIL-D as well as FOIL-DH(1-6) constructs accurate, compact rules. However, for the can-reach relation, FOIL-DH(0) generates long, convoluted clauses that consist of only literals with *one* old variable. Furthermore, FOIL-DH(0) does not generate any rules for any of the twelve family relationships. The reason for this can be seen in Figure 4.4. Whenever FOIL-DH(0) makes an estimation of the gain for literals with *multiple* old variables, it always under-estimates the actual gain of the highest scoring literals, whereas it always estimates the *exact* actual gain for literals with exactly one old variable. Hence, literals with exactly one old variable will score higher than literals with multiple old variables, resulting in the unique property of the rules learned by FOIL-DH(0) for the relation can-reach. This reasoning also supports the results of FOIL-DH(0) on the relationships of the family dataset because the definitions of those relations require literals with two old variables to be added first to the body of the clause.

[3]FOIL-DH(0) learns no rules in cases where the first rule it "learns" covers zero positive examples and is discarded.

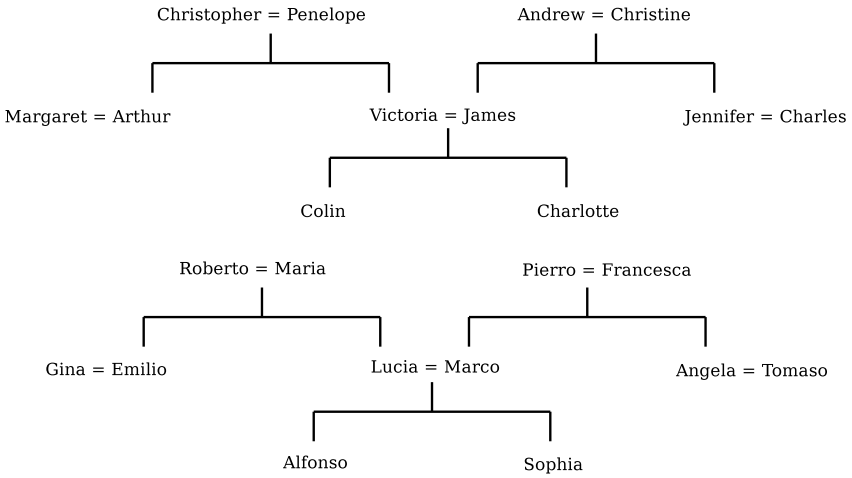


Figure 4.3 Two family trees, where = means married-to, from [65] and [121]

Table 4.6 Rules learned for target relation mother

FOIL type	Rules learned
FOIL-D, FOIL-DH(1-6)	mother(X0, X1) ← daughter(X1, X0), husband(X2, X0) mother(X0, X1) ← son(X1, X0), husband(X2, X0)
FOIL-DH(0)	No rules learned

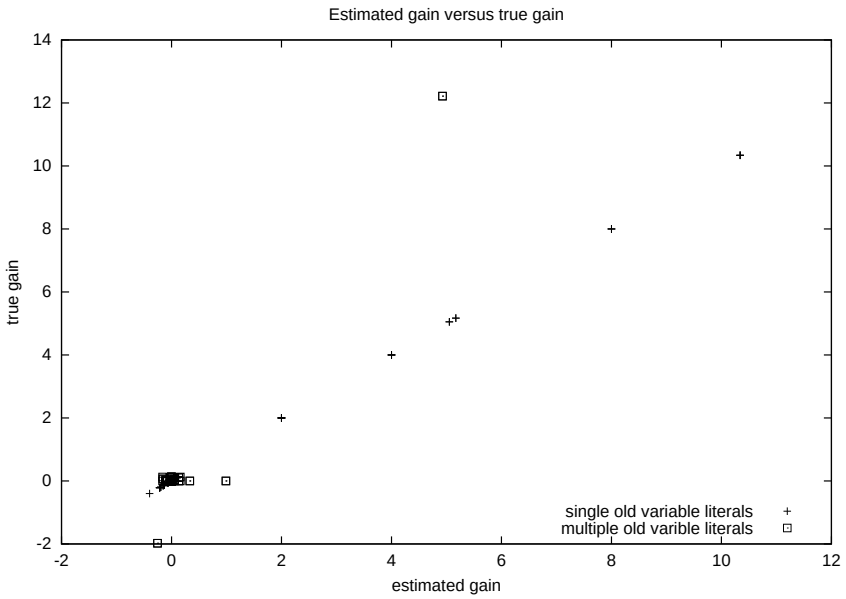


Figure 4.4 Estimated gain versus actual gain calculations for adding the first literal in the relation uncle. The literal with the highest scoring true gain, niece(X1,X0), has true gain of near 12 and estimated gain of near 5. This literal is not chosen by FOIL-DH(0), but since this literal has the highest estimated score of the literals with multiple old variables it is correctly chosen by FOIL-DH(1).

How then was FOIL-DH(1), with a filter size of just 1, able to learn the correct definitions for can-reach and the twelve familial relationships? One possible explanation is that even though the estimations for literals with multiple old variables are lower, the order of the estimated gain for these literals is maintained. This would allow FOIL-DH(1), which specifically considers the highest estimated literal with multiple old variables, to accurately select the best literal.

Figure 4.5 shows the total number of JOINs performed for the different FOIL types. FOIL-DH(0) to FOIL-DH(6) grows linearly in the number of JOINs performed. Hence, the use of FOIL-DH(F), with a resonable filter size provided as input, would still provide significantly more savings than FOIL-D.

This work was originally published in the Proceedings of the Fourteenth International Conference on Inductive Logic Programming [17].

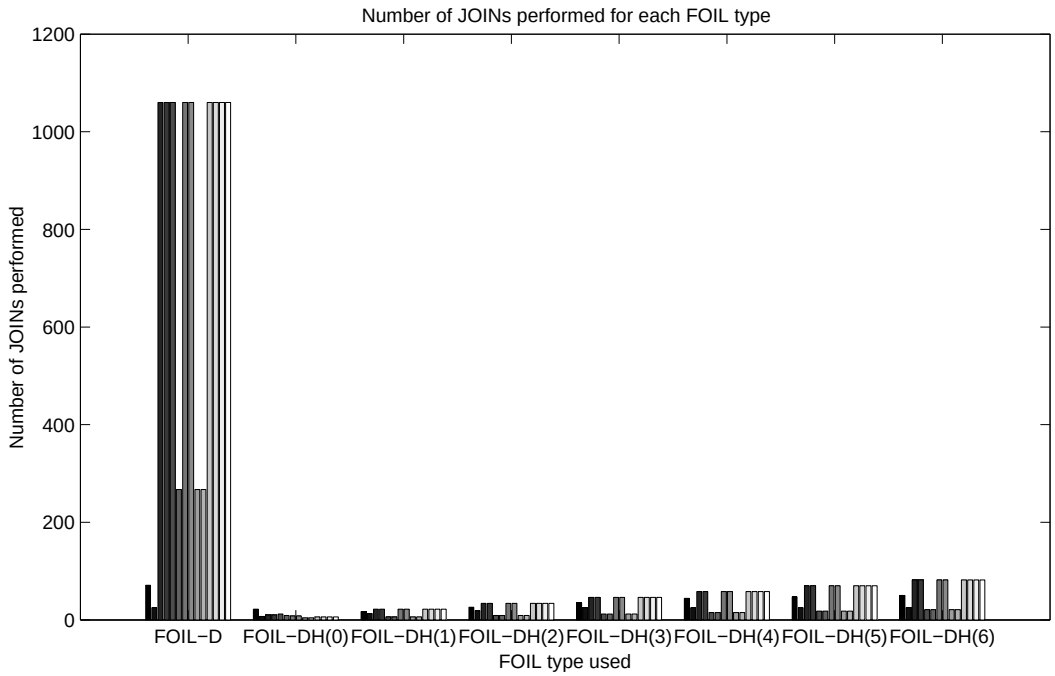


Figure 4.5 Total number of JOINs performed for the different FOIL types when learning the following relations (in order from left to right in the histogram): can-reach, eastbound, mother, father, wife, husband, son, daughter, sister, brother, aunt, uncle, niece, nephew

Chapter 5

Inferring Regulatory Networks from Time Series Expression and Relational Data via Inductive Logic Programming

5.1 Introduction and Motivation

Diseases are often caused by genetic factors, environmental factors or both. Genetic mutations or environmental toxins cause cells to respond differently, often in an unfavorable manner. Gaining insight into the underlying regulation of genes within organisms is important not just for understanding the cause of diseases but also for developing treatments.

For nearly a decade now, DNA microarray technology has enabled the simultaneous measurement of mRNA abundance of genes in an organism under normal conditions or under various treatments or perturbations. However, microarray experiments still have many sources of error: sample preparation, hybridization, scanning, image processing, normalization, etc. Because samples for microarray data are usually obtained by pooling extracts from a population of cells rather than a single cell, in addition to experimental variables and limitations of the technology, the measurements obtained can be noisy. Noisy data inherently makes it more difficult to reverse engineer the underlying regulatory network.

Despite the difficulty of deciphering genetic regulatory networks from microarray data, numerous approaches to the task have been quite successful. Friedman *et al.* [45] 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.* [116] followed up that work by using BNs to learn master regulator sets. Other approaches include Boolean networks (Akutsu

et al. [3], Ideker *et al.* [72]) and other graphical approaches (Tanay and Shamir [155], Chrisman *et al.* [24]).

The methods above can represent the dependence between interacting genes, but they cannot capture causal relationships. Pe'er *et al.* [117] 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. In our previous work [108], we proposed that the analysis of time series gene expression microarray data using Dynamic Bayesian networks (DBNs) could allow us to learn potential causal relationships (Figure 5.1). The constancy of the arcs of the DBN is justified by an assumption that the process being modeled is *stationary* though not static. While values of variables may change over time, the manner in which the value of one variable influences the value of a variable at the next time step (i.e., the parents and the conditional probability distribution for the latter variable) will not change.

DBN learning can provide more insight into causality than ordinary BNs. An induced arc from gene X_1 to gene X_2 in an ordinary BN simply means that the expression of gene X_1 is a good predictor of the expression of gene X_2 *at the same time* (Figure 5.2a). While this good prediction may be because expression of gene X_1 influences expression of gene X_2, it could just as easily be because expression of gene X_2 influences expression of gene X_1 or expression of both gene X_1 and gene X_2 are influenced by expression of another gene X_3 (Figure 5.2b). On the other hand, an induced arc from gene X_1 to gene X_2 in a DBN implies that expression of gene X_1 at one time slice is a consistently good predictor of gene X_2 at *the next time slice*. This good prediction

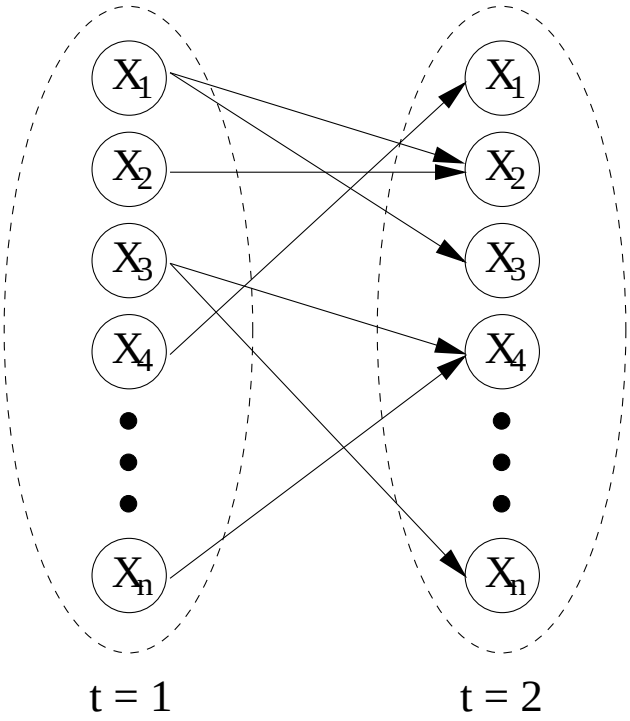


Figure 5.1 Simple DBN model. Labeled circles within a dotted oval represent our variables in one time slice. Formally, arcs connecting variables from one time slice to variables in the next have the same meaning as in a BN, but they intuitively carry a stronger implication of causality. We note that in a DBN with more time slices, the arcs are always the same, e.g., the arc from X_1 at time slice 1 to X_2 at time slice 2 is also present from time slice t to time slice $t + 1$ for all $1 \leq t < T$ where T is the last time slice in the model.

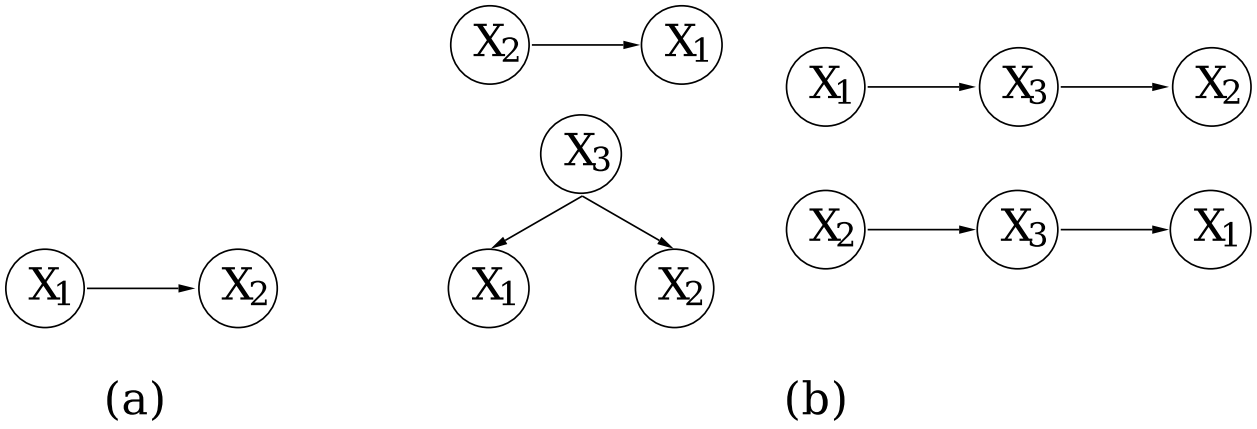


Figure 5.2 (a) X_1 may be a good predictor of X_2, but is X_1 *regulating* X_2? (b) Ground truth might be any one of these or a more complicated variant.

is unlikely to be because expression of gene X_2 influences expression of gene X_1; intuitively, it seems likely to be because expression of gene X_1 influences expression of gene X_2.[1]

While temporal gene expression data contains causal information in the temporal data sequence, the dependence on the appropriate sampling rate, the small sample size, the large number of variables, and the presence of many hidden (signaling and other molecular interactions for which we do not have measurements) variables make it difficult for learning algorithms to completely determine the network.

In this chapter, our goal is to utilize the abundant information available from many years of low-throughput as well as recent high-throughput research that are currently available in public databases to infer new relationships that cannot be learned from expression data alone. We are interested in discovering whether ILP is able to infer theories for particular pathways from time series microarray data and use other known relational information about the organism to refine what is already known about that pathway. Specifically, we formulate the learning in the same

[1] An arc in a DBN does not establish causality definitively. Nevertheless, if a learned DBN contains arcs that imply novel potential causal relationships, in some cases biologists can test these novel relationships with additional, more focused (and time-consuming) experiments.

way as a DBN by learning theories of gene expression that are good predictors of the expression of particular genes at the next time step.

Regulatory sequences control gene expression temporally as well as spatially by *cis*-acting elements and *trans*-acting factors. *Cis*-acting elements are DNA sequences in the vicinity of the target gene, usually upstream of the transcription start site. *Trans*-acting factors, bind to the *cis*-acting sequences to control gene expression in several ways: the factor may (1) be expressed temporally (specific times in life cycle), (2) be expressed spatially (in a specific location), (3) require modification (phosphorylation), (4) be activated by ligand binding, (5) be sequestered until an environmental signal allows it to interact with the nuclear DNA. Hence, by integrating temporal gene expression data with additional information such as protein-protein interaction, transcription factor and kinase-substrate (phosphorylation) information, we believe we can capture some of these causal relationships and underlying mechanisms.

5.2 Related Work

Our goal in this chapter is similar to that of Tu *et al.* [158]. We are interested in determining whether ILP can learn the pathway links between causal genes and target genes that explain the regulatory relationships between them. In the past few years, we have seen an increase in the use of inductive logic programming (ILP) methods for learning functional genomics [19, 85, 127, 152], metabolic networks [154] and also predicting gene expression levels [114]. Papatheodorou *et al.* [114] used Abductive logic programming (ALP) to learn rules that would explain how gene interactions can cause changes in gene expression levels.

Recently, Fröhler and Kramer [47], applied ILP to the task of predicting up- and down-regulation of gene expression in *S. cerevisiae* under different environmental stress conditions [52] with the use of additional information. Fröhler and Kramer used the data from Middendorf *at al.* [98], where the presence of transcription factor binding sites (pruned list of 354 after removing redundant and rare sites) in the gene's regulatory region and the expression levels of regulators (selected list of 53, 50 of which were top ranking regulators identified by Segal *et al.* [139]) are used to predict gene regulation.

Following Middendorf, Fröhler and Kramer consider three classes of gene activity: up-regulation ($>$ 1.2 log fold change), down-regulation ($<$ -1.2log fold change), and no change. The up- and down-regulated genes consist of 5% of all the data points since 95% of the expression were unstimulated. Their results report on discriminating between up- and down-regulation, with excellent results, although the original work from Middendorf's showed that discriminating between the three classes is a much harder task. We similarly discretize into three classes to reduce noise, but our up- and down-regulated classes are about 20% of the total number of examples, so one would expect the discrimination task in our case to be harder.

Our work differs from that of Fröhler and Kramer in four ways. First, we learn rules to predict the up-regulation of a gene based on the activity and expression of genes from the *previous time step* as in a DBN since we are interested in learning causal relationships from the data. Secondly, we discretize the gene expression data by comparing two consecutive time series measurements under the same experimental condition and determining whether the change in expression was up, down or same based on a threshold of greater than 0.3, less than -0.3, or in between. Thirdly, we use information on transcription factors rather than transcription factor binding sites and we do not restrict the transcription factor or regulator set as our goal is to learn possible new players in the network. Finally, we use Aleph [146] instead of Tilde [15].

5.3 ILP and Aleph

Inductive logic programming (ILP) is a popular approach for learning first-order, multi-relational concepts between data instances. ILP uses logic to induce hypotheses from observations (positive and negative examples) and background (prior) knowledge by finding a logical description of the underlying data model that differentiates between the positive and negative examples. The learned description is a set of easily interpretable rules or clauses.

There are many ILP systems available, but we chose to use Aleph because it has been shown to perform well even on fairly large datasets. This is because Aleph implements the Progol algorithm [102], which learns rules from a pruned space of candidate solutions. The Progol algorithm structures and limits the search space in two steps. Initially, it selects a positive instance to serve

as the seed example and searches the background knowledge for the facts known to be true about the seed example - the combination of these facts form the example's most specific or saturated clause. Then, Aleph defines the search space to be clauses that generalize a seed example's saturated clause, and performs a general to specific search over this space. The key insight of the Progol algorithm is that some of these facts explain the seed example's classification, thus generalizations of those facts could apply to other examples.

5.4 Data and Methodology

To test our hypotheses, we use time series gene expression data of environmental stress response experiments, including DNA-damaging agents from Gasch *et al.* [51, 52]. We chose to use this dataset on yeast because yeast is a model organism used for studying many basic cellular processes and there exist many publicly accessible databases containing various sources of data from many years of research. We focused our study on the DNA damage checkpoint pathway because it is an important pathway that has been widely studied. There are about 6500 genes in yeast, 19 of which are considered to be in the "DNA damage checkpoint" pathway based on a recent review by Harrison and Haber [60].

It is well known that a common problem with current microarray data is the small number of sample points and the large number of features or genes. Nevertheless, it is hoped that discretization as well as other sources of information will permit useful results to be obtained. We determined the relative change in expression from one time step to the next by comparing the expression levels between two consecutive time series measurements. The time series data were discretized into one of three possible discrete values by comparing two consecutive time series measurements: if the change increased by 0.3, we consider the expression to be up-regulated, if the change decreased by 0.3, we consider the expression to be down-regulated, otherwise we say the expression stayed the same.

As alluded to earlier, there are many other spatial and molecular interactions that are not captured by expression data. Known transcription factors for specific genes can allow the learning algorithm to focus on specific proteins that are known to interact with the DNA of the target gene.

The learning algorithm could also potentially discover combinations of transcription factors (pairs, trios, etc.) required to trigger a change in expression of a particular set of genes. Because transcription factors can also interact with other proteins or metabolites on their way to activating gene expression, background knowledge of proteins that are known to interact with each other can allow for the discovery of novel proteins in the pathway. Furthermore, an estimated 30% of proteins need to be phosphorylated in order to trigger a change in the protein's function, activity, localization and stability [76]. Thus, background knowledge about a large number of protein phosphorylation events in yeast was also included [38].

Recent technological advances have produced more high-throughput data that capture different types of interactions. ChIP-chip (chromatin immunoprecipitation, a well-established procedure to investigate interactions between proteins and DNA, coupled with whole-genome DNA microarrays) technology allows one to determine the entire spectrum of in vivo DNA binding sites for any given transcription factor or protein. Mass spectrometry, large-scale two-hybrid screens, single-cell analysis of flow cytometry, and protein microarrays have all been used to generate high-throughput measurements of certain types of molecules such as proteins, metabolites, protein-protein interactions and also signaling events such as phosphorylation within cells. Most of these data are also known to be noisy especially those obtained through high-throughput methods that were conducted in vitro (outside the organism). High-throughput protein-protein interaction and phosphorylation data are especially noisy because the conditions under which the data are collected differs quite significantly from that in a cell, i.e. detecting interactions that would not actually occur in vivo (inside the organism) or missing interactions that actually take place.

We aim to link known interactions with gene expression activity to possibly learn new mechanisms. We do this by associating the up- or down-regulation of specific genes from the previous time step with its transcription factor, a protein it might interact with, or a phosphorylation event. We assume that an event in the previous time step will contribute to the change in expression at the current time. This assumption does not necessarily hold for all biological activity but a similar assumption, that of using a gene's expression level to approximate the activity of other genes within the same pathway, have been used by others [166].

Table 5.1 Cross validation accuracies

Fold	0	1	2	3	4	5	6	7	8	9	Average
Accuracy	0.73	0.87	0.81	0.72	0.83	0.84	0.73	0.79	0.75	0.78	0.79

The MIPS Comprehensive Yeast Genome Database (CYGD) [58] provided much of the information regarding yeast genes, their function, location, phenotype and disruption. We obtained protein-protein interaction data from BioGRID [148], transcription factor data from the YEASTRACT database [157], and over 4000 yeast phosphorylation events from Ptacek *et al.* [38]. The ILP system, Aleph [146], was used to learn rules from the data.

We first learn rules using inductive logic programming (ILP) to predict the discretized gene expression level at the *next time step* as in a DBN. Then we use the learned theory to generate a pruned network or graph that show interactions corresponding to proofs for the rules.

5.5 Experiments and Results

We performed ten-fold cross validation experiments to learn theories for predicting held-out gene expression values for genes in the DNA damage checkpoint pathway at the *next time step*. The discretized microarray experiments were divided into ten folds, grouping replicate experiments together to avoid bias, based on the different experimental conditions.

We obtained an accuracy of 79% on predicting up-regulated examples averaged over ten folds of the cross-validation procedure (see Table 5.1). A comparison of how well the rules found by ILP with a baseline algorithm such as Naïve Bayes would give us a better idea of whether the additional relational data was helpful. However, a better comparison would be to compare these results with those of a DBN, but that will be left for future work.

Examples of some of the rules learned across the folds are:

Rule 1 up(GeneA,Time,Expt) :-

previous(Time,Time1), down(GeneA,Time1,Expt),

interaction(tof1,GeneA), up(tof1,Time1,Expt),
function(GeneA,'CELL CYCLE AND DNA PROCESSING:cell cycle:mitotic cell cycle and cell cycle control:cell cycle arrest').

Rule 2 up(GeneA,Time,Expt) :-
previous(Time,Time1), down(GeneA,Time1,Expt),
phosphorylates(GeneA,GeneE),
up(GeneE,Time1,Expt),
transcriptionfactor(GeneF,GeneE), down(GeneF,Time1,Expt),
transcriptionfactor(GeneF,cdc20), down(cdc20,Time1,Expt).

Rule 3 up(GeneA,Time,Expt) :-
previous(Time,Time1), down(GeneA,Time1,Expt),
interaction(GeneE,GeneA), down(GeneE,Time1,Expt),
interaction(GeneE,mms4), down(mms4,Time1,Expt),
function(GeneA,'METABOLISM').

These rules all specify the activity of specific genes involved in the larger DNA damage pathway. Tof1 is a subunit of a replication-pausing checkpoint complex (Tof1p-Mrc1p-Csm3p) that acts at the stalled replication fork to promote sister chromatid cohesion after DNA damage, facilitating gap repair of damaged DNA. Cdc20, which is regulated by cell-cycle genes, is an activator of anaphase-promoting complex/cyclosome (APC/C), which is required for metaphase/anaphase transition. It is part of the DNA damage checkpoint pathway and directs ubiquitination of mitotic cyclins, Pds1p, and other anaphase inhibitors. Finally, Mms4 is a subunit of the structure-specific Mms4p-Mus81p endonuclease that cleaves branched DNA and is involved in recombination and DNA repair.

The learned rules prove examples, and proofs generate paths between genes, so using the theories from all the folds we generated graphs. The graphs only show links that can be used in proofs

for at least five examples from the train and test set. We could have combined all the data from the train and test sets to learn rules before generating the graphs in order to make use of all the data, but we just used all the rules learned from the ten-fold cross validation experiments.

The width of a line in the graph is an indication of the proportion of examples used in the proof. Note that the graph only displays literals that were used in successful proofs. Hence, paths in the graph correspond to proofs and the nodes are examples of literals which were used to prove the rules. The learned graph of interactions amongst the 19 genes in the DNA damage checkpoint pathway are shown in Figure 5.3. A more detailed graph showing interactions amongst the genes in the DNA damage checkpoint pathway as well as transcription factors and phosphorylators can be seen in Figure 5.4.

5.6 Discussion

The DNA damage checkpoint monitors genome integrity, and ensures that damage is corrected before cell division occurs. When DNA damage is detected, the checkpoint network transmits signals that stall the progression of the cell cycle and mobilize repair mechanisms. The graph resulting from our analysis recapitulates many of the central aspects of this signaling network, and connects that network temporally to the normal progression through the cell cycle.

DNA damage (often in the form of a double strand break) is first recognized by MRX, a protein complex consisting of Mre11, Rad50 and Xrs2. These proteins are shown to interact together slightly to the left of the middle of Figure 5.4, with Mre11 linked to both Xrs2 and Rad50. The MRX complex coordinates the restructuring of the damaged region. MRX stimulates the phosphorylation of histones, H2A, in the region adjacent to the DNA double strand break (via Tel1) and recruits an exonuclease to generate a stretch of single stranded DNA. Our graph does not include physical interactions between Tel1 and the MRX complex, however both are connected through Mec1 and through the DNA binding protein Rap1. Rap1 can act as an inducer or a repressor, and is active in many disparate elements of cell biology, including ribosome synthesis and telomere preservation.

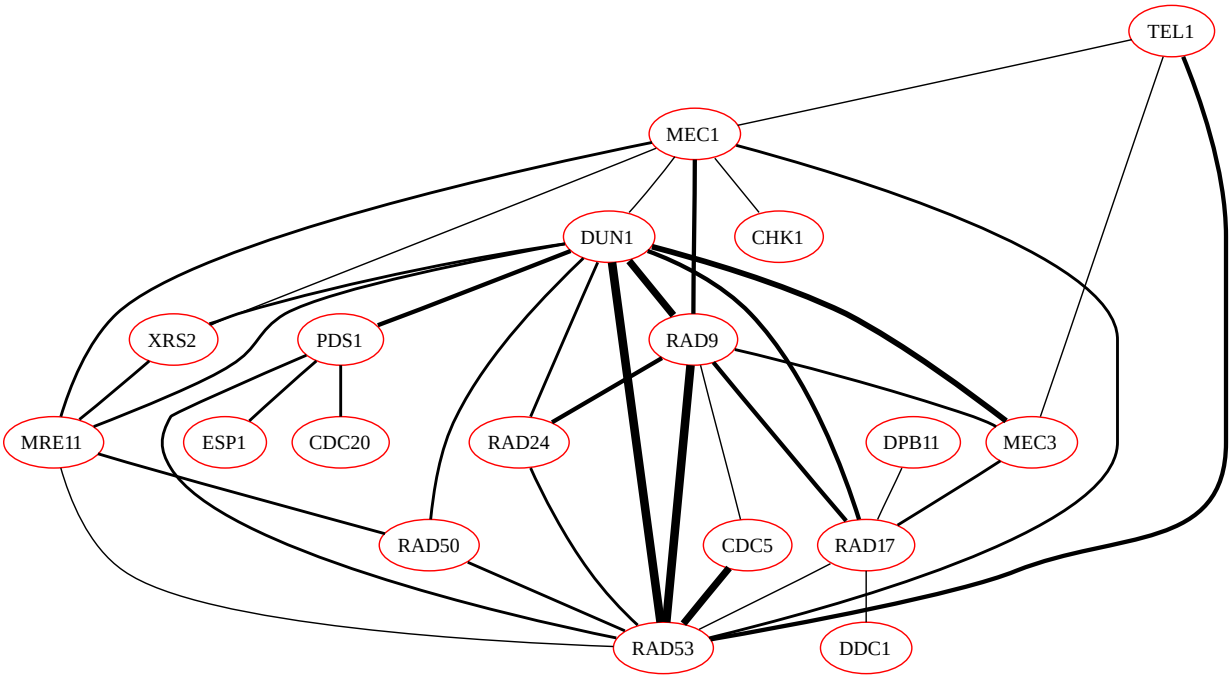


Figure 5.3 Learned graph of interactions from successful proofs amongst the 19 genes in DNA damage checkpoint pathway from Harrison and Haber [60]. Straight edges represent protein-protein interactions. The width of a line in the graph is an indication of the number of examples that used this interaction in a proof.

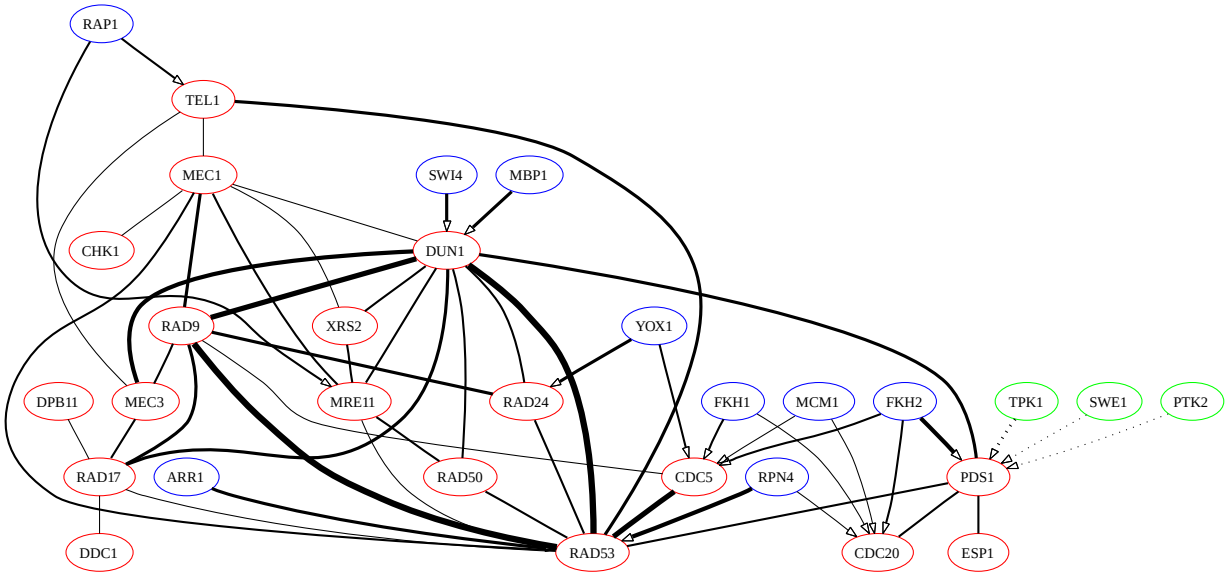


Figure 5.4 Learned graph of interactions from successful proofs for the DNA damage checkpoint pathway. Straight edges represent protein-protein interactions, solid lined arrows represent transcription factor to target gene interaction, and dotted arcs represent kinase to substrate phosphorylation. The width of a line in the graph is an indication of the number of examples that used this interaction in a proof. A larger figure can be found at: http://www.biostat.wisc.edu/~ong/new2a.ps

Once single stranded DNA is generated, it is bound by the heterotrimer replication protein A (RPA) and two things occur. First, Mec1/Ddc2 binds and activates the signaling cascade. Mec1 phosphorylates Rad9 (shown as physical interaction in the graph), which in turn recruits Rad53. Ddc2 is conspicuously absent from this graph due to the requirement that only links that can be used in proofs for a certain number of examples are displayed. Next, the 9-1-1 clamp, which consists of three proteins Rad17, Mec3 and Ddc1, binds and demarcates the ssDNA/dsDNA junction, and facilitates some of the interactions described above. The 9-1-1 clamp components are grouped at the left side of the graph, linked by protein-protein interactions.

At the heart of the signaling network is Rad53, a well-connected, essential yeast kinase. Rad53 phosphorylates Dun1, a kinase whose activity ultimately controls much of the transcriptional response to DNA damage. Dun1 is also a very central protein in this network, demonstrating interactions with the 9-1-1 clamp, Rad24, the MRX complex, Rad53 and Pds1, a cell cycle control gene. Finally, Rad53 signals cell cycle arrest through Pds1 (via Cdc20), and Cdc5. Pds1 governs entry into mitosis, and Cdc5 controls exit from mitosis. All of these interactions are present in our results.

DNA damage is an inevitable consequence of DNA synthesis, and the graph reveals that the expression of the gene responsible for signaling the induction of DNA repair genes (Dun1) is coordinated by two transcription factors (Swi4 and Mbp1) that are active in the period just before DNA synthesis begins. Likewise, the transcription factors Mcm1, Fhk1 and Fkh2 are known to control the transition from G2 to mitosis, and in our graph these TFs are linked to Cdc5, Cdc20 and Pds1, which govern this transition.

At a broader level, the results shown in Figure 5.4 illustrates the centrality of Rad9, Rad53 and Dun1. These genes are instrumental in coordinating the various aspects of this response: detection of damage, cell cycle arrest, and mobilization of repair mechanisms.

5.7 Chapter Summary

As a first step, we concentrated our experiments on learning the DNA damage checkpoint pathway because it is an important pathway that have been implicated in cancer and aging, and

because it has been well studied. This pathway plays an important role by responding to single and double-stranded DNA breaks, and is therefore often activated in stressful environments. Hence, it involves a lot of signaling kinases that phosphorylates proteins that are already present within the cell or that only require molecular amounts to trigger a response.

After performing our analysis, we found that the phosphorylation dataset from Ptacek *et al.* [38] did not specifically include any phosphorylation relationships for the kinase and substrates in the DNA damage checkpoint pathway. The results we obtained show that our method is quite good at learning important pathway interactions and regulators despite the fact that the data may be noisy or incomplete. This further emphasizes the utility of integrating different data types, since many potential interactions, including those that were not evident from single data sources were identified.

This work was originally published in the Proceedings of the Sixteenth International Conference on Inductive Logic Programming [109].

Chapter 6

Mode Directed Pathfinding

6.1 Introduction

Over the past few years there has been a surge of interest in learning from multi-relational domains. Applications have ranged from bioinformatics [111], to web mining [22], to law enforcement and security [77]. Typically, learning from multi-relational domains has involved learning rules about distinct entities so that they can be classified into one category or another. As an example, given a drug compound and a database describing drug compound properties, one may want to predict whether the compound is carcinogenic or not [147]. However, there are also interesting applications that are concerned with the problem of learning whether a number of entities are connected. Examples of these include determining whether two proteins interact in a cell (Figure 6.1), whether two identifiers are aliases, or whether a web page will refer another one. These are examples of link mining [56], which in general can be described as learning connections in a graph.

A number of approaches have been proposed for exploiting link structure to improve results. Most of these approaches are graph based, including SUBDUE, which discovers substructures that compress the original database to represent interesting structural concepts in the data [25], and ANF, which computes properties pertaining to the connectivity or neighborhood structure of the graph [112].

Our focus here is on first-order learning systems such as ILP. Most of the approaches in ILP rely on hill-climbing heuristics in order to avoid the combinatorial explosion of hypotheses that can be generated in learning first-order concepts. However, hill-climbing is susceptible to local maxima

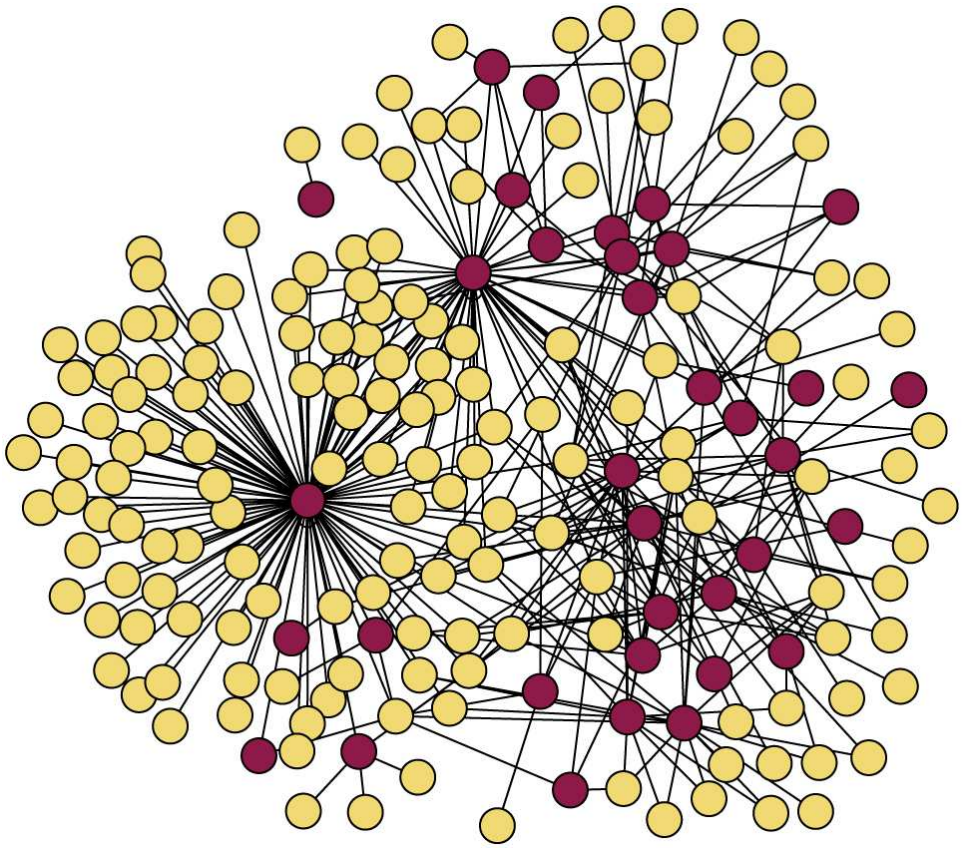


Figure 6.1 Graph showing branching factor of protein-protein interactions [55].

and local plateaus. Hence, it becomes crucial to constrain the search process. A seminal work in directing the search in ILP systems was the use of *saturation* [132], which generalizes literals in the seed example to build a *bottom clause* [103]. The theoretical role of the bottom clause is to anchor one end of the search and introduce constants and variable co-references. Another method for directing the search, *pathfinding*, was proposed by Richards and Mooney [128]. Pathfinding was originally proposed in the context of the FOIL ILP system, which does not rely on creating a saturated clause. Richards and Mooney realized that the problem of learning first-order concepts could be represented using graphs, and using the intuition that if two nodes interact there must exist an explanation, proposed that the explanation should be a connected path linking the two nodes. Craven *et al.* [26] developed an algorithm for pathfinding, that is a variant of Richards and Mooney's relational pathfinding method.

In the last few years, ILP systems have been increasingly applied to large datasets, where the number of individuals and the branching factor per node can be very large [31, 32]. Ideally, saturation based search and a good scoring method should eventually lead us to the interesting clauses. Unfortunately, experience has shown that the search space can grow so quickly that we risk never reaching an interesting path in a reasonable amount of time (See Figure 6.2). This prompted us to consider alternative ways, such as pathfinding, to constrain the search space.

Unfortunately, the original pathfinding algorithm assumes the background knowledge forms an undirected graph. In contrast, the saturated clause is obtained by using *mode declarations*: in a nutshell, a literal can only be added to a clause if the literal's *input* variables are known to be bound. Mode declarations thus embed directionality in the graph formed by literals.

We propose a new algorithm for finding paths in the saturated clause. Our major insight is that a saturated clause for a moded program can be described as a directed *hypergraph*, which consists of nodes and hyperarcs that connect a nonempty set of nodes to one target node. Given this, we show that path finding can be reduced to reachability in the hypergraph, whereby each hyperpath will correspond to a hypothesis. However, we may be interested in non-minimal paths and in the composition of paths. We thus propose and evaluate an algorithm that can enumerate all such hyperpaths according to some heuristic.

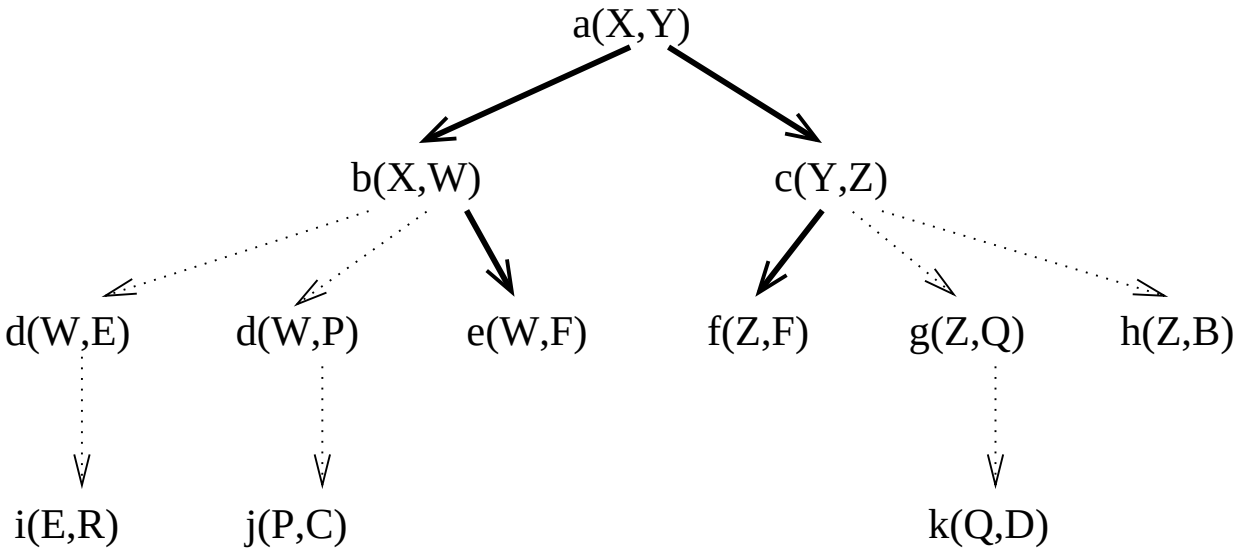


Figure 6.2 Search space induced by a saturated clause. The literal at the root of the graph represents the head of the saturated clause. All the other literals in the graph appear in the body of the saturated clause. Bold arcs indicate a possibly interesting path linking X to Y. Dotted arcs indicate parts of the search space that will not lead to determining connectivity of the two entities.

This chapter is organized as follows. First, we review Richards and Mooney's [128] groundbreaking work on pathfinding. Second, we discuss how the saturated clause can be seen as forming a hypergraph. We then present our algorithm through an example, and give conclusions.

6.2 Path Finding

We illustrate the relational path finding algorithm by Richards and Mooney [128] with their example using a portion of the data in Hinton's family domain [66]. Suppose we want to learn the relationship *uncle*, given an initially empty rule, the positive instance `uncle(Arthur,Charlotte)` and the database in Figure 6.3a, which includes family relations between seven individuals. Our first step, shown in Figure 6.3b, is to explore nodes one hop away from `Arthur` and `Charlotte`. Starting from `Arthur`, we have a connection to individuals `Christopher` and `Penelope`. Starting from `Charlotte`, we get `Victoria` and `James`. Since we have not encountered an intersection, we continue by expanding all paths originating from `Arthur` (Figure 6.3c). Here, we recognize an intersection between paths from `Arthur` and `Charlotte` at `Victoria`. By visual inspection, we can see that there are actually three paths leading from `Arthur` to `Victoria`. In this case, the path with the best overall accuracy would be chosen as a rule:

uncle(X,Y) $\leftarrow$ **parent(Z,X),parent(Z,W),parent(W,Y)**.

If the rule is still too general, usually because it requires non-relational antecedents, it can further be specialized by using a standard FOIL-like technique.

Our first impression was that we could easily apply this pathfinding algorithm to the saturated clause. We could select a positive instance (seed), and instead of directly applying the path finding algorithm, first saturate the instance to generate a saturated bottom clause. *Literals* can serve as nodes, and *shared variables* and *constants* can serve as edges. Unfortunately, just using path finding might result in clauses that break the mode induced language bias. We show in sections 6.3 and 6.4 how we make use of mode declarations to find connected paths.

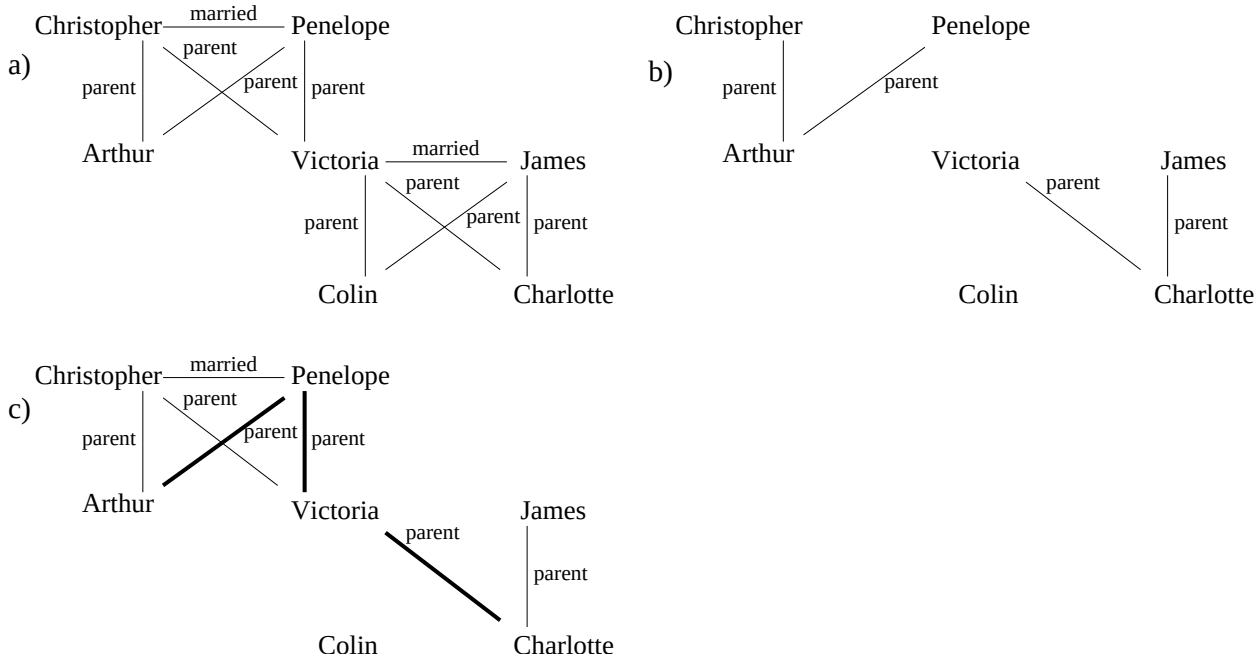


Figure 6.3 (a) Family tree domain from Hinton [66]. (b) Expanded nodes 1 hop away from nodes `Arthur` and `Charlotte`. Since no intersection was found, continue to expand nodes. (c) Expand all paths originating from `Arthur`; `Victoria` is at the intersection of paths from `Arthur` and `Charlotte`, hence a path if found. The bold arcs indicate a path found.

6.3 The Saturated Clause and Hypergraph

The similarities between the properties of a saturated clause and a hypergraph provide a natural mapping from one to the other. A directed hypergraph H is defined by a set of nodes N and a set of hyperarcs H. A hyperarc has a nonempty set of source nodes $S \subseteq N$ linked to a single target node $i \in N$ [11].

A saturated clause can be mapped to a hypergraph in the following way. First, as a saturated clause is formed by a set of literals, it is thus natural to say that each literal L_i is a node in the hypergraph. Second, we observe that a node L_i may need several input arguments, and that each input argument may be provided from a number of literals. Thus, if a node L_i has a set of input variables, I_i, each hyperarc is given by a set of literals generating I_i and L_i. In more detail, a saturated clause maps to a hypergraph as follows:

1. Each node corresponds to a literal.

2. Each hyperarc with $N' \subseteq N$ nodes is generated by a set V of $i' - 1$ variables $V_1, \ldots, V_{i'-1}$ appearing in literals or nodes $L_1, \ldots, L_{i'-1}$. The mapping is such that

   (i) every variable $V_k \in I'_i$ appears as an `output` variable of node L_k

   (ii) every variable V_k appears as argument k in I'_i

Intuitively, the definition says that each node $L_1, \ldots, L_{N'-1}$ generates an input variable for node L'_N. Note that if node L'_N has a single input argument, the hyperarc will reduce to a single arc. Note also that the same variable may appear as different input arguments, or that the same literal may provide different `output` variables. Thus, the size of a hyperarc may be less than the number of input arguments plus 1.

The number of variables and nodes in the saturated clause is finite, therefore, the mapping is also finite and can be computed in finite time. In fact, one can compute the mapping in linear time by keeping reverse edges in the graph, since generating all hyperarcs essentially corresponds to computing the cartesian product of the `output` sets for each input argument.

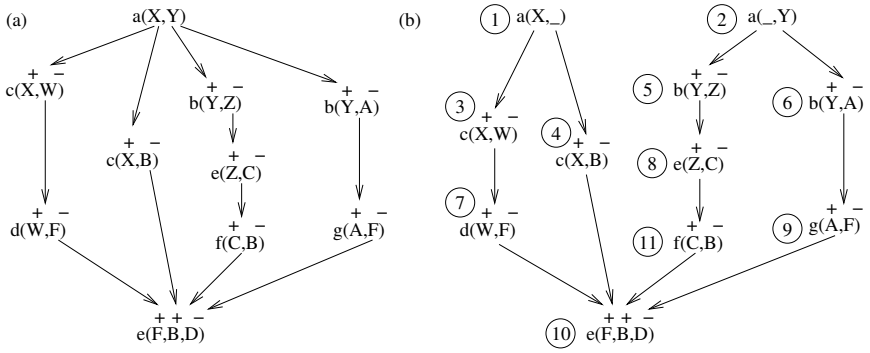


Figure 6.4 (a) Hypergraph of our example saturated clause ($a(X, Y)$ is the head of the saturated clause, '+' indicates `input` variable, '−' indicates `output` variable):
$a(X, Y) \leftarrow c(X, W), c(X, B), b(Y, Z), b(Y, A), d(W, F), e(Z, C), g(A, F), e(F, B, D), f(C, B)$.
(b) Transformation of hypergraph from (a) splits the head literal into its component arguments, which then serve as different sources for the path finding algorithm. The number preceeding each literal indicates the number by which we will refer to the literal in Figure 6.5.

Figure 6.4a shows an example saturated clause and resulting hypergraph. The literal at the root of the graph, $a(X, Y)$, is the head of the saturated clause. Other literals in the graph appear in the body of the saturated clause. All literals of arity 2 have mode $+, -$, that is, `input`, `output`. Literal $e(F, B, D)$, has arity 3 and mode $f(+, +, -)$. Arcs in the figure correspond to dependencies induced by variables, thus there is an arc between $c(X, W)$ and $d(W, F)$ because $d(W, F)$ requires `input` variable W. On the other hand, there is no arc between $c(X, B)$ and $f(C, B)$ since variable B is an `output` variable in both cases.

If a literal has a single input variable, its hyperarc reduces to an arc. This is the case for all literals but $e(F, B, D)$. There are four hyperarcs for node $e(F, B, D)$; they are

$$\{d(W, F), c(X, B), e(F, B, D)\}, \{g(A, F), c(X, B), e(F, B, D)\},$$

$$\{d(W, F), f(C, B), e(F, B, D)\}, \text{ and } \{g(A, F), f(C, B), e(F, B, D)\}.$$

Before we present the path finding algorithm, we first need to present a simple transformation. The graph for a saturated clause is dominated by the seed example, L_0. If the seed example has M arguments, it generates M variables, which we transform as follows:

1. Generate M new nodes L'_j such that each node will have one of the variables in L_0. Each such node will have an `output` variable V_j.

2. Replace the edge between L_0 and some other node induced by the variable V_j by an edge to the new node L'_j.

3. Make each `output` variable M an input variable for L_0.

The transformation can be performed in $O(M)$ time. Figure 6.4b shows this transformation for the hypergraph generated by our example saturated clause. Path generation thus reduces to the problem of finding all hyperpaths that start from nodes $L'_1, \ldots, L'_M$.

6.4 Algorithm

In directed graphs, a path π is a sequence of edges $e_1, e_2, \ldots, e_k$ and nodes $n_1, n_2, \ldots, n_k$, such that $e_i = (n_{i-1}, n_i), 1 \leq i \leq k$. The shortest hyperpath problem is the extension of the classic shortest path problem to hypergraphs. The problem of finding shortest hyperpaths is well known [11, 124] and examples of applications include the grammar problem (of a context free language) [87], linear production systems, transit networks and relational databases [33, 50]. Shortest hyperpath algorithms are used as building blocks of enumerative algorithms for hard combinatorial problems [49]. We do not require optimal hyperpaths; rather, we want to be able to enumerate all possible paths, and we want to do it in the most flexible way, so that we can experiment with different search strategies and heuristics. On the other hand, as one good heuristic is path length, our algorithm should be close to optimal path finding algorithms, such as Knuth's [87].

We present our path finding algorithm through the transformed hypergraph shown in Figure 6.4b. First, we would like to point out that we want to generate paths in the 'pathfinding' sense, which is slightly different from hyperpaths in the graph theorical sense. More precisely, a hyperpath will lead from a node to a set of other nodes in the hypergraph. A *path* is a set of hyperpaths, each starting from a different source node, such that the two hyperpaths have a variable in common. For example, in Figure 6.4b, nodes $\{1, 4\}$ form a hyperpath, and nodes $\{2, 5, 8, 11\}$

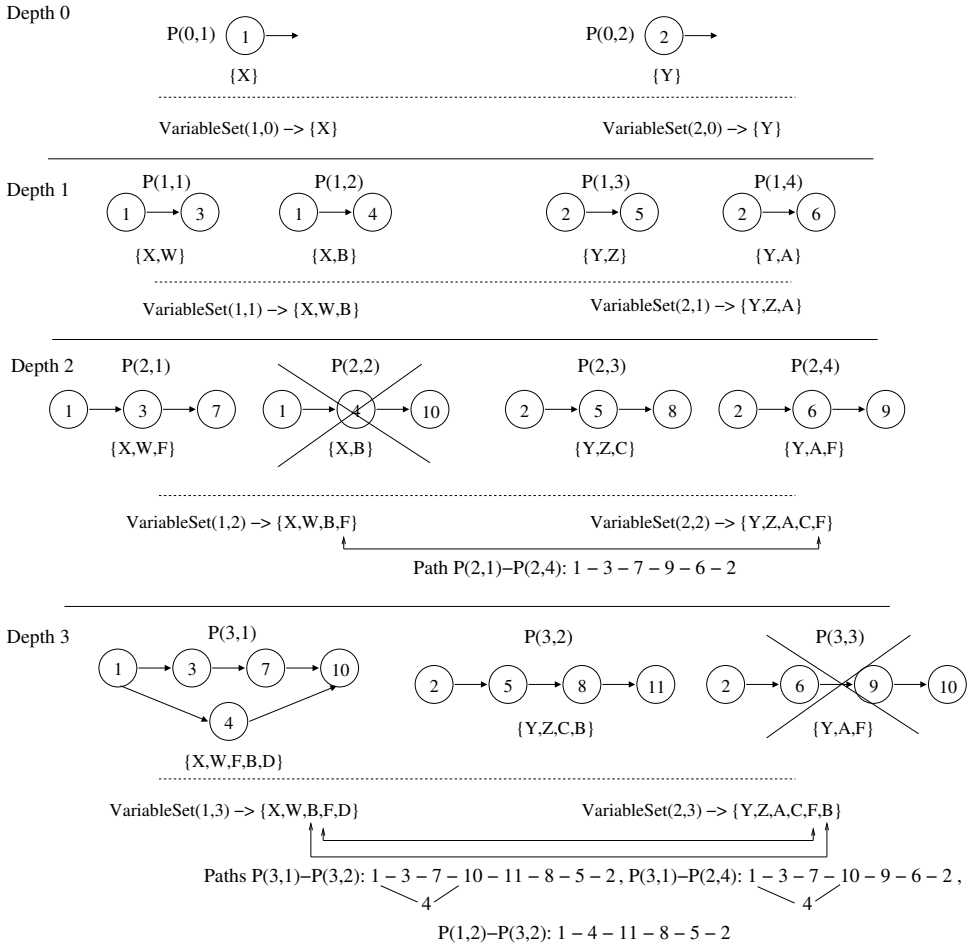


Figure 6.5 Illustration of path finding algorithm using mode declarations. Given transformed hypergraph in Figure 6.4b, which includes information for `input` and `output` variables for each node or literal and $source$ nodes 1 and 2, the algorithm in Figure 6.6 returns a list of paths connecting all input variables. The algorithm iterates by depth, as illustrated above on the left hand side. Current paths are expanded at each depth if the node to be expanded's `input` variables are bound. Otherwise, they are crossed out like $P(2,2)$ and $P(3,3)$. $VariableSet(s,d)$ represents the set of variables reachable from source argument s at depth d. They are used to find common variables between variable sets of different sources at a particular depth. Hyperpaths found so far are indicated by $P(d,n)$, where d indicates the depth and n is the node index at that depth. Paths found are indicated with $P(d,n_1) - P(d,n_2)$ representing the hyperpaths that are connected.

form another hyperpath. Since node 4 and node 11 share variable B, $\{1, 2, 4, 5, 8, 11\}$ form a *path*. Our algorithm *generates paths as combinations of hyperpaths*.

Given hypergraph H, which includes information for `input` and `output` variables for each node or literal, $source$ nodes and desired $maxdepth$ (a function of clause length), the algorithm described in Figure 6.6 as procedure $findPaths$, returns a list of paths connecting all input variables. Figure 6.5 illustrates our path finding algorithm on the example hypergraph in Figure 6.4b. The numbered nodes of graphs in Figure 6.5 correspond to the labels of literals in Figure 6.4b. $VariableSet(s, d)$ represents the set of variables reachable from source argument s at depth d. They are used to find common variables between variable sets of different sources at a particular depth. Hyperpaths found so far are indicated by $P(d, n)$, where d indicates the depth and n is the node index at that depth. Paths found are indicated with $P(d, n_1) - P(d, n_2)$ representing the hyperpaths that are connected.

At each depth the algorithm proceeds as follows:

1. Starting from source node, s

2. Expand paths from previous depth, $d - 1$, only if `input` variables for the newly expanded node, n, exist in the previous variable set $VariableSet(s, d - 1)$ and are bound by parents reachable at the current depth (i.e., all n's `input` variables are bound by parent nodes which contain n's `input` variables.

3. Place all variables reachable from s at current depth d in $VariableSet(s, d)$.

4. Check $VariableSet$ of each source at the current depth for an intersection of variables; for each variable in the intersection create paths from permutations of all nodes containing the variable, including those from previous depths.

Depth 0 corresponds to the initial configuration with our 2 source nodes, 1 and 2, which we represent as hyperpaths $P(0, 1)$ and $P(0, 2)$ respectively. The `input` and `output` variables of source nodes 1 and 2 are placed into their respective variable sets ($VariableSet(1, 0)$ and $VariableSet(2, 0)$). Depth 1 corresponds to the immediate extension for nodes 1 and 2. Node

1 has two hyperpaths of size 2. One to node 3, $P(1,1)$ and the other to node 4, $P(1,2)$. Node 2 can reach nodes 5 and 6 giving hyperpaths $P(1,3)$ and $P(1,4)$. At this step we can reach variables W and B from X, and variables Z and A from Y, indicated by $VariableSet(1,1)$ and $VariableSet(2,1)$. Since we do not have a common variable between both variable sets, we cannot build a path.

Depth 2 corresponds to expanding the hyperpaths that reach nodes $3,4,5,6$. Hyperpath $P(1,1)$ can reach node 7 allowing X to reach variable F. Hyperpath $P(1,2)$ tries to reach node 10, but this hyperpath only contains variable B, whereas node 10 needs both F and B, hence $P(2,2)$ is not included as an expanded path. Hyperpath $P(1,3)$ can be extended with 8, and hyperpath $P(1,4)$ can be extended with node 9. At this point we have hyperpaths from the first argument reaching variables W, B, F, and from the second argument reaching Z, A, C, F. Variable F can be reached with hyperpaths starting at the nodes that define variables X and Y, so we must have a path. We thus find that if we merge hyperpaths $P(2,1)$ and $P(2,4)$ we have our first path: $1,3,7,9,6,2$ $(P(d,n_1) - P(d,n_2))$.

Depth 3 allows one to expand a hyperpath. Hyperpath $P(2,1)$ can reach node 10 by merging with hyperpath $P(1,2)$. This creates the hyperpath $(P(3,1))$ which reaches variables X, W, F, B, D. Hyperpath $P(2,3)$ is expanded to include node 11 but hyperpath $P(2,4)$ cannot be expanded to include node 10 as it does not contain variable B required as an input variable for node 10. Hence $P(3,3)$ is omitted. Now we have two new hyperpaths that can be combined between themselves and with older hyperpaths to generate new paths. Hyperpath $P(3,1)$ reaches variables X, W, F, B, D. We can build a new path by connecting $P(3,1)$ with hyperpath $P(3,2)$, as they both share variable B. $P(3,2)$ can also be connected to $P(2,4)$, as they both share F. Hyperpaths $P(3,2)$ and $P(1,2)$ share variable B, so we can generate the new path $P(3,2) - P(1,2)$. For hyperpaths that are already a path, as they touch X and Y, we can further extend them by merging them with other hyperpaths, obtaining non-minimal paths.

Figure 6.6 Path finding pseudocode: Main routine.

```
1: procedure FINDPATHS(G, source, maxdepth)
2:   expand := ∅; varbydepth := ∅; pathsFound := ∅
3:   maxdepth := max(largest depth of all sources, maxdepth)
4:   for all src ∈ Source do
5:     expand := GETEXPANDLISTBYDEPTH(SRC, VARBYDEPTH, MAXDEPTH)
6:   end for
7:   for d = 1 : maxdepth do
8:     for src1 = 0 : |Source| − 1 do
9:       varSet1 := src1's current depth's var set in varbydepth or last depth available
10:      for src2 = src1 + 1 : |Source| do
11:        if src2 == src1||src1&&src2's last depth <= current depth then continue
12:        end if
13:        varSet2 := src2's current depth's var set or last depth available
14:        for all var ∈ varSet2 do                ▷ iterate through variables for intersection
15:          if var ∈ varSet1 then
16:            for all var that intersect do
17:              pathsFound := permutation of nodes (var ∈ node) from diff depths
18:            end for
19:          end if
20:        end for
21:      end for
22:    end for
23:  end for
24:  return pathsFound
25: end procedure
```

Figure 6.7 Path finding pseudocode: Expanding list by depth.

```
1: procedure GETEXPANDLISTBYDEPTH(source, varbydepth, maxdepth)
2:   expand := ∅                                   ▷ Keep track of expand nodes for each depth
3:   open := source
4:   Add input and output variables of Source to varbydepth at depth 0
5:   while open! = ∅ do
6:     currNode := remove first element of open
7:     for all child reachable from currNode do
8:       if ivars of child is in Varbydepth of the previous depth then
9:         Add ivars and ovars of child to varbydepth at current depth
10:        open := child
11:        if |ivars| of child > 1 then              ▷ if child has > 1 parents bound to each ivar
12:          unbound := child's parent node          ▷ put in bins according to bound var
13:          if open's 1st element > current depth then   ▷ all child's parents found
14:            expand := create permutations of hyperpaths to child
15:          end if
16:        else expand := child
17:        end if
18:      else                                        ▷ ivars of child not in Varbydepth of previous depth
19:        unbound := child's parent node
20:      end if
21:    end for
22:    closed := child
23:  end while
24:  return expand                                  ▷ expanded nodes indexed by depth
25: end procedure
```

Table 6.1 Comparison of the number and length of clauses for Aleph versus Path Finding

Folds	Aleph		Path Finding	
	# Of Clauses	Avg ClauseLength	# Of Clauses	Avg ClauseLength
Theory	2	4.5	2	5
AI	3	3.7	2	5.5
Graphics	3	4	2	6
Languages	1	3	3	7
Systems	1	4	3	5.7

6.5 Experimental Evaluation

Paths can be used in a number of ways. One way is to use Richards and Mooney's method to perform search by generating a number of paths, and then refining them [128, 145]. Alternatively, one can consider the paths found as a source of extra information that can be used to extend the background knowledge (i.e., add paths as background knowledge). In this case, paths can be seen as intensional definitions for new predicates in the background knowledge.

We used the UW-CSE dataset by Richardson and Domingos [129] for a first study of path finding on a heavily relational dataset. The dataset concerns learning whether one entity is advised by other entity based on real data from the University of Washington CS Department. The example distribution are skewed as we have 113 positive examples versus 2711 negative examples. Following the original authors, we divided the data into 5 folds, each one corresponding to a different group in the CS Department. We perform learning in two ways for our control and experiment. In the first approach, we used Aleph to generate a set of clauses. In the second approach, we used path finding to find paths, which are treated as clauses. Further, we allow Aleph to decorate paths with attributes by trying to refine each literal on each path.

We were interested in maximizing performance in the precision recall space. Thus, we extended Aleph to support scoring using the F-measure. The search space for this experiment is

Table 6.2 Test Set Performance (results given as percentage)

Folds	Aleph			Path Finding		
	Recall	Precision	F-measure	Recall	Precision	F-measure
Theory	38	27	32	81	11	19
AI	63	9	16	75	12	21
Graphics	85	46	60	95	20	33
Languages	22	100	36	33	100	50
Systems	87	8	15	82	9	16

relatively small, so we would expect standard Aleph search to find most paths. The two systems do find different best clauses, as shown in Table 6.1 which show both number of clauses and average clause length, including the head literal. Although most of the clauses found by Aleph are paths, the path finding implementation does find longer paths that are not considered by Aleph and also performs better on the training set.

Table 6.2 presents performance on the test set. This dataset is particularly hard as each fold is very different from the other [129], thus performance on the training set may not carry to the testing set. Both approaches perform similarly on the Theory fold. The AI fold is an example where both approaches learn a common rule. Path Finding further finds an extra clause which performs very well on the training set, but badly on the test data. On the other hand, the Languages fold is a different example where both approaches initially found the same clause, but Path Finding goes on to find two other clauses, which in this case resulted in better performance. We were surprised that in both Graphics and Systems Path Finding found good relatively small clauses that were not found by Aleph.

6.6 Chapter Summary

We have presented a novel algorithm for path finding in moded programs. Our approach takes advantage of mode information to reduce the number of possible paths and generate only legal

combinations of literals. Our algorithm is based on the idea that the saturated clause can be represented as a hypergraph, and the use of hyperpaths in this hypergraph to compose the final paths. It is interesting that Path Finding has higher recall than Aleph despite learning longer clauses on average as it is opposite from what one would expect.

This work was originally published in the Proceedings of the Sixteenth European Conference on Machine Learning [107].

Chapter 7

Conclusion

7.1 Modeling Gene Regulatory Pathways

We have reported the first experiment in learning Dynamic Bayesian networks as a means of modeling time series gene expression microarray data, with the aim of gaining insights into regulatory pathways. The prior structure and prior CPTs of our BN and DBN encode background knowledge about gene expression in the organism being modeled, *E. coli*. The experiments provide evidence that BN and DBN with an explicit model of operons are capable of identifying operons in *E. coli* that are in a common regulatory pathway.

There are several directions for further research. First, a larger data set may improve the performance of the approach and allow us to determine whether causality can be determined from time series data. Nevertheless, potentially offsetting any such gain is the need to include additional genes (observed variables) and operons (hidden variables) in the analysis. The present data set used only 169 genes appearing in 142 operons. But the full *E. coli* genome has over 4000 genes, and the predicted operon map has well over 1000 multigene operons.

A second, and perhaps more important, shortcoming of the present work is that computation time did not permit a more encompassing search to be employed. Our full-search algorithm modifies incoming arcs to every hidden node. As the arcs coming into one hidden node are modified, and the CPTs updated, this node may become a better parent for another node. A cascade of such improvements could dramatically improve the fit of the model and hence, potentially, the match of the model with the actual regulatory structure of the organism. Therefore, a crucial direction for further work is to decrease the computation time of the learning algorithm. An approximate

approach to updating CPTs and calculating scores based on local Markov blanket (the parents, children, and children's parents of a node) of operons where the structure changed might speed up the learning process. Alternatively, increasing the efficiency of the implementation of the learning algorithm by parallel execution on a Condor pool [93] can allow the full algorithm to be tested. The faster implementation also will facilitate more extensive experimentation, including cross-validation to estimate the accuracies of expression levels that the model predicts for various genes at various time steps.

Third, we could include a variety of other variables, hidden or observed. Observed variables might include environmental factors such as glucose, tryptophan or temperature. Unobserved variables might include transcription factors and other protein products. In the present work we omitted these because we wanted to see the extent to which changes in the expression levels of a gene could be predicted solely on the basis of changes to other genes, regardless of the environmental causes of those changes.

7.2 Computational Model to Guide the Design of Time Series Experiments

The formal results of our theoretical analysis, on learning DBN structure from time series data, imply the following. First, many short time series experiments are preferable to a single long time series experiment. Second, given the number of time series experiments that can feasibly be run with present-day costs, the number of parents per node in a DBN should be limited to at most three per node, two if possible. Third, even with only two parents per node, the worst-case number of examples required to guarantee a given level of accuracy with a given probability is cubic in the number of variables n, and this number typically is in the thousands. If we are concerned with gaining insight into—or accurate prediction of—only a small number m of the n variables, we can reduce this term to n^2m. This often is the case where we are interested in learning or refining a model of a particular regulatory pathway, and we know most of the key players (genes).

7.3 Scaling FOIL for Multi-Relational Data Mining of Large Datasets

We have presented preliminary methods towards enabling the ILP system FOIL to be applied to multi-relational data mining tasks on large data sets. To deal with insufficient physical memory we have employed relational database management systems. We have implemented FOIL-D, a system that mimics the operation of FOIL but that performs the memory intensive FOIL operations using SQL queries in a relational database. To deal with the slowness of FOIL on scoring, we have FOIL-DH, an extension of FOIL-D that uses histograms to quickly estimate the gains of candidate literals without performing expensive database join operations. We also showed how we use the estimation procedure as a filter to select a small number of candidate literals whose gain we compute exactly. Our experimental results show that while FOIL-DH dramatically reduces the numbers of joins it sometimes fails to learn the correct theories on a set of standard ILP problems because of erroneous estimates. If, however, we use the estimates as a filter, we are able to learn the same correct theories as FOIL-D on the datasets we looked at while performing significantly fewer joins.

7.4 Inferring Regulatory Networks from Time Series Expression Data and Relational Data via Inductive Logic Programming

We concentrated our experiments on learning the DNA damage checkpoint pathway as it is an important pathway that have been implicated in cancer and aging, and because it has been well studied. This pathway plays an important role by responding to single and double-stranded DNA breaks, and is therefore often activated in stressful environments. Hence, it involves a lot of signaling kinases that phosphorylates proteins that are already present within the cell or that only require molecular amounts to trigger a response.

The results we obtained show that our method is quite good at learning important pathway interactions and regulators despite the fact that the data may be noisy or incomplete. This further emphasizes the utility of integrating different data types, since many potential interactions, including those that were not evident from single data sources were identified.

A possible next step will be to perform a comparison with DBNs. We could also explore the larger network of genes that are connected with the core DNA damage checkpoint genes by including more specific background knowledge. This set is likely to include known targets of Dun1 activation, and genes that coordinate the biological processes involved in cell division. It may also include genes heretofore un-implicated in this process, and may provide good starting points for future wet lab experimentation. With a larger search space, we are also interested in improving the search for finding important protein-protein interactions. In the future, we also plan to study other pathways and organisms, incorporate other sources of relational data including knockout data.

7.5 Mode Directed Pathfinding

We have presented a novel algorithm for path finding in moded logic programs. Our approach takes advantage of mode information to reduce the number of possible paths and generate only legal combinations of literals. Our algorithm is based on the idea that the saturated clause can be represented as a hypergraph, and that hyperpaths can be used within this hypergraph to compose the final paths.

Muggleton used a similar intuition in *seed*-based search. His idea is to use a heuristic to classify clauses: a clause's score depends both on coverage as well as the distance the literals in the clause have to each entity [103]. In contrast, our approach is independent of scoring function.

Preliminary results on a medium-sized dataset showed that path finding allows one to consider a number of interesting clauses that would not easily be considered by Aleph. On the other hand, path finding does seem to generate longer clauses, which might be more vulnerable to overfitting. To reduce chances of overfitting, we might try to further restrict the length of the paths generated. In future work we plan to apply this algorithm to larger biological datasets, to learn protein interactions, where path finding is necessary to direct search.

A potentially fruitful research direction for refining pathways is to use a model that combines statistical and relational learning methods. This would allow us to utilize the abundant infomation available from many years of detailed research to infer new connections that cannot be learned from any one source of data alone. Interpolation techniques that model the dynamics of time series

data should also be used to detect global changes rather than just the changes between time steps as this relies too much on an appropriate sampling rate. Also, by interpolating across time steps and then sampling at a consistent rate across all experiments, we can alleviate the problem of combining data with different time step intervals.

7.6 Last Words

This thesis has developed and utilized machine learning techniques for reverse engineering gene regulatory networks from data. The problem of learning gene regulatory networks presents significant challenges to a machine learning system. The models and algorithms presented in this thesis represent substantial progress on this problem. Nonetheless, it is our hope that this work will serve as a foundation for further advances.

The work of this thesis can also be view more abstractly as a set of techniques for applying machine learning to problems characterized by (i) sequential data, (ii) heterogeneous data sources, (iii) complex dependencies and (iv) significant uncertainty. We believe that the work presented here will prove useful to researchers investigating problems which the above characteristics.

LIST OF REFERENCES

[1] Moore. A. and W. Wong. Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 552–559, Washington, DC, 2003.

[2] N. Abe and M.K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[3] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 695–702, 1998.

[4] T. Akutsu, S. Miyano, and S. Kuhara. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. *Proceedings of the Pacific Symposium on Biocomputing*, 4:17–28, 1999.

[5] T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16:727–734, 2000.

[6] 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 U S A*, 96:6745–6750, 1999.

[7] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[8] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2:343–370, 1988.

[9] A. Arkin, J. Ross, and H. H. MacAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected *escherichia coli* cells. *Genetics*, 149:1633–1648, 1998.

[10] A.P. Arkin, P.-D. Shen, and J. Ross. A test case of correlation metric construction of a reaction pathway from measurements. *Science*, 277:1275–1279, 1997.

[11] G. Ausiello, R. Giaccio, G.F. Italiano, and U. Nanni. Optimal traversal of directed hypergraphs. Technical report, Dipartimento di Informatica e Sistemistica, Universit di Roma La Sapienza, 1997.

[12] M.M. Babu, N.M. Luscombe, L. Aravind, M. Gerstein, and S.A. Teichmann. Structure and evolution of transcriptional regulatory networks. *Current Opinion in Structural Biology*, 14:283–291, 2004.

[13] S. Bay, L. Chrisman, A. Pohorille, and J. Shrager. Temporal aggregation bias and inference of causal regulatory networks. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence – Workshop on Learning Graphical Models for Computational Genomics*, pages 58–65, 2003.

[14] H. Blockeel and L. L. De Raedt. Relational knowledge discovery in databases. In *Proceedings of the ICML - 1996 Workshop on Data Mining with Inductive Logic Programming*, 1996.

[15] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101:285–297, 1998.

[16] H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up Inductive Logic Programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3:59–93, 1999.

[17] J. Bockhorst and I.M. Ong. FOIL-D: Efficiently scaling FOIL for multi-relational data mining of large datasets. In *To appear in "Proceedings of the 14th Conference on Inductive Logic Programming"*, Porto, Portugal, 2004. Springer-Verlag.

[18] P. Brockhausen and K. Morik. Directaccess of an ilp algorithm to a database management system. In *Proceedings of the MLnet Familiarization Workshop*, pages 95–110, 1996.

[19] C.H. Bryant, S.H. Muggleton, S.G. Oliver, D.B. Kell, P. Reiser, and R.D. King. Combining inductive logic programming, active learning and robotics to discover the function of genes. *Electronic Transactions in Artificial Intelligence*, 5-B1:1–36, 2001.

[20] E.S. Burnside, J. Davis, V. Santos Costa, I. Dutra, C.E. Kahn Jr., J.P. Fine, and D. Page. Knowledge discovery from structured mammography reports using inductive logic programming. In *In the Proceedings of the American Medical Informatics Association Annual Symposium*, 2005.

[21] M.C. Campi and M. Vidyasagar. Learning with prior information. *IEEE Transactions on Automatic Control*, 46:1682–1695, 2001.

[22] S. Chakrabarti. *Mining the Web*. Morgan Kaufman, 2002.

[23] D.M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, pages 112–128, January 1995.

[24] 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.

[25] D. Cook and L. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15:32–41, 2000.

[26] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118:69–113, 2000.

[27] M. Craven, D. Page, J. Shavlik, J. Bockhorst, and J. Glasner. A probabilistic learning approach to whole-genome operon prediction. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 116–127. AAAI Press, 2000.

[28] S. Dasgupta. The sample complexity of learning fixed-structure Bayesian networks. *Machine Learning*, 29:165–180, 1997.

[29] A. Datta, A. Choudhary, M. L. Bittner, and E. R. Dougherty. External control in markovian genetic regulatory networks. *Machine Learning*, 52:169–191, 2003.

[30] J. Davis, V. Santos Costa, I. M. Ong, D. Page, and I. Dutra. Using Bayesian classifiers to combine rules. In *To appear in the Third Workshop on Multi-Relational Data Mining at the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 2004.

[31] J. Davis, V. Santos Costa, I.M. Ong, D. Page, and I. Dutra. Using bayesian classifiers to combine rules. In *3rd Workshop on Multi-Relational Data Mining (MRDM 2004). In conjunction with KDD 2004*, 2004.

[32] Jesse Davis, Inês Dutra, David Page, and Vítor Santos Costa. Establishing identity equivalence in multi-relational domains. In *Proceedings of the International Conference on Intelligence Analysis*, 2005.

[33] R. De Leone and D. Pretolani. Auction algorithms for shortest hyperpath problems. *SIAM Journal on Optimization*, 11:149–159, 2000.

[34] S. E. Decatur and R. Gennaro. On learning from noisy and incomplete examples. In *Proceedings of the 8th Annual ACM Conference on Computational Learning Theory*, pages 353–360. ACM Press, 1995.

[35] F. Dimaio and J. Shavlik. Speeding up relational data mining by learning to estimate candidate hypothesis scores. In *Proceedings of the ICDM Workshop on Foundations and New Directions of Data Mining*, 2003.

[36] F. Dimaio and J. Shavlik. Learning an approximation to inductive logic programming clause evaluation. In *To appear in "Proceedings of the 14th Conference on Inductive Logic Programming"*, Porto, Portugal, 2004. Springer-Verlag.

[37] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences U S A*, 95:14863–14868, 1998.

[38] J. Ptacek et al. Global analysis of protein phosphorylation in yeast. *Nature*, 438:679–684, 2005.

[39] P. Flach. From extensional to intensional knowledge: Inductive Logic Programming techniques and their application to deductive databases. In B. Freitag, H. Decker, M. Kifer, and A. Voronkov, editors, *Transactions and Change in Logic Databases*, volume 1472, pages 356–387. Springer-Verlag, 1998.

[40] N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303:799–805, 2004.

[41] N. Friedman and L. Getoor. Efficient learning using constrained sufficient statistics. In *Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, 1999.

[42] N. Friedman and D. Koller. Being Bayesian about network structure. a Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–125, 2003.

[43] N. Friedman, I. Nachman, and D. Pe'er. Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 206–215, San Francisco, CA, 1999. Morgan Kaufmann.

[44] Nir Friedman. The Bayesian structural EM algorithm. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 129–138, Madison, WI, 1998. Morgan Kaufmann.

[45] 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.

[46] Nir Friedman, Kevin Murphy, and Stuart Russell. Learning the structure of dynamic probabilistic networks. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 139–147. Morgan Kaufmann, 1998.

[47] S. Fröhler and S. Kramer. Inductive logic programming for gene regulation prediction. In *Proceedings of the 16th International Conference on Inductive Logic Programming*, pages 83–85, Santiago de Compostela, Spain, 2006. University of Corunna.

[48] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1995 Int'l Workshop. on Knowledge Discovery and Deductive and Object-Oriented Databases*, pages 39–46, 1995.

[49] G. Gallo, C. Gentile, D. Pretolani, and G. Rago. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Math Programming*, 80:213–237, 1998.

[50] G. Gallo and M.G. Scutella. Directed hypergraphs as a modelling paradigm. Technical report, Dipartimento di Informatica, Universitá di Pisa, 1999.

[51] A.P. Gasch, M. Huang, S. Metzner, D. Botstein, S.J. Elledge, and P.O. Brown. Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. *Mol Biol Cell.*, 12:2987–3003, 2001.

[52] A.P. Gasch, P.T. Spellman, C.M. Kao, O. Carmel-Harel, M.B. Eisen, G. Storz, D. Botstein, and P.O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Mol Biol Cell.*, 11:4241–57, 2000.

[53] Oak Ridge National Laboratory Genome Management Information System. $http : //genomics.energy.gov/gallery/systems_biology/detail.np/detail - 09.html.$

[54] Oak Ridge National Laboratory Genome Management Information System. $http : //genomics.energy.gov/gallery/basic_genomics/detail.np/detail - 27.html.$

[55] Oak Ridge National Laboratory Genome Management Information System. $http : //genomics.energy.gov/gallery/gtl/detail.np/detail - 50.html.$

[56] Lise Getoor. Link mining: A new data mining challenge. *SIGKDD Explorations*, 5:84–89, 2003.

[57] S. Goldman. Can PAC learning algorithms tolerate random attribute noise? *Algorithmica*, 14:70–84, 1995.

[58] U. Güldener, M. Münsterkötter, G. Kastenmüller, N. Strack, J. van Helden, C. Lemer, J. Richelles, S.J. Wodak, J. Garcia-Martinez, J.E. Perez-Ortin, H. Michael, A. Kaps, E. Talla, B. Dujon, B. Andre, J.L. Souciet, J. De Montigny, E. Bon, C. Gaillardin, and H.W. Mewes. CYGD: the Comprehensive Yeast Genome Database. *Nucleic Acids Research*, 33:D364–8, 2005.

[59] E. Gyftodimos and P. Flach. Hierarchical Bayesian networks: an approach to classification and learning for structured data. *Proceedings of the ECML/PKDD - 2003 Workshop on Probablistic Graphical Models for Classification*, pages 25–36, 2003.

[60] J.C. Harrison and J.E. Haber. Surviving the breakup: The DNA damage checkpoint. *Annu. Rev. Genet.*, 40:209–235, 2006.

[61] A.J. Hartemink, D.K. Gifford, T.S. Jaakkola, and R.A. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 422–433, 2001.

[62] R.F. Hashimoto, S. Kim, I. Shumulevich, W. Zhang, M.L. Bittner, and E.R. Dougherty. Growing genetic regulatory networks from seed genes. *Bioinformatics Advance Access*, 2004.

[63] D. Heckerman. A Bayesian approach to learning causal networks. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 285–295, Montreal, Quebec, 1995. Morgan Kaufmann.

[64] D. Heckerman, C. Meek, and G. Cooper. A Bayesian approach to causal discovery. In C. Glymour and G. Cooper, editors, *Computation, Causation, and Discovery*, pages 141–166. AAAI Press, Menlo Park, CA, 1999.

[65] G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Fifth International Joint Conference on Artificial Intelligence*, pages 356–362, Amherst, MA, 1986. Lawrence Erlbaum.

[66] G.E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst, MA, 1986. Erlbaum.

[67] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, Edmonton, CA, 2002. ACM Press.

[68] G. Hulten, P. Domingos, and Y. Abe. Mining massive relational databases. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence – Workshop on Learning Statistical Models from Relational Data*, Acapulco, Mexico, 2003.

[69] L. Hunter. Life and its molecules: A brief introduction. *AI Magazine*, 25:9–22, 2004.

[70] D. Husmeier. Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks. *Bioinformatics*, 19:2271–2282, 2003.

[71] T. Ideker, V. Thorsson, J.A. Ranish, R. Christmas, J. Buhler, J.K. Eng, R. Bumgarner, D.R. Goodlett, R. Aebersold, and L. Hood. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292:929–934, 2001.

[72] 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.

[73] J. Ihmels, S. Bergmann, and N. Barkai. Defining transcription modules using large-scale gene expression data. *Bioinformatics*, 2004.

[74] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD Conference*, pages 233–244, 1995.

[75] Y. E. Ioannidis and V. Poosala. Histogram-based solutions to diverse database estimation problems. *IEEE Data Eng. Bull.*, 18:10–18, 1995.

[76] Ptacek J. and Snyder M. Charging it up: global analysis of protein phosphorylation. *Trends in Genetics*, 22:545–554, 2006.

[77] David Jensen. Prospective assessment of AI technologies for fraud detection: A case study. In *Working Papers of the AAAI-97 Workshop on Artificial Intelligence Approaches to Fraud Detection and Risk Management*, 1997.

[78] P.D. Karp, M. Riley, M. Saier, I.T. Paulsen, Collado-Vides J., S.M. Paley, A. Pellegrini-Toole, C. Bonavides, and S. Gama-Castro. The EcoCyc Database. *Nucleic Acids Research*, 30:56–58, 2002.

[79] Kearns and Schapire. Efficient distribution-free learning of probabilistic concepts. In S. J. Hanson, G. A. Drastal, and R. L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospect*, volume 1. MIT Press, Bradford, MA, 1994.

[80] M. Kearns and M. Li. Learning in the presence of malicious errors. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, pages 267–280, Chicago, IL, 1988. ACM Press.

[81] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of Boolean formulae. In *Proceedings of the 19th annual ACM conference on Theory of computing*, pages 285–295, New York, NY, 1987. ACM Press.

[82] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, 1994.

[83] A. Khodursky, B. J. Peter, N. R. Cozzarelli, D. Botstein, P. O. Brown, and C. Yanofsky. Dna microarray analysis of gene expression in response to physiological and genetic changes that affect tryptophan metabolism in *escherichia coli*. *Proceedings of the National Academy of Science USA*, 97(22):12170–12175, 2000.

[84] S.V. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic Bayesian networks. *Briefings in Bioinformatics*, 4:228–235, 2003.

[85] R.D. King, K.E. Whelan, F.M. Jones, P.J.K. Reiser, C.H. Bryant, S. Muggleton, D.B. Kell, and S Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.

[86] H. Kitano. Computational systems biology. *Nature*, 420:206–210, 2002.

[87] D.E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6:1–5, 1997.

[88] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the Boolean network model. *Machine Learning*, 52:147–167, 2003.

[89] N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[90] S. Liang, S. Fuhrman, and R. Somogyi. Reveal: a general reverse engineering algorithm for inference of genetic network architectures. In *In Proceedings of the Pacific Symposium Biocomputing 1998*, pages 18–29, 1998.

[91] G. Lindner and K. Morik. Coupling a relational learning algorithm with a database system. In *Workshop Notes of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 163–168, 1995.

[92] F. A. Lisi and D. Malerba. Inducing multi-level association rules from multiple relation. *Machine Learning*, 55:175–210, 2004.

[93] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, Washington, DC, 1988. IEEE Computer Society.

[94] A. Martinez-Antonio, H. Salgado, S. Gama-Castro, R.M. Gutierrez-Rios, V. Jimenez-Jacinto, and J. Collado-Vides. Environmental conditions and transcriptional regulation in *escherichia coli*: a physiological integrative approach. *Biotechnology and Bioengineering*, 84:743–749, 2003.

[95] H.H. McAdams and A.P. Arkin. Simulation of prokaryotic genetic circuits. *Annual Review of Biophysics and Biomolecular Structure*, 27:199–224, 1998.

[96] D. McFarlin and D. Page. Modeling Ras signal transduction in normal and transformed cells with exclusively bi-directional reaction equations, 2004.

[97] R. S. Michalski, I. Mozetič, J. Hong, and N. Lavrač. The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, Philadelphia, PA, 1986. Morgan Kaufmann.

[98] M. Middendorf, A. Kundaje, C. Wiggins, Y. Freund, and C. Leslie. Predicting genetic regulatory response using classification. *Bioinformatics*, 20:i232–240, 2004.

[99] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[100] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[101] R. J. Mooney, P. Melville, L. R. Tang, J. Shavlik, I. Dutra, D. Page, and V. Santos Costa. Relational data mining with inductive logic programming for link discovery. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Data Mining: Next Generation Challenges and Future Directions*, volume To appear. AAAI Press, 2004.

[102] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[103] S.H. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[104] K. Murphy. Bayes net toolbox http://www.cs.berkeley.edu/˜murphyk/bayes/bnt.html.

[105] K. Murphy and S. Mian. Modelling gene expression data using dynamic Bayesian networks. Technical report, Computer Science Division, University of California, Berkeley, 1999.

[106] K. Noto and M. Craven. Learning regulatory network models that represent regulator states and roles. *First Annual RECOMB Satellite Workshop on Regulatory Genomics. To appear in Lecture Notes in Bioinformatics*, 2004.

[107] I.M. Ong, I. Dutra, D. Page, and V. Santos Costa. Mode directed path finding. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 673–681, 2005.

[108] 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.

[109] I.M. Ong, S.E. Topper, D. Page, and V. Santos Costa. Inferring regulatory networks from time series expression data and relational data via inductive logic programming. In *Proceedings of the Sixteenth International Conference on Inductive Logic Programming*, 2006.

[110] D. Page and I.M. Ong. Experimental design of time series data for learning from dynamic bayesian networks. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 267–278, 2006.

[111] David Page and Mark Craven. Biological applications of multi-relational data mining. *SIGKDD Explorations*, 5:69–79, 2003.

[112] C. Palmer, P. Gibbons, and C. Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[113] B.Ø. Palsson. The challenges of in silico biology. *Nature Biotechnology*, 18:1147–1150, 2000.

[114] I. Papatheodorou, A. Kakas, and M. Sergot. Inference of gene relations from microarray data by abduction. In *Eighth International Conference on Logic Programming and Non-Monotonic Reasoning*. Springer, 2005.

[115] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1992.

[116] D. Pe'er, A. Regev, G. Elidan, and N. Friedman. Inferring subnetworks from perturbed expression profiles. In *Proceedings of the 9th International Conference on Intelligent Systems for Molecular Biology*, pages S215–S224. Oxford University Press, 2001.

[117] 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.

[118] B. E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. D'Alche-Buc. Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, 19:II138–II148, 2003.

[119] L. Pitt. On exploiting knowledge and concept use in learning theory. In M. Li and A. Maruoka, editors, *Proceedings of the Eighth International Workshop on Algorithmic Learning Theory*, volume 1316, pages 62–84. Springer-Verlag, 1997.

[120] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3:131–169, 1999.

[121] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[122] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pages 3–20, Vienna, Austria, 1993.

[123] R. Ramakrishnan. *Database Management Systems*. McGraw-Hill, New York, 1998.

[124] G. Ramalingam and Thomas W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21:267–305, 1996.

[125] C.V. Rao and A.P. Arkin. Control motifs for intracellular regulatory networks. *Annual Review of Biomedical Engineering*, 3:391–419, 2001.

[126] S. Ray and D. Page. Multiple instance regression. In *Proceedings of the 17th International Conference on Machine Learning*, pages 425–432, San Francisco, CA, 2001. Morgan Kaufmann.

[127] P.G.K. Reiser, R.D. King, D.B. Kell, S.H. Muggleton, C.H. Bryant, and S.G. Oliver. Developing a logical model of yeast metabolism. *Electronic Transactions in Artificial Intelligence*, 5-B2:223–244, 2001.

[128] Bradley L. Richards and Raymond J. Mooney. Learning relations by pathfinding. In *National Conference on Artificial Intelligence*, pages 50–55, 1992.

[129] M. Richardson and P. Domingos. Markov logic networks. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2004.

[130] D. Ron, Y. Singer, and N. Tishby. Learning probabilistic automata with variable memory length. In *Proceedings of the 7th Annual Conference on Learning Theory*, pages 35–46, 1994.

[131] F.P. Roth, J.D. Hughes, P.W. Estep, and G.M. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mrna quantitation. *Nature Biotechnology*, 16:939–945, 1998.

[132] C. Rouveirol. Flattening and saturation: Two representation changes for generalization. *Machine Learning*, 14:219–232, 1994.

[133] H. Salgado, S. Gama-Castro, A. Martinez-Antonio, E. Diaz-Peredo, F. Sanchez-Solano, M. Peralta-Gil, D. Garcia-Alonso, V. Jimenez-Jacinto, A. Santos-Zavaleta, C. Bonavides-Martinez, and Collado-Vides J. Regulondb (version 4.0): transcriptional regulation, operon organization and growth conditions in *escherichia coli* k-12. *Nucleic Acids Research*, 32:D303–306, 2004.

[134] H. Salgado, A. Santos, U. Garza-Ramos, J. van Helden, E. Diaz, and J. Collado-Vides. Regulondb (version 2.0): a database on transcriptional regulation in *escherichia coli*. *Proceedings of the National Academy of Science USA*, 27:59–60, 1999.

[135] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270:467–470, 1995.

[136] B. Schoeberl, C. Eichler-Jonsson, E. D. Gilles, and G. Muller. Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. *Nature Biotechnology*, 20:370–375, 2002.

[137] E. Segal, Y. Barash, I. Simon, N. Friedman, and D. Koller. From promoter sequence to expression: A probabilistic framework. In *Proceedings of the 6th International Conference on Research in Computational Molecular Biology*, Washington, DC, 2002.

[138] E. Segal, D. Pe'er, D. Koller, and N. Friedman. Learning module networks. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, 2003.

[139] E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: Identifying regulatory modules and their condition specific regulators from gene expression data. *Nature Genetics*, 34:166–176, 2003.

[140] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, 17:S243–252, 2001.

[141] E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19:i264–i272, 2003.

[142] S.S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of *escherichia coli*. *Nature Genetics*, 31:64–68, 2002.

[143] I. Shmulevich, E. R. Dougherty, K. Seungchan, and W. Zhang. Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18:261–274, 2002.

[144] I. Shmulevich, A. Saarinen, O. Yli-Harja, and J. Astola. Inference of genetic regulatory networks under the best-fit extension paradigm. In W. Zhang and I. Shmulevich, editors, *Computational and Statistical Approaches To Genomics*. Kluwer Academic Publishers, Boston, 2002.

[145] S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 38–52. Springer Verlag, 1998.

[146] A. Srinivasan. *The Aleph Manual*. University of Oxford, 2001.

[147] A. Srinivasan, R.D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis Predictions Using ILP. In *Proceedings of Inductive Logic Programming*, pages 273–287, 1997.

[148] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 34:D535–539, 2006.

[149] J. Stark, D. Brewer, M. Barenco, D. Tomescu, R. Callard, and M. Hubank. Reconstructing gene networks: what are the limits? *Biochemical Society Transactions*, 31:1519–1525, 2003.

[150] M. Steffen, A. Petti, J. Aach, P. D'haeseleer, and G. Church. Automated modelling of signal transduction networks. *BMC Bioinformatics*, 3:34–44, 2002.

[151] M. Stonebraker and G. Kemnitz. The postgres next-generation database management system. *Communications of the ACM*, 34:78–92, 1991.

[152] J. Struyf, S. Dzeroski, H. Blockeel, and A. Clare. Hierarchical multi-classification with predictive clustering trees in functional genomics. In *Progress in Artificial Intelligence: 12th Portugese Conference on Artificial Intelligence*, pages 272–283. Springer, 2005.

[153] J. M. Stuart, E. Segal, D. Koller, and S. K. Kim. A gene-coexpression network for global discovery of conserved genetic modules. *Science*, 302:249–255, 2003.

[154] A. Tamaddoni-Nezhad, R. Chaleil, A. Kakas, and S.H. Muggleton. Application of abductive ILP to learning metabolic network inhibition from temporal data. *Machine Learning*, 64:209–230, 2006.

[155] A. Tanay and R. Shamir. Computational expansion of genetic networks. In *Bioinformatics*, volume 17, 2001.

[156] L. R. Tang, R. J. Mooney, and P. Melville. Scaling up ilp to large examples: Results on link discovery for counter-terrorism. In *Proceedings of the KDD-2003 Workshop on Multi-Relational Data Mining*, pages 107–121, Washington, DC, 2003.

[157] M.C. Teixeira, P. Monteiro, P. Jain, S. Tenreiro, A.R. Fernandes, N.P. Mira, M. Alenquer, A.T. Freitas, A.L. Oliveira, and I. S-Correia. The YEASTRACT database: a tool for the analysis of transcription regulatory associations in *Saccharomyces cerevisiae*. *Nucleic Acids Research*, 34:D446–451, 2006.

[158] Z. Tu, Li. Wang, M.N. Arbeitman, T. Chen, and F. Sun. An integrative approach for causal gene identification and gene regulatory pathway inference. *Bioinformatics*, pages e489–e496, 2006.

[159] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[160] D.M. Wolf and A.P. Arkin. Motifs, modules and games in bacteria. *Current Opinion in Microbiology*, 6:125–134, 2003.

[161] S. Yamada, T. Taketomi, and A. Yoshimura. Model analysis of difference between EGF pathway and FGF pathway. *Biochemal and Biophysical Research Communications*, 314:1113–1120, 2004.

[162] X. Yang and E. E. Ishiguro. Involvement of the n terminus of ribosomal protein l11 in regulation of the rela protein of *escherichia coli*. *Journal of Bacteriology*, 183(22):6532–6537, 2001.

[163] E. Yeger-Lotem, S. Sattath, N. Kashtan, S. Itzkovitz, R. Milo, R.Y. Pinter, U. Alon, and H. Margalit. Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction. *Proceedings of the National Academy of Sciences U S A*, 101:5934–5939, 2004.

[164] C. Yoo, V. Thorsson, and G.F. Cooper. Discovery of causal relationships in a gene-regulation pathway from a mixture of experimental and observational dna microarray data. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 498–509, 2002.

[165] L. You, P.F. Suthers, and J. Yin. Effects of *escherichia coli* physiology on growth of phage T7 in vivo and in silico. *Journal of Bacteriology*, 184:1888–1894, 2002.

[166] A. Zien, R. Kuffner, R. Zimmer, and T. Lengauer. Analysis of gene expression data with pathway scores. *Proc Int Conf Intell Syst Mol Biol*, 8:407–17, 2000.