PROBABILISTIC MODELING OF GENE REGULATORY NETWORKS FROM DATA

Thesis submitted for the degree
"Doctor of Philosophy"
by

Iftach Nachman

Submitted to the Senate of the Hebrew University
November 2004

This work was carried out under the supervision of
Prof. Nir Friedman

Acknowldegements

There are many people to whom I owe many thanks for helping me going through this long process of completing a PhD. First, to my adviser Nir Friedman. During the six years I worked with Nir, not only did I learn from him a whole lot, but I was also inspired by his capabilities: Developing a birds-eye view, getting to the interesting core of a problem, passing hurdles without making an issue, being very generous to his students - all these, and more, I hope I have learned from him over those years.

I had the pleasure of collaborating with several people along my work: Michal Lineal who was our first true biological collaborator, also provided support and help along the way like a caring mother. Zohar Yakhini and Amir Ben-Dor, from whom I learned the art of cross-atlantic collaborative work. Dana Pe'er, who shared with me the first steps of this research. Gal Elidan, my collaborator in the last paper of the PhD - it was a great fun! Aviv Regev, not only an excellent collaborator and a great scientist but also a gracious host in my new work place.

Many thanks to all the other lab members: Matan,Ori,Yoseph,Tommy, Yuval, Hillel, Noa, Ariel, Ilan and Omri. Many of you contributed from your time and efforts more than once when a pass on a manuscript or a presentation practice run was needed. You do not have to watch your back anymore when mentioning discrete variables! Gil Bejerano and Elon Portugaly, my office mates, for making the hours there much more fun.

Many thanks to Alisa Shadmi from the ICNC. You were like a caring mother during this period, and were always happy to help out of bureaucratic traps. Ruti Suchi and Regina Krizhanovsky also helped many times with administrative issues. I am grateful to those who gave me bread during the PhD years: the ICNC program masters scholarship, and the Horowitz fund for its generous scholarship.

Finally, I want to thank my family. My parents Ovadia and Zipora and my brother Yuval for constant interest and support in all ways. My wife Orit who went with me through all this journey with love, and my children Yoav and Michal who were born into it - you made it all worthwhile.

I dedicate this work to the memory of my grandmother, Hofshia, who followed it from its first day to the very last.

Abstract

The living cell consists of a complex system of interacting networks, involved in signal transduction, metabolism, and regulation. Regulatory networks can be thought of as the core brain, or master plan, controlling and operating all the functions of the cell. In this dissertation we try to gain better understanding of such networks by analyzing experimental data. Our approach is based on probabilistic generative modeling of experimental observations.

The challenge of learning about regulatory networks includes both unraveling their structure (a qualitative aspect) and the precise nature of local interactions in them (a quantitative aspect). Both these aspects are important for understanding how these networks work, what dynamic behaviour they display under different conditions and what is their expected response to specific stimuli. New experimental methods that have emerged in the past few years have made the exploration of such networks possible. These methods include parallel measurement of mRNA abundance, or alternatively regulator binding location, for many thousands of genes simultaneously. However, learning from such data sources poses several challenges. First, both the biological processes that generated the observed data and the processes involved in measuring those phenomena involve stochastic aspects, and are therefore not deterministic, nor exactly replicable. Second, most of the stages and quantities in the regulatory pathways are not observed, and many of the design details are not known. Finally, current data sets are sparse, while the number of participating genes in those networks can be very large.

The model-based approach we present tries to describe how such data sets were generated. To cope with the above challenges, it has to account for stochasticity at all levels, to allow the use of unobserved entities, and to be able to search for the "correct" design diagram. In the theoretical part of this work we present our choice of modeling language, which is based on probabilistic graphical models, as well as novel representation schemes, model assessment methods and search algorithms which are needed to adapt this language to our task, coping with its specific difficulties. We then present two approaches for using these tools to learn regulatory networks from data.

We first introduce a novel representation scheme for Bayesian networks based on Gaussian processes priors. These priors are semi-parametric in nature and can learn almost arbitrary noisy functional relations. We develop the Bayesian score of Gaussian Process Networks and describe how to learn them from data. We present empirical results on artificial and real-life domains with non-linear dependencies.

Next, we introduce two novel methods for learning the structure of Bayesian networks, which address specific difficulties: The first enables handling very large domains, and the second allows learning continuous variable networks with non-linear interactions as well as handling new hidden variables. The *Sparse Candidate* algorithm achieves faster learning by restricting the search space. It iteratively restricts the parents of each variable to belong to a small subset of candidates, and then searches for a network that satisfies these constraints. We evaluate this algorithm both on synthetic

and real-life data. Our results show that it is significantly faster than alternative search procedures without loss of quality in the learned structures. The *Ideal Parent* method is a general method for significantly speeding the structure search for continuous variable networks with common parametric distributions. Importantly, this method facilitates the efficient addition of new hidden variables into the network structure. We demonstrate the method on several data sets, both for learning structure on fully observable data, and for introducing new hidden variables during structure search.

In the second part of the thesis we apply the tools from probabilistic modeling for the task of learning regulatory networks from gene expression data. We begin by introducing the first application of Bayesian networks for modeling gene regulatory networks. We show how such networks can describe interactions between genes. We then describe a method for recovering gene interactions from microarray data using Bayesian network learning techniques, and demonstrate the method on real data from yeast.

Finally, we introduce a more realistic modeling approach. Unlike most previous works in the field, here we employ *quantitative* transcription rates, and simultaneously estimate both the kinetic parameters that govern these rates, and the activity levels of unobserved regulators that control them. We apply our approach to expression data sets from yeast and E. Coli and show that we can learn the unknown regulator activity profiles, as well as the binding affinity parameters. We also show how the "Ideal Parent" method enables us to improve initial guesses of regulation topology, as well as reconstruct *ab initio* the regulatory network from those data sets.

Contents

Chapter 1

Introduction

The living cell can be viewed as a complex system of interacting networks. These networks can be roughly divided to three types. Signal transduction networks can be thought of as the cell's input/output interfaces to the outside world. Metabolic networks take care of matter and energy. Regulatory networks are the core brain, or master plan, controlling and operating all the functions of the cell. The regulation of genetic expression lies at the heart of biological variety, accounting for differences between cell types (e.g., blood, muscle, neural) in multi-cell organisms, as well as different cell states and processes (e.g., differentiation, mating, replication, sugar metabolism) in organisms of all levels. Moreover, it drives almost all dynamic processes occurring in the cell. In this dissertation we try to gain better understanding of cellular networks, in particular gene regulatory networks, by analyzing experimental data.

The genomic era brought massive amounts of data and new knowledge, and also opened the path to exploring many questions which were not accessible up to now. The accumulating sequencing projects, along with progress in genome annotation, are clearing up on a fundamental question for many organisms: what is the cell's working gene set? This set can be thought of as the set of building blocks and working tools available for manufacturing in each cell. These blocks and tools must be produced at the right times and in the right quantities for the cell to carry its essential processes. Once this set is determined, we can turn to the next level of questions, such as:

- What is the function (or functions) of the proteins coded by each gene?
- In what processes or structures does it participate?
- Under which circumstances is this gene activated?
- What mechanisms regulate its expression level and how?

On a larger scale, we can ask questions on the networks involving the cell's genes: What is the structure of those networks? What are the parameters governing the interactions in them? What dynamics do these networks follow?

New experimental methods that have emerged in the past few years have made the exploration of these questions possible. Most notably, methods for the parallel measurement of the expression of

thousands of genes now supply snapshots of the whole cell transcriptome under specific conditions, or at a specific stage of development (DeRisi et al., 1997; Lockhart et al., 1996; Wen et al., 1998). A derived methodology can also detect all the locations in the DNA to which a given transcription factor binds. Data from these two types of experiments allows us to try and reconstruct regulatory networks, even without any prior knowledge.

Of course, extracting this knowledge is not straightforward. First, we need a language and methodology for turning this wealth of data into useful models of regulatory networks. Second, we must overcome difficulties such as measurement errors, stochastic biological behaviour, partial observations, a limited number of experiments and a large number of modeled entities (i.e. genes).

We now give a brief description of the biology behind genetic regulation, followed by some of the experimental and analysis methods used to decipher them.

1.1 The Biology of Gene Regulation

Living cells comprise the lowest living organisms (such as bacteria and yeast), and are the building blocks of higher organisms. Each cell is composed of a lipid membrane sac, containing many diverse molecules which execute the cell's functions and constitute its structure. The majority of those molecules are proteins, and the main difference between cells of the same organism performing different functions is in their protein content, or *proteome*. How is this variety achieved?

According to the central dogma of molecular biology, each cell of a given species contains the same blueprint for manufacturing proteins. This blueprint comes in the form of long *DNA* molecules, organized in one or more chromosomes, and is replicated each time a cell divides. There are two main steps in the production of proteins: First, *RNA* molecules are transcribed from a region on the DNA molecule called a *gene*, which encodes the protein. Second, these *RNA* molecules are processed, exported out of the nucleus (in eukaryotic organisms), and are then *translated* into proteins. This flow of information is depicted in Figure 1.1. The quantity of protein molecules, as well as their activity state, are both subject to regulation occurring at different stages of this process. The differences in regulation between different genes, and under different conditions creates the variety and dynamics in the cell. Transcription initiation is mostly regulated by *transcription factors*. These factors are proteins themselves, which bind to specific binding sites on the DNA region upstream of the regulated gene (called the gene's *promoter*). Some of these factors act as *activators*, while others act as *repressors* of transcription. For many genes, the relevant transcription factors display combinatorial effects. For example, the combination of two activators might be needed to initiate transcription. A third factor, if bound, might repress this activation. That same factor might act as an activator for different genes.

Other modes of regulation include the control of the DNA structure. For a gene to be transcribed, the surrounding DNA should be accessible to the transcription machinery. The dynamic changing of this structure (called chromatin remodeling), plays an important role, though less understood, of transcription regulation. Other regulation mechanisms include the active degradation of mRNA

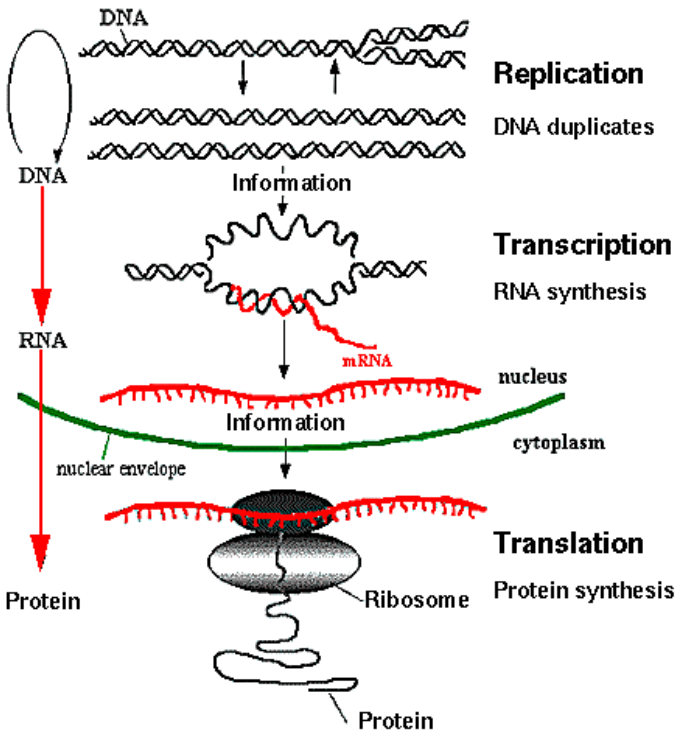


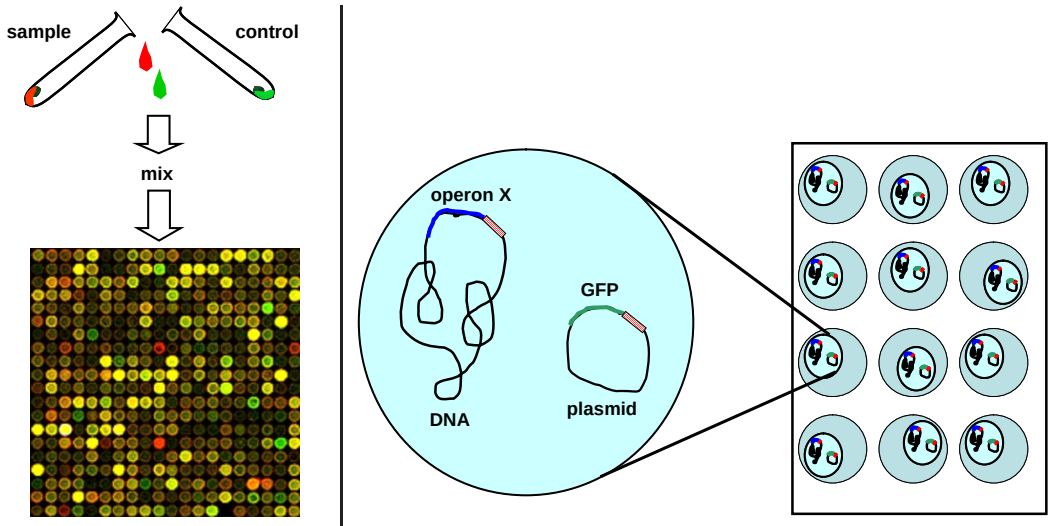
Figure 1.1: The central dogma of molecular biology. Information flows from DNA to RNA by a transcription process, and from RNA to protein by translation. (Image taken from `http://bass.bio.uci.edu/ hudel/bs99a/lecture20/central2.gif`)

or protein products. The completed protein may be subject to further regulation, applied to its localization in the cell, and its modification through addition or removal of chemical groups.

If we take into account that most of the machinery responsible for specific regulation is composed of proteins, and that all proteins are themselves subject to such processes of regulation, the emerging picture is of a complex network of regulation interactions, governing the cell state and dynamics.

1.2 Experimental Methods

To study the mechanisms and structure of regulation, we need experimental methods to measure the products of the regulation path at different stages. Specifically, we want to measure the protein and/or mRNA molecule quantities of specific genes. These quantities are referred to as the gene's protein or mRNA *expression levels*, and the whole process of regulation is generally called *gene expression regulation*. Up to a few years ago, experimentalists who wished to examine some of these regulation processes could only measure the expression levels of few individual genes by one of several methods. The biggest breakthrough of recent years was the introduction of methods



(a) cDNA microarray method (b) Reporter plasmid method

Figure 1.2: mRNA expression measurement methods. (a) cDNA microarray method. Equal amounts of sample (tagged red) and control (tagged green) extracts are poured onto the chip. The ratio of green to red in each probe indicates how much that gene is over or under-expressed. (b) Reporter plasmid method. Each well holds a different E.Coli strain, containing a plasmid with a reporter GFP gene. In each strain the GFP is preceded by a promoter of a different operon. The rate of GFP production is indicative of the promoter activity, and therefore the transcription rate.

which can measure such expression levels in parallel, for many hundreds and even thousands of genes. The usage of these methods has seen an exponential growth since their introduction.

We now briefly describe two approaches for measuring mRNA expression levels. Data sets from these methods are the main data source used in this thesis. We note that methods for measuring protein levels in parallel have also been developed (see Haynes and Yates (2000) for a review), but the availability of large scale data sets from these methods is much more limited.

1.2.1 cDNA Microarrays

In recent years, technical breakthroughs in spotting hybridization probes and advances in genome sequencing efforts lead to development of *DNA microarrays* (DeRisi et al., 1997; Lockhart et al., 1996). These consist of many species of probes, either oligonucleotides or cDNA, that are immobilized in a predefined organization to a solid phase.

The basis of these methods is the phenomena of *hybridization*: a single strand RNA (or DNA) molecule will hybridize under the right conditions to a molecule containing the reverse complement sequence. Each *probe*, therefore, contains many copies of the reverse complement of one or more segments of a specific gene. The extracted mRNA from the studied sample is reverse-transcribed into coding DNA (cDNA) molecules, which are then poured onto the microarray. The cDNA molecules of each gene then hybridize to the corresponding probe. The hope is that the

amount of hybridized molecules in the probe will be proportional to their amount in the total sample. By labeling the sample's cDNA molecules prior to hybridization, usually by attachment of phosphorous tags, one can then estimate the amount of hybridized molecules in each probe by measuring the emitted light intensity.

Microarrays come in two main flavors: cDNA microarrays and oligonucleotide chips. In **cDNA microarrays**, each probe consists of cDNA molecules a few hundred base pair long, taken from cDNA libraries. The advantage of this method is that the probed genes do not even have to be sequenced to be measured on the array. The common use of cDNA microarrays is for measurement of *competitive hybridization*: the extract from the sample of interest is labeled with red fluorescent tags. It is then mixed with an equal amount of extract from a reference condition which has been labeled with green tags. The mixture is then applied to the microarray, and for each gene the corresponding cDNA copies from the two populations compete on hybridization onto the probe. The ratio between red (sample) and green (control) signals then quantifies how much that gene is over or under-expressed in the sample compared to the control (Figure 1.2(a)). Using this ratio has the advantage of filtering out noise factors with similar effects on both populations.

In **oligonucleotide chips** (Lockhart et al., 1996) for each gene there are multiple probes composed of short oligonucleotides (usually 25 base pair long), half of which representing exact segments from the measured gene, and the other half representing the same segments with a single mutation. Processing this combination of probes yields a more sensitive estimate of the gene's mRNA level, controlling for different hybridization efficiencies and other sources of noise. Oligonucleotide chips are typically used with a single fluorescently tagged sample.

Early microarray experiments examined few samples, and mainly focused on differential display across tissues or conditions of interest. The design of recent experiments focuses on performing a larger number of microarray assays ranging in size from a dozen to a few hundreds of samples. Typical experiments may track the development of a phenomenon through a temporal sequences of samples, examine multiple samples of similar conditions (e.g., many samples of different tumor biopsies) and/or examine multiple samples at varying conditions (e.g. gene knock-outs and varying cultures). In such experiments, the measured expression levels of a given gene across the different assays is termed the *expression profile* of that gene.

1.2.2 Reporter Plasmid Assays

Recently, Kalir et al. (2001), introduced an experimental method that provides detailed transcription rate information. The method measures the promoter activity of different *E. Coli* operons by monitoring green fluorescent protein (GFP) level in different E. Coli strains. In each strain, the GFP gene is implanted in a reporter plasmid, preceded by a promoter sequence of the operon of interest. GFP level is therefore indicative of the total RNA transcribed from that operon (given that GFP degradation rate is virtually zero in *E. Coli*). Since the colonies go through an exponential growth stage during the experiment, it is necessary to estimate the total number of cells at each measurement. This is done by measuring the *optical density* (OD) of the culture, which is directly proportional to

the number of cells. The experiment design is such that the same protocol is executed in parallel to several dozen such *E. Coli.* strains, placed in separate wells (Figure 1.2(b)). In each well, the experiment records the GFP and OD level every 2 or 3 minutes, and the number of such measurements is essentially unlimited. A gene's transcription rate can therefore be estimated by the derivative of the GFP level divided by the OD level measured in the corresponding well.

The reporter plasmid method offers several advantages over the microarray methods. First, the measurements are done *in vivo*. Second, since each measurement only involves a visual scan, rather than a brand new microarray, it enables sampling of high frequency time series of several hundred samples at a very low cost. The limitation of this method is in it's scale, which is somewhat smaller. Current experiments measure around a 100 gene probes simultaneously. This enforces experimental design where the measured genes have to be carefully pre-selected prior to measurements.

1.2.3 Sources of Noise

The measurement of mRNA transcript levels is subject to many different types of noise, both from biological and from technical sources. First, the process of gene regulation is stochastic in nature, with every stage along its path subject to variability. The mRNA transcription level of a gene is a result of a sum of stochastic events, such as DNA binding, transcription initiation and elongation, and eventually also degradation (McAdams and Arkin, 1997). The higher the transcription rate, the more events are involved, resulting in a higher variance. This suggests that most biological sources of noise are *multiplicative* in nature.

On the technical side there are many sources of noise in the methods we reviewed. The efficiency of the hybridization process differs between probes, as well as its dependence on temperature. Cross-hybridization may occur between homologous genes. The amount of sample that reaches each probe has some variability. The spot size and density depends on surface and solution properties, which vary between probes. Uneven surface can also lead to different background intensities, as some of the tagged molecules stick to that surface. Some of these sources of noise do not depend on the gene's mRNA quantity in the sample, and therefore contribute *additive* components to the experiment noise. In general, in cDNA microarray experiments some of the noise can be reduced by using the expression ratio signal. In oligonucleotide chip experiments many of these sources of noise are eliminated by proper signal normalization algorithms (Irizarry et al., 2003; Li and Wong, 2001) which take into account both the different probes per gene and external control probes present on the chip.

1.3 Previous Analysis Approaches

We now briefly review previous approaches used for analyzing gene expression data. In the final chapter of this thesis we get back to these methods and discuss their relation to our work. Most of the early analysis tools for expression data sets were based on *clustering* algorithms. These

approaches attempt to locate groups of genes that have similar expression patterns over a set of experiments (Alon et al., 1999; Ben-Dor et al., 1999; Eisen et al., 1998b; Michaels and et al., 1998; Spellman et al., 1998). The genes in each group are then postulated to have similar mechanisms of regulation, and are therefore assumed to be functionally related. Oftentimes, a sequence motif finding procedure is applied to the promoter regions of the genes in each cluster, in order to find putative binding sites of the cluster's common regulators (Zhang, 1999). Different clustering approaches differ by the similarity measure employed (for example, linear correlation) and the clustering algorithms (for example, hierarchical clustering). Alternatively, clustering and classification algorithms have also been applied to the experiments (Alizadeh et al., 2000; Ben-Dor et al., 2000; Bhattacharjee et al., 2001). Besides classifying disease types (which can be useful for diagnosis), this approach can also reveal which genes are relevant to which biological conditions, an important step towards identification of drug targets.

A more ambitious goal for analysis is revealing the structure of the transcriptional regulation process. A number of models have been proposed, along with methods for learning them from data. Perhaps the simplest is the Boolean networks model (Kauffmann, 1993; Somogyi et al., 1996). In this model, each gene is modeled as a boolean entity, which can be in one of two states: on or off. The dynamics are modeled over a discrete series of time points. The state of each gene is determined by a boolean function of some of the other genes at the previous time step. Different algorithms have been proposed for inferring the network structure of such models from observations (Akutsu et al., 1998; Liang et al., 1998), typically by employing information-theoretic considerations. Other deterministic approaches model the expression of a gene as a linear (D'Haeseleer et al., 1999) or sigmoid (Mjolsness et al., 1991; Weaver et al., 1999) function of its regulators, either directly or as a solution to a set of differential equations (Chen et al., 1999b). In these approaches every gene is apriori assumed to depend on all other genes, and the connection strengths are learned through optimization, thus substituting structure learning with parameter learning.

The network inference methods we describe here are *model based*: They are all based on some simplified description, or a *model*, of the biological processes that could have generated the observed data. This is in contrast to most clustering approaches, which are *procedural* in nature: they specify a procedure, or a sequence of procedures, to be applied to the input data, without basing it on a specific model. Why do we need a model? After all, when we are interested in a specific question (like which genes are functionally related), a procedure based on insightful analysis of the problem can do the work pretty well. This holds for experimental procedures as well. What is the advantage of formulating a description of our domain of interest?

First, a model has an intuitive appeal in that its components correspond to real entities in our domain. We can therefore use it to get new insights into the nature of the domain, even if they were not part of the original motivation for building it. For example, we can analyze the in-degree distribution of nodes in a regulation network model, and learn something about the number of regulators per gene. Second, we can use the model to generate predictions, by simulation or analysis. For example, we can ask what would be the outcome of turning off a certain gene. Finally, the

natural interpretation of a model allows us to use it modularly, for example by adding or removing genes into a network model.

A common feature to the models described above is their determinism: Once the structure and parameters of a model are learned, setting it to an initial state will determine the exact dynamics of the system, including its steady state (if one exists). As we discussed in the previous section, the biological processes of regulation and the measurement methods employed contain many stochastic components, noise sources and hidden quantities. This might explain why the success of these deterministic methods in learning from real data sets has been limited.

1.4 Our Approach

Our aim in this thesis is to learn about gene regulation networks from experimental data. To approach this challenging task, we adopt a probabilistic model based approach. Given a data set of measurements of any kind, we try to come up with a probabilistic model describing how this data was generated. Such a model tries to account for both the biological processes that generated the observed phenomena (for example, certain amounts of molecules of a given kind at a given time), and the processes involved in measuring these phenomena (such as assays which measure molecule quantities). Both types of processes involve stochastic events (and therefore are not deterministic, nor exactly replicable). Moreover, some of the stages and quantities in those processes are not observed. For some the design details are not known: how they are connected to the other components in the system. The modeling language we use, therefore, has to account for stochasticity at all levels, to allow the use of unobserved entities, and to be able to search for the "correct" design diagram.

We choose to base our modeling language on *Bayesian networks* (Pearl, 1988). These networks are graphical representations of joint probability distributions over many random variables, capturing properties of conditional independence between these variables. Such models are attractive for their ability to describe complex stochastic processes, and since they provide clear methodologies for model learning from (noisy) observations. Bayesian networks allow different representations for the local regulatory interactions between the modeled entities, and we explore here some possible representations. Also, their formalism allows to model hidden, or unobserved entities. We will use this capability when we develop a realistic model of regulation.

We learn our models (both parameters and structure) automatically from data, trying to maximize the probability of the model given the data. Though there are well known methods for learning such models, we have to face specific difficulties related to our domain, including a small number of samples, a large number of variables, hidden variables at several levels and non-standard interaction and noise forms. To this end we develop two algorithms for speeding up model structure search, to overcome both the size of the domain and the cost of parameter computations in some of the models. We also develop two specific representation schemes, one of them non-parametric and the other a non-linear interaction model based on first principles of the biological processes.

The application of Bayesian network methods to our task requires careful definition of the model

semantics, the variables used and the assumptions at the background. We present two different approaches for learning regulation networks. In the first one we show how specific network features can be assessed by learning ensembles of models, using a simplified interaction model. In the second one we model explicitly the hidden variables in the regulation process, and employ a realistic interaction model, resulting in quantitative as well as qualitative predictions. Both methods are tested on data sets from different organisms, demonstrating their effectiveness.

1.5 Outline

The outline of this thesis is as follows. In Chapter 2 we review Bayesian networks, which form the basis to all our models. After definitions are given, we describe different possibilities for modeling dependencies in these networks. We then review methods for learning these models, both structure and parameters, from observational data. We focus on features of the Bayesian network model and on model variants which will be useful later on.

In Chapter 3 we introduce a novel representation scheme for Bayesian networks based on Gaussian process priors. These priors are semi-parametric in nature and can learn almost arbitrary noisy functional relations. We develop the Bayesian score of Gaussian Process Networks and describe how to learn them from data. We present empirical results on artificial data as well as on real-life domains with non-linear dependencies.

In Chapter 4 we introduce two novel methods for structure learning, which address specific difficulties: One for handling very large domains, and one for learning continuous variable networks with non-linear interactions as well as handling new hidden variables. The *Sparse Candidate* algorithm achieves faster learning by restricting the search space. It iteratively restricts the parents of each variable to belong to a small subset of candidates, and then searches for a network that satisfies these constraints. We evaluate this algorithm both on synthetic and real-life data, and show that it is significantly faster than alternative search procedures without loss of quality in the learned structures. The *Ideal Parent* method is a general method for significantly speeding the structure search for continuous variable networks with common parametric distributions. Importantly, this method facilitates the addition of new hidden variables into the network structure efficiently. We demonstrate the method on several data sets, both for learning structure on fully observable data, and for introducing new hidden variables during structure search.

In Chapter 5 we give a first application of Bayesian networks for modeling gene regulatory networks. We start by showing how Bayesian networks can describe interactions between genes. We then describe a method for recovering gene interactions from microarray data using tools for learning Bayesian networks and for statistically validating their features. Finally, we demonstrate this method on the *S. Cerevisiae* cell-cycle measurements of Spellman et al. (1998).

In Chapter 6 we discuss some of the limitations of the models presented in Chapter 5. We then describe a more realistic modeling approach. Unlike previous works, here we employ *quantitative* transcription rates, and simultaneously estimate both the kinetic parameters that govern these

rates, and the activity levels of unobserved regulators that control them. We apply our approach to expression data sets from yeast and E. Coli and show that we can learn the unknown regulator activity profiles, as well as the binding affinity parameters. Finally we show how the "Ideal Parent" method enables us to improve initial guesses of regulation topology, as well as reconstruct *ab initio* the regulatory network from those data sets.

In Chapter 7 we give concluding remarks, discuss recent related work, and describe some future challenges which come up from missing aspects in our framework.

Chapter 2

Bayesian Networks

In the introduction we hinted at some of the qualities we require from a modeling language to be suitable for modeling gene regulatory networks from experimental data. This language should be able to account for stochasticity, both at the biological process level and at the measurement level. It must be able to model unobserved entities, since many of the biological processes and molecules involved in regulation are not measured. And finally, it should be able to search for the "correct" design diagram, since our knowledge of the regulation wiring diagram is at best partial.

In this chapter we review Bayesian networks, our main modeling tool. As we show in the next chapters, it will form the basis for our models of gene regulation networks. We show how within the Bayesian network framework we can answer such questions as: What is the most likely structure of a regulation network, given a set of observations and possibly some prior knowledge? Which features of this structure are we more confident in? Given the structure of a network, can we predict the effect of specific expression values in some of the genes on the expression of some other genes?

Bayesian networks are a language for representing joint probability distributions of many random variables. They are particularly effective in domains where the interactions between variables are fairly local: each variable directly depends on a small set of other variables. Bayesian networks have been applied extensively for modeling complex domains in different fields (see, for example Heckerman et al., 1995b). This success is due both to the flexibility of the models and to the naturalness of incorporating expert (or prior) knowledge into the domain. Another important ingredient for many applications is the ability to induce such models from data. This is particularly important when our knowledge about the domain is partial, as is the case in the biological domains we are interested in. We now give a brief overview of the formalism of Bayesian networks and the algorithms for learning such models from observed data.

2.1 Model Definition

We begin with a number of notations. Consider a finite set $\mathcal{X} = \{X_1, \ldots, X_N\}$ of random variables. Each variable X_i may be discrete, in which case it may take on any value x_i from the domain

$Val(X_i)$, or it may be continuous, in which case it might take a value from some real interval. In this thesis, we use capital letters, such as X, Y, Z, for variable names and lowercase letters x, y, z to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, and assignments of values to the variables in these sets are denoted by boldface lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$.

A key notion in the language of Bayesian networks is that of *conditional independence*.

Definition 2.1.1: We say that $\mathbf{X}$ is *conditionally independent* of $\mathbf{Y}$ given $\mathbf{Z}$ if

$$P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P(\mathbf{X}|\mathbf{Z})$$

and we denote this statement by $(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z})$.∎

To demonstrate this notion, consider the case of a very rare genetic mutation in a family of three generations: grandfather, father and son. Let's say we are interested in the probability of the son to carry this mutation. With no prior knowledge, this probability might be very small (say one in a million). If we know the grandfather has the mutation, then this probability rises to 0.25. In other words, the son's genotype is dependent on his grandfather's genotype. Now, assume we know the father does not carry the mutation. In this case, the son's chances of having the mutation are again low, regardless of whether the grandfather carries the mutation or not. This is true because genetic information can pass from the grandfather to the son only through the father. We therefore say the son's genotype is conditionally independent of his grandfather's given his father's genotype.

As we shall see, the Bayesian network model associates a set of random variables with a graph representation. We therefore want to tie the notion of conditional independence with some graphical representation. One intuitive way of doing this, is through a set of rules called *local Markov assumptions*:

Definition 2.1.2: Let $\mathcal{G}$ be a *Directed Acyclic Graph* (DAG) whose vertices correspond to random variables $\mathcal{X} = \{X_1, \ldots, X_N\}$. Let $\mathbf{U}_{\mathbf{X_i}}$ denote the parents of X_i in $\mathcal{G}$. We say that $\mathcal{G}$ encodes the *local Markov assumptions* over $\mathcal{X}$: Each variable X_i is independent of its non-descendants, given its parents in $\mathcal{G}$.

$$\forall X_i \ \ (X_i \perp \mathbf{NonDescendants}_{X_i} \mid \mathbf{U_i})$$

and we denote the set of these assumptions as *Markov*$(\mathcal{G})$∎

To demonstrate the concept of local Markov assumptions, we extend our genetic mutations example to a family of 5 persons. Figure 2.1 shows a graph for the random variables denoting the mutation carrying for each of the 5 persons. In this particular example, the structure of this network also represents the relevant part of the family tree (i.e. a parent in this network corresponds to a parent in real life). According to the Markov assumptions, we can read several independence relations from this network, which in this example make sense intuitively. For example, the three

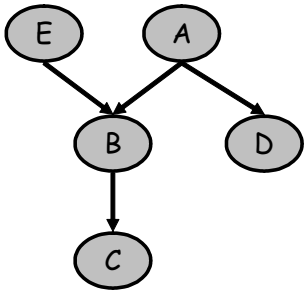


Figure 2.1: An example of a simple Bayesian network structure for the family genetics example. This network structure implies several conditional independence statements:$(A \perp E)$,$(B \perp D \mid A, E)$, $(C \perp A, D, E \mid B)$, $(D \perp B, C, E \mid A)$, and $(E \perp A, D)$.

generation example we gave above corresponds to the assumption that C is independent of A (or E) given B. As another example, $(B \perp D \mid A, E)$ means that if we know the genotype of the grandparents E and A, the genotype of B is independent of that of his brother D. This is intuitive: B and D are brothers, and so their genotypes are dependent. But once we know the genotype of their common parent A, none of them adds new information about the other.

Armed with the notion of reading independencies from a graph structure, we can now define Bayesian networks. These include the structural graph component, but also a quantitative component.

Definition 2.1.3: (Pearl, 1988) A *Bayesian network* is a representation of a joint probability distribution, consisting of two components. The first component, $\mathcal{G}$, is a *directed acyclic graph* (DAG) whose vertices correspond to the random variables $\mathcal{X} = X_1, \ldots, X_N$, and whose structure encodes the *Markov assumptions Markov*$(\mathcal{G})$ over $\mathcal{X}$. The second component, θ, describes a conditional probability distribution (CPD), $P(X_i|\mathbf{U_{X_i}})$, for each variable X_i in $\mathcal{X}$. ∎

The first component of the Bayesian network gives a set of independence conditions between the variables. The second component gives a local probability model for each variable given its parents in the network. These two components, $\mathcal{G}$ and θ, specify a unique distribution over $X_1, \ldots, X_N$, thanks to a result due to Pearl (1988):

Theorem 2.1.4: The independence assumptions derived from *Markov*$(\mathcal{G})$ are satisfied by a distribution $P(X_1, \ldots, X_N)$ if and only if P can be written as

$$P(X_1, \ldots, X_N) = \prod_{i=1}^{n} P(X_i \mid \mathbf{U_i}) \tag{2.1}$$

where $\mathbf{U_i}$ are the parents nodes of the variable X_i in $\mathcal{G}$.

This theorem is a direct consequence of the chain rule of probabilities and properties of conditional independence. The product form in Eq. (2.1) is called the *chain rule for Bayesian networks*. This

product form makes a Bayesian network representation of a joint distribution compact and economizes the number of parameters. For example, for our domain of 5 variables, if we do not use any independence assumptions, the joint distribution can be decomposed as:

$$P(A, B, C, D, E) = P(A)P(B|A)P(C|A, B)P(D|A, B, C)P(E|A, B, C, D)$$

while when using the independence assumptions implied by the network inFigure 2.1 we can write the same distribution as:

$$P(A, B, C, D, E) = P(A)P(E)P(B|A, E)P(C|B)P(D|A)$$

In the case where all the variables are binary, the former form requires $1 + 2 + 4 + 8 + 16 = 31$ parameters, while the latter requires only $1 + 1 + 2 + 4 + 2 = 10$ parameters. More generally, if $\mathcal{G}$ is defined over N binary variables and its indegree (i.e., maximal number of parents) is bounded by K, then instead of representing the joint distribution with $2^N - 1$ independent parameters we can represent it with at most $2^K N$ independent parameters.

2.1.1 Equivalence Classes

More than one graph can imply exactly the same set of independencies. For example, consider the graphs $X \rightarrow Y$ and $X \leftarrow Y$. Both imply the set $\text{Ind}(\mathcal{G}) = \emptyset$. This observation leads to a definition of equivalence between networks:

Definition 2.1.5: Two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are *equivalent* if $\text{Ind}(\mathcal{G}_1) = \text{Ind}(\mathcal{G}_2)$. That is, both graphs are alternative ways of describing the same set of independencies .∎

This notion of equivalence is crucial, since when we examine observations from a distribution, we cannot distinguish between equivalent graphs, under the common scenario of learning networks. This scenario is violated in two cases: First when we restrict the allowed networks to a certain structural family, for example trees. Second, when we use a type of CPDs that prefers a certain directionality in the connections. In both these cases we might have a preference of one equivalent network over another. Pearl and Verma (1991) show that we can characterize *equivalence classes* of graphs using a simple representation. In particular, these results establish that equivalent graphs have the same underlying undirected graph but might disagree on the direction of some of the arcs. We first define a useful sub-structure that plays a key role in the definition of graph equivalence.

Definition 2.1.6: (Pearl, 1988) A *v-structure* is an induced sub-graph of the form $X \rightarrow Y \leftarrow Z$ so that no edge exists between X and Z.∎

The v-structure implies an interesting set of dependencies. Given the value of Y, two possibly independent variables become dependent. A classic example of such a dependency can be seen in our genetic mutation example (Figure 2.1). Consider the variable B and its parents E and A. The

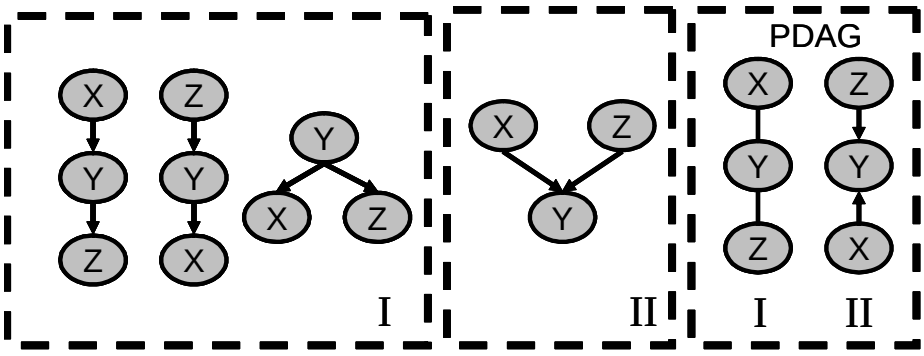


Figure 2.2: The skeleton $X - Y - Z$ is partitioned into two equivalence classes: I representing $\{(X \perp Z \mid Y), \neg(X \perp Z \mid \emptyset)\}$ and II the v-structure representing $\{\neg(X \perp Z \mid Y), (X \perp Z \mid \emptyset)\}$. The right pane illustrates the corresponding PDAGs.

genotypes of E and A are independent apriori, since they have no blood relation. However, given the genotype of their common child B, they become dependent: if we know B carries the mutation, then we know one of the parents has to carry it as well. In this case if the father A does not carry the mutation ($A = 0$), then the mother E must carry it. In other words, $P(E = 1 \mid A = 0, B = 1) = 1$. On the other hand, if the father does carry the mutation, the chance that the mother also carries it is small. In other words, $P(E = 1 \mid A = 1, B = 1) = \epsilon$. The variables A and E are therefore dependent given the value of their common child B. This type of dependency is exceptional among those found in small network sub-structures.

We can now give precise conditions for two networks to be equivalent:

Theorem 2.1.7: (Pearl and Verma, 1991) Two Bayesian network structures are equivalent if and only if they have the same underlying undirected graph (termed *skeleton*) and the same v-structures.

For example, the skeleton $X - Y - Z$ can be partitioned into two equivalence classes. One containing three graphs representing $\{(X \perp Z \mid Y), \neg(X \perp Z \mid \emptyset)\}$ and the v-structure representing $\{\neg(X \perp Z \mid Y), (X \perp Z \mid \emptyset)\}$. Moreover, an equivalence class of network structures can be uniquely represented by a *partially directed graph* (PDAG) $\mathcal{P}$, where a directed edge $X \rightarrow Y$ denotes that all members of the equivalence class contain the directed edge $X \rightarrow Y$; an undirected edge $X - Y$ denotes that some members of the class contain the directed edge $X \rightarrow Y$, while others contain the directed edge $Y \rightarrow X$. Given a DAG $\mathcal{G}$, the PDAG representation of its equivalence class can be constructed efficiently (Chickering, 1995). Figure 2.2 shows the equivalence classes and corresponding PDAG representations for a three variable skeleton.

2.2 Representing Dependencies: the CPD

We now focus on the quantitative part of the Bayesian network model, namely the parameterization θ. This parameterization defines the *conditional probability distributions* (CPDs) $P(X_i \mid \mathbf{U_i})$,

which can be of any general form. We explore some of the common choices for CPD representations.

2.2.1 Discrete Variables: multinomial CPDs

When both the variable X_i and its parents $\mathbf{U_i}$ are discrete, the most general representation for a CPD is a *conditional probability table* (CPT). Each row in these tables corresponds to a specific joint assignment $\mathbf{u_{X_i}}$ to $\mathbf{U_{X_i}}$, and specifies the probability distribution for X_i conditioned on $\mathbf{u_{X_i}}$. For example, if $\mathbf{U_{X_i}}$ consists of k binary valued variables, the table will specify 2^k distributions. This general representation can describe any discrete conditional distribution. This flexibility, however, comes at a price: The number of free parameters is exponential in the number of parents.

As an example, consider the conditional distribution of B in our genetic domain given its parents A and E. By basic genetics, if a parent carries the mutation, there is a 50% chance he would pass it to the child. The conditional probability table will therefore look like:

a	e	$P(b=0)$	$P(b=1)$
$a=0$	$e=0$	1.00	0.00
$a=0$	$e=1$	0.50	0.50
$a=1$	$e=0$	0.50	0.50
$a=1$	$e=1$	0.25	0.75

In this case, the number of free parameters needed to describe the CPD is $2^2 = 4$. In some cases, depending on the structure of dependence of the variable on its parents, we can represent the CPD with other forms, using fewer parameters. For example, there are cases where observing one of the parent variables makes the other parent irrelevant: A person's probability of being involved in an accident on a certain day depends on his driving history. However, given that he did not drive that day, that history becomes irrelevant. This phenomena is captured by a *Context Specific Independence* (CSI) representation for CPDs (Boutilier et al., 1996) that is in fact a limited form of a decision tree. Another representation form is *Default tables* (Friedman and Goldszmidt, 1996). These are suitable for cases where the probability distribution of X_i given $\mathbf{U_i}$ has some default value excluding a small number of assignments $\mathbf{u_i}$.

2.2.2 Continuous Variables: Linear Gaussians and more

When a variable X and some or all of its parents are continuous (real) valued, no general parametric form can capture all types of dependence. There are different representation choices, all of them hide some modeling assumptions. We now briefly describe some representations for the case where all the parent variables are continuous.

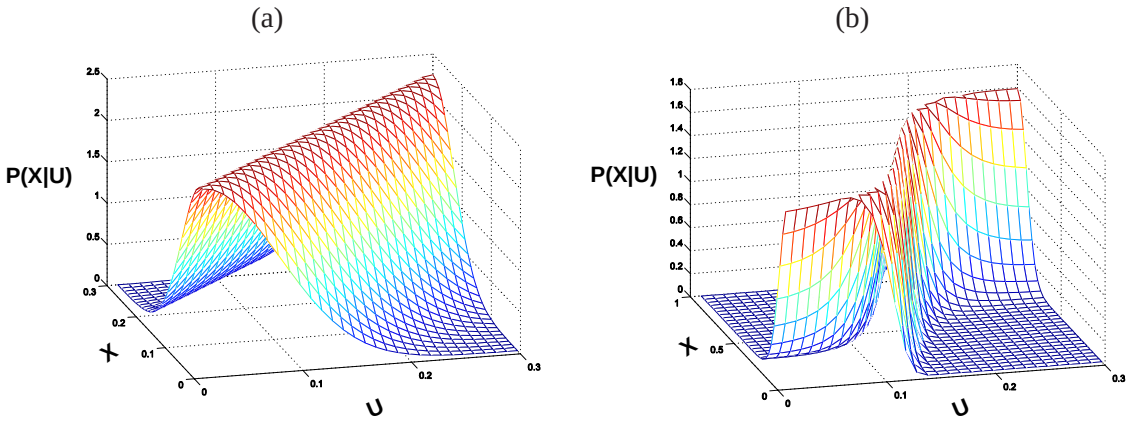


Figure 2.3: Two examples of conditional density representations for a variable X having a single parent $\mathbf{U}$. (a) A linear Gaussian CPD. (b) A sigmoid Gaussian CPD.

The simplest and best understood families of conditional densities are the *linear Gaussian* models. In this model we assume that

$$P(X \mid \mathbf{U}) \sim N(a_0 + \sum_i a_i \cdot \mathrm{U}^i, \sigma^2).$$

That is, X is normally distributed around a mean that depends *linearly* on the values of its parents. The variance of this normal distribution is independent of the parents' values. In this representation there are $|\mathbf{U}| + 2$ parameters: $|\mathbf{U}| + 1$ linear coefficients and one variance parameter. This figure is much smaller than the number of parameters for multinomial distributions, a property which is common to many of the continuous variable CPDs.

Networks where all the variables have a linear Gaussian CPD are called *Gaussian networks* (Geiger and Heckerm 1994). In such networks the joint density of the whole domain is a multivariate Gaussian. Furthermore, there is a simple procedure for going between the Gaussian network representation (parameterized by $\{\mathbf{a}_i\}, \{\sigma_i\}$) and the multivariate Gaussian representations (parametrized by a mean vector μ and a covariance matrix Σ). This makes several computations and tasks (like inference, discussed in Section 2.3) particularly simple. The drawback of Gaussian networks is that their representation is limited to modeling linear dependencies between variables. Thus, if the dependencies in the modeled domain are significantly non-linear, even the best-fit parameters will not describe observations from that domain very well, and the model's predictions are likely to be poor.

In many real world domains the dependencies are known to be non-linear, and some functional form of this non-linearity is assumed. For example, when modeling neural interactions, it is popular to represent their dependencies using a sigmoid function (Bishop, 1995). This form shows a saturation effect, and therefore is plausible biologically. The simplest way to embed this type of

dependency within a CPD is:

$$P(X \mid \mathbf{U}) \sim N(\theta_1 \frac{1}{1 + e^{-\sum_i a_i \mathbf{U}^i}} + \theta_0, \sigma^2) \tag{2.2}$$

That is, X is normally distributed around a mean that depends in a sigmoid manner on the values of its parents. Here, again, we can use a fixed variance parameter σ^2 for all values of the parents. Figure 2.3 demonstrates the sigmoid and linear Gaussian dependencies. In our domain of interest, namely gene expression regulation, we also expect a non-linear form of dependencies. InChapter 6 we develop the functional form of this dependency from basic principles of the biological processes involved, and use it to define CPDs.

There are cases where we do not know in advance what form of dependence to expect. A possible approach in this case is to use *mixtures of Gaussians* (Xu and Jordan, 1996). In this approach we model the conditional distribution as a weighted mixture

$$P(X \mid \mathbf{U}) = \sum_j w_j f_j(X \mid \mathbf{U})$$

where each f_j is a linear Gaussian distribution. In theory, such mixtures can approximate a wide range of conditional distributions. In particular, they can represent multi-modal distributions, and thus can represent relationships that are not purely functional. An alternative approach is to use a non-parametric representation for the CPD. We discuss this approach at length inChapter 4.

2.2.3 Hybrid Families

When our network contains a mixture of discrete and continuous variables, we need to consider how to represent a conditional distribution for a continuous variable where some of its parents are discrete valued, and for a discrete variable with continuous parents. The latter case is usually disallowed, and received very little theoretical treatment (see, e.g. (Koller et al., 1999; Lerner et al., 2001)). When a continuous variable X has a combination of discrete parents $\mathbf{U_D}$ and continuous parents $\mathbf{U_C}$, we usually use a mixture of densities(Lauritzen and Wermuth, 1989): For each joint assignment $\mathbf{u_D}$ to $\mathbf{U_D}$, we represent the conditional density of X given $\mathbf{U_C}$ using one of the simple CPD forms we described. For example, if we use linear Gaussians, the resulting CPD is a mixture of linear Gaussians. The number of components in this mixture is as the number of possible assignments to $\mathbf{U_D}$, which grows exponentially with the number of discrete parents.

2.3 Inference

A fundamental task in any graphical model is that of inference. That is, we want to be able to answer general queries of the form $P(\mathbf{X} \mid \mathbf{Z})$ where $\mathbf{X}, \mathbf{Z} \subset \mathcal{X}$, as efficiently as possible. Assume for example, that we want to evaluate the probability of the grandson C to carry the rare mutation

in the model of Figure 2.1. By the complete probability formula

$$P(c) = \sum_{a,b,d,e} P(a, b, c, d, e)$$

We can improve on this by utilizing the decomposition of the joint probability which results in

$$P(c) = \sum_{b} P(c|b) \sum_{a} P(a) \sum_{e} P(e) P(b|e, a) \sum_{d} P(d|a)$$

which is significantly more efficient: Each internal summation typically goes over the different assignments of only few variables, eliminating one of them. The computed factors are then reused in the next outer summation. While the complexity of the simple summation is $O(2^N)$ (in the case of binary valued variables), this decomposed summation costs only $O(N2^C)$, where C is roughly the number of possible assignments to the largest family in the network. The larger the network is, the bigger the advantage of this procedure. Furthermore, the factors computed during this summation can be reused in the computation of other queries. This procedure of *variable elimination* (summation) is the basis of all exact inference methods.

There are methods that enable answering multiple queries at the cost of two variable elimination computations. These include *Bucket Elimination* (Dechter, 1996) and *Junction Trees* (e.g., (Jensen et al., 1990)), both of which widely used. However, these cannot overcome the fact that inference in Bayesian networks is in general (excluding tree structured networks) NP-hard (Cooper, 1990). Consequently, to cope with large scale networks, a range of approximate inference techniques have been developed. These include sampling methods such as *Gibbs sampling* (see Neal, 1993, for an overview of sampling techniques), variational approximation methods such as the *Mean Field* approximation (see Jordan et al., 1998, for an introduction) and *Loopy Belief Propagation* (e.g., Murphy and Weiss, 1999, and references therein). While these methods have shown great success in different scenarios, like exact inference, approximate inference is NP-hard (Dagum and Luby, 1997) and choosing the best method of inference for a particular task remains a challenge.

2.4 Learning Bayesian Networks

A major advantage of Bayesian network models is the ability to learn them from observed data. This is important, since rarely are we in a situation where we know apriori both the structure and the exact parameters of the network describing our domain of interest. In the more common case, we might have some idea on the structure of dependencies, and maybe some idea on what local form these dependencies take. In these cases we need a methodology to learn a model that best describes the underlying distribution that generated the data.

The learning task deals with the following situation: We are given a *training set* of samples $\mathcal{D} = \{\mathbf{x}[1], \ldots, \mathbf{x}[M]\}$ that are independently drawn from some unknown generating Bayesian

network $\mathcal{G}^*$ with an underlying distribution P^*. Our goal is to recover $\mathcal{G}^*$.

We first deal with an easier task, where we assume the correct network structure $\mathcal{G}$ is given, and we only need to estimate the best parameters. We then present some approaches for learning the best structure given a data set.

2.4.1 Parameter Learning

Assume we are given a network structure $\mathcal{G}$, and a set of data instances $\mathcal{D}$ for the variables represented in $\mathcal{G}$. A natural question is what values for the network parameters θ best describe the process that generated the data. Though the assumption of known structure is not a reasonable one in our domain, the theory of parameter estimation is a basic building block for the structure learning methods described in Section 2.4.2.

We first need to define a measure of quality for a set of parameters θ over a data set $\mathcal{D}$. Without using any prior assumptions on the parameters, an intuitive and widely used measure is the probability that a model equipped with θ assigns to $\mathcal{D}$. This is called the *likelihood function* of θ given $\mathcal{D}$, which we denote here by $L(\theta : \mathcal{D})$:

$$L(\theta : \mathcal{D}) = \prod_{m=1}^{M} P(\mathbf{x}[m] \mid \theta)$$

In *Maximum likelihood estimation* we wish to choose parameters $\hat{\theta}$ that maximize the likelihood of the data:

$$\hat{\theta} = \max_{\theta} L(\theta : \mathcal{D}) \tag{2.3}$$

In many common cases a maximum likelihood estimator is both consistent and unbiased: it tends to the true value of the parameter as the number of data samples grows larger, and its mean value for finite sample sizes is also equal to the true parameter. There are cases, however, where neither consistency nor unbiasedness hold (see for example (Minka, 1998)).

Eq. (2.3) could potentially be a hard expression to optimize, due to the high dimensionality of θ and the large number of parameters that need to be concurrently optimized. One of the big advantages of the Bayesian network representation is that this likelihood decomposes into local likelihood functions. Not only does this simplify the calculation of the likelihood, more importantly it renders finding its optimal parameters tractable. Each local likelihood can be optimized in an independent manner, thus decomposing a complex global problem into smaller sub-problems.

$$\begin{aligned} L(\theta : \mathcal{D}) &= \prod_{m=1}^{M} P(\mathbf{x}[m]) \\ &= \prod_{m=1}^{M} \prod_{i=1}^{n} P(x_i[m] \mid \mathbf{u_i}[m] : \theta) \\ &= \prod_{i=1}^{n} \left[\prod_{m=1}^{M} P(x_i[m] \mid \mathbf{u_i}[m] : \theta) \right] \\ &= \prod_{i=1}^{n} L_i(\theta_{X_i|\mathbf{u_i}} : \mathcal{D}) \end{aligned}$$

where $L_i(\theta_{X_i|\mathbf{U_{X_i}}} : \mathcal{D}) = \prod_{m=1}^{M} P(x_i[m] \mid \mathbf{u_i}[m] : \theta)$ is the *local likelihood function* for X_i, whose form depends on our choice of CPD representation.

The maximum likelihood estimate for the local likelihood function parameters can often be computed in closed form. In the case of table CPDs this local likelihood can be further decomposed into a simple tractable form. Suppose we have a variable X with its parents $\mathbf{U}$, then we have a parameter $\theta_{x|\mathbf{u}}$ for each combination of $x \in Val(X)$ and $\mathbf{u} \in Val(\mathbf{U})$. The idea behind the decomposition is to group together all the instances in which $X = x$ and $\mathbf{U} = \mathbf{u}$. We denote by $M[x, \mathbf{u}]$ the number of these instances, and $M[\mathbf{u}] = \sum_{x \in X} M[\mathbf{u}, x]$. Then by rearranging the order of the product we can write

$$L_i(\theta_{X|U} : \mathcal{D}) = \prod_{\mathbf{u} \in Val(\mathbf{U_i})} \prod_{x \in Val(X)} \theta_{x|\mathbf{u}}^{M[x,\mathbf{u}]} \tag{2.4}$$

By optimizing the local likelihood functions under normalization constraints, we obtain the maximal likelihood estimators (MLE) for the parameters of the multinomial table CPD:

$$\hat{\theta}_{x|\mathbf{u}} = \frac{M[x, \mathbf{u}]}{M[\mathbf{u}]} \tag{2.5}$$

We call the counts $M[x, \mathbf{u}]$ and $M[\mathbf{u}]$ *sufficient statistics*. Given these counts, the actual data instances $x[1] \ldots x[M]$ themselves are no longer needed. The sufficient statistics summarize all the relevant information from the data that is needed in order to calculate the likelihood. For the case of linear Gaussian CPDs, for example, the sufficient statistics required for computing the MLE estimators are the first and second moments of the variables and its parents (that is, the observed expectations $E[X]$, $E[\ \mathrm{U}^i]$, $E[X\ \mathrm{U}^i]$ and $E[\ \mathrm{U}^i\ \mathrm{U}^j]$). each i and j).

The Bayesian Approach

The maximum likelihood estimation approach for fitting parameters to data has a number of disadvantages. As most estimators in orthodox statistics, it is based on some intuition, and may or may not possess some desired qualities, such as consistency and unbiasedness, depending on the CPD. But the most obvious drawback of the ML estimator is its total reliance on the training data, without assuming any prior knowledge. This drawback is most evident when the training data set is small. Assume, for example we are trying to estimate the probability of a coin to fall heads up from three coin tosses only. If in all tosses the coin falls tails up, our ML estimation of the coin's parameter would be $P(headsup) = 0$. This counters our prior belief that the coin is more or less balanced, and that the three successive tosses represent a not-so-unlikely event. We say in such cases that the ML estimator *overfits* the training data. This causes it to represent poorly the true distribution.

We therefore turn to the *Bayesian approach*, which formulates this concept of prior belief in a principled manner. We first demonstrate this approach on a data set $\mathcal{D}$ of a single variable X. The idea is that before observing any data, we have some initial distribution, $P(\theta)$, termed the *prior*, which encodes our beliefs on the modeled process. This prior can encode a strong belief (as it might do in the example of the coin toss), in which case $P(\theta)$ can be sharply peaked around certain values. On the other hand, this prior can also encode ignorance, in cases where we do not have a strong notion of the process. In those cases, the prior might be more flat.

After we observe some data $\mathcal{D}$ we can update our belief over θ, to reflect the combination of our prior belief and the observations. The updated distribution, denoted $P(\theta \mid \mathcal{D})$, is called the *posterior distribution*, and is obtained through Bayes rule:

$$P(\theta \mid D) = \frac{P(\mathcal{D} \mid \theta)P(\theta)}{P(\mathcal{D})}. \tag{2.6}$$

The term $P(\mathcal{D})$, termed the *marginal likelihood*, averages the probability of the data over all possible parameter assignments:

$$P(\mathcal{D}) = \int P(\mathcal{D} \mid \theta)P(\theta)d\theta \tag{2.7}$$

In the Bayesian approach we do not give one estimate of the parameters. Rather than pretending we know the true value of θ, we use our updated posterior belief over θ to weigh the different possibilities. A new sample $X[M+1]$ will therefore have the following distribution:

$$P(X[M+1] \mid \mathcal{D}) = \int P(X[M+1] \mid \mathcal{D}, \theta)P(\theta \mid \mathcal{D})d\theta \tag{2.8}$$

We now turn to the issue of choosing a prior distribution $P(\theta)$. We would like this prior to have certain qualities: first, its parametric form should allow a good description of our prior belief over the parameters. Second, the choice of parameter values for this prior should reflect this prior belief. And last, we would like the prior to have a convenient form for the computations we need to perform, particularly the integrals in Eq. (2.7) and Eq. (2.8). The prior we choose will typically

represent a compromise between those desired qualities. One well-known strategy is to choose a prior with a suitable form so the posterior belongs to the same functional family as the prior. The choice of the family depends on the form of the likelihood function, and so such a prior is said to be *conjugate* to the likelihood parametric family. Having the same form for the posterior and the prior has some advantages, as we discuss below.

For multinomial distributions the conjugate prior has the form of a *Dirichlet distribution* (DeGroot, 1970). This distribution is parameterized by a set of *hyperparameters* $\alpha_{x^1|\mathbf{u}}, \ldots \alpha_{x^K|\mathbf{u}}$, one such hyperparameter corresponding to each $x^j \in Val(X)$. The Dirichlet distribution is specified by:

$$P(\theta) = Dirichlet(\alpha_{x^1|\mathbf{u}}, \ldots \alpha_{x^K|\mathbf{u}}) \propto \prod_j \theta_{x^j|u}^{\alpha_{x^j|u}-1} \tag{2.9}$$

This form is very similar to the multinomial likelihood form in Eq. (2.4), and so the posterior distribution has the same form:

Proposition 2.4.1: *(DeGroot, 1970) If $P(\theta)$ is $Dirichlet(\alpha_{x^1}, \ldots \alpha_{x^K})$ then the posterior $P(\theta \mid \mathcal{D})$ is $Dirichlet(\alpha_{x^1} + M[x^1], \ldots, \alpha_{x^K} + M[x^K])$ where $M[x]$ are the sufficient statistics derived from $\mathcal{D}$.*

Thus, the hyper-parameters α_{x_i} play a similar role to the empirical counts and are often referred to as *imaginary counts* and their sum $M' \equiv \sum_x \alpha_x$ is called the *effective sample size* of the prior. That is, using a Dirichlet prior with the above hyper-parameters is equivalent to having seen, prior to $\mathcal{D}$, M' other samples where in $M'\alpha_{x^k}$ of them are $X = x^k$. This intuition is also reflected in the resulting form for the likelihood of a new sample:

$$P(X[M+1] \mid \mathcal{D}) = \frac{\alpha_x + M[x]}{\sum_{x'} \alpha_{x'} + M[x']} \tag{2.10}$$

For Gaussian densities there also exist conjugate priors. For a general multivariate Gaussian distribution, the conjugate prior for the mean vector μ and precision matrix W is called a normal-Wishart prior (DeGroot, 1970, p. 178). The posterior also has a normal-Wishart form, and the likelihood of a new sample has the form of a t distribution. Geiger and Heckerman (1994) give a full account of these results.

We now turn back to full Bayesian networks. In the case of MLE we have seen that the likelihood function decomposes according to the network structure. This allowed us to estimate the parameters $\theta_{X_i|\mathbf{U_i}}$ for each family independently in Eq. (2.3). For Bayesian estimation, this is not guaranteed a priori. There can be priors that introduce dependencies between parameters of different variables, or between different parameters of the same variable. When setting a prior over all the network's parameters, it is therefore useful to introduce some independence assumptions:

Definition 2.4.2: (Spiegelhalter and Lauritzen, 1990) A parameter prior $P(\theta)$ for a Bayesian network is said to satisfy *global parameter independence* if it decomposes into the following form

$$P(\theta) = \prod_{i=1}^{n} P(\theta_{X_i|\mathbf{U_i}})$$

∎

For multinomial CPDs, where X has a distinct set of parameters for each instantiation of its parents $\mathbf{U}$, a further independence assumption is:

Definition 2.4.3: Let X be a variable with parents $\mathbf{U}$ and a multinomial CPD. We say the prior $P(\theta_{X|\mathbf{U}})$ has *local parameter independence* if $P(\theta_{X|\mathbf{U}}) = \prod_{\mathbf{u}} P(\theta_{X|\mathbf{u}})$ ∎

We say that the prior $P(\theta)$ satisfies *parameter independence* if it satisfies both global and local parameter independence. Another useful requirement is *parameter modularity*:

Definition 2.4.4: (Geiger and Heckerman, 1994) A parameter prior satisfies *parameter modularity* if for any two network structures $\mathcal{G}$ and $\mathcal{G}'$ in which $\mathbf{U}_G^{X_i} = \mathbf{U}_{G'}^{X_i} = \mathbf{U}$ then

$$P(\theta_{X|\mathbf{U}} \mid \mathcal{G}) = P(\theta_{X|\mathbf{U}} \mid \mathcal{G}') \tag{2.11}$$

∎

That is, the prior for a conditional distribution depends only on the choice of parents for X_i and is independent of other aspects of the graph $\mathcal{G}$.

Assuming parameter independence, we can assign an independent prior distribution $\theta_{x_i|\mathbf{u_i}} \sim Dirichlet(\alpha_{x_i^1|\mathbf{u_i}}, \ldots \alpha_{x_i^K|\mathbf{u_i}})$ for each variable and each instantiation of its parents in the network. The estimation problem in this case decomposes to the different families in the network:

$$P(X_i[M+1] = x_i \mid \mathbf{U_i}[M+1] = \mathbf{u_i}, \mathcal{D}) = \frac{\alpha_{x_i|\mathbf{u_i}} + M[x_i, \mathbf{u_i}]}{\sum_{x_i'} \alpha_{x_i'|\mathbf{u_i}} + M[x_{i'}, \mathbf{u_i}]} \tag{2.12}$$

For the Gaussian network case, Geiger and Heckerman (1994) show that one can use a normal-Wishart prior over the parameters of the multivariate Gaussian distribution on $\mathcal{X}$ implied by the network. For any subset of variables $\mathbf{Y} \in \mathcal{X}$ this prior implies a normal-Wishart prior on the relevant part of the mean vector and the covariance matrix. The posterior therefore also has a normal-Wishart form, and the new sample density for $\mathbf{Y}$ is a multivariate t distribution. By using $\{X_i, \mathbf{U_i}\}$ or $\mathbf{U_i}$ in the role of $\mathbf{Y}$, we can obtain the new sample local estimate for the Gaussian network case:

$$P(X_i[M+1] = x_i \mid \mathbf{U_i}[M+1] = \mathbf{u_i}, \mathcal{D}) = \frac{P(X_i[M+1] = x_i, \mathbf{U_i}[M+1] = \mathbf{u_i} \mid \mathcal{D}^{X_i,\mathbf{U_i}})}{P(\mathbf{U_i}[M+1] = \mathbf{u_i} \mid \mathcal{D}^{\mathbf{U_i}})}$$

where $\mathcal{D}^{\mathbf{Y}}$ denotes the data set limited to the variables in $\mathbf{Y}$. Geiger and Heckerman (1994) prove the correctness of this equation using the parameter independence and modularity assumptions, and show that the normal-Wishart prior is consistent with those assumptions (Heckerman and Geiger, 1995) .

2.4.2 Learning Structure

There are few cases where we can determine a full Bayesian network structure solely by expert knowledge of the domain. This is typical for simple domains with not too many variables. The more common case, especially in domains like the gene regulation domain, is that we either have no idea about the structure or we have partial knowledge on some of its parts. In such cases we would like to be able to learn the structure from data. In this section we describe the known approaches for learning structure from data.

Constraint Based vs. Score Based Approaches

The theory of learning networks from data has been examined extensively over the last decade. Somewhat generalizing, there are two approaches for finding structure. The first approach poses learning as a *constraint satisfaction* problem. In that approach, we try to estimate properties of conditional independence among the attributes in the data. Usually this is done using a statistical hypothesis test, such as χ^2-test. We then build a network that exhibits the observed dependencies and independencies. Examples of this approach include Pearl and Verma (1991) and Spirtes et al. (1993). The second approach poses learning as an *optimization* problem. We start by defining a statistically motivated *score* that describes the fitness of each possible structure to the observed data. The learner's task is then to find a structure that maximizes the score. In general, this is an NP-hard problem (Chickering, 1996), and thus we need to resort to heuristic methods. Although the constraint satisfaction approach is efficient, it is sensitive to failures in independence tests. Thus, the common opinion is that the optimization approach is a better tool for learning structure from data.

In this thesis we take the score based approach to learning. We start by defining a structure score. We then describe a search algorithm that attempts to find the highest scoring structure.

2.4.3 Scoring a Structure

The most principled approach for scoring a network structure is, again, the Bayesian approach. As in the case of parameter estimation, we do not assume a single "correct" structure, but rather state our prior beliefs on the structure using a prior distribution $P(\mathcal{G})$ over the space of possible network structures, and then update those beliefs using the data and Bayesian conditioning, to give a posterior distribution $P(\mathcal{G} \mid \mathcal{D})$ over this space.

We start with the prior over structures, $P(\mathcal{G})$. Several priors have been proposed, all of which

are quite simple. Without going into detail, a key property of all these priors is that they satisfy *Structure modularity*:

Definition 2.4.5: A prior $P(\mathcal{G})$ satisfies *Structure Modularity* if it can be written in the form

$$P(\mathcal{G}) \propto \prod_i \rho(X_i, \mathbf{U}_{\mathcal{G}}^{X_i});$$

■

That is, the prior decomposes into a product, with a term for each family in $\mathcal{G}$. In other words the choices of the families for the different nodes are independent a priori. The uniform prior, for example, in which all structures have the same prior probability, satisfies this property in a trivial manner.

We next need to specify a prior over parameters given the structure, $P(\theta \mid \mathcal{G})$. Once we define these priors, we can examine the form of the posterior probability. Using Bayes rule, we have that

$$P(\mathcal{G} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \mathcal{G})P(\mathcal{G}).$$

The term $P(\mathcal{D} \mid \mathcal{G})$ is the *marginal probability* of the data given $\mathcal{G}$ and is defined as the integration over all possible parameter values for $\mathcal{G}$

$$P(\mathcal{D} \mid \mathcal{G}) = \int P(\mathcal{D} \mid \mathcal{G}, \theta)P(\theta \mid \mathcal{G})d\theta \tag{2.13}$$

Alternatively, we can define $P(\mathcal{D} \mid \mathcal{G})$ using the chain rule:

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_i P(\mathbf{x}[i] \mid \mathbf{x}[1], \ldots, \mathbf{x}[i-1], \mathcal{G})$$

where $P(\mathbf{x}[i] \mid \mathbf{x}[1], \ldots, \mathbf{x}[i-1], \mathcal{G})$ is the probability of the i'th instance after observing the previous $i-1$ instances. If the parameter prior we use satisfies parameter independence and parameter modularity as defined in Section 2.4.1 one can show (see Heckerman et al., 1995a) that if $\mathcal{D}$ is complete, then:

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_i P(x_i[1], \ldots, x_i[M] \mid \mathbf{u}_i[1], \ldots, \mathbf{u}_i[M])$$

If, additionally, the prior $P(\mathcal{G})$ satisfies structure modularity, we can also conclude that the posterior probability $P(\mathcal{G} \mid \mathcal{D})$ decompose into families. Expressing this in log space, we obtain the

Bayesian score for structure:

$$\begin{aligned} &\text{Score}_{Bayesian}(\mathcal{G} \ : \ \mathcal{D}) \\ &= \log P(\mathcal{G} \mid \mathcal{D}) \\ &= \sum_i [\log \rho(X_i, \mathbf{U}_{\mathcal{G}}^{X_i}) + \log P(x_i[1], \ldots, x_i[M] \mid \mathbf{u}_i[1], \ldots, \mathbf{u}_i[M])] + Const. \end{aligned} \tag{2.14}$$

This decomposition of the score to *local terms* is crucial for learning structure, as we show in the next section. The terms in the sum are called the *family scores*. In case we are using a uniform prior on the structure space, the $\rho()$ terms only add a constant to the sum, and can be ignored. For multinomial variables, Heckerman et al. (1995a) show that by using a Dirichlet prior over the parameters, one can compute the family scores in Eq. (2.14) in closed form:

$$\begin{aligned} &\text{FamScore}_{BDe}(X_i, \mathbf{U_i} \ : \ \mathcal{D}) \\ &= \log P(x_i[1], \ldots, x_i[M] \mid \mathbf{u}_i[1], \ldots, \mathbf{u}_i[M]) \\ &= \log \left[\prod_{\mathbf{u_i}} \frac{\Gamma(\alpha_{\mathbf{u_i}})}{\Gamma(\alpha_{\mathbf{u_i}} + M[\mathbf{u_i}])} \prod_{x_i} \frac{\Gamma(\alpha_{x_i|\mathbf{u_i}} + M[x_i, \mathbf{u_i}])}{\Gamma(\alpha_{x_i|\mathbf{u_i}})} \right] \end{aligned} \tag{2.15}$$

where Γ is the Gamma function, $\alpha_{\mathbf{u_i}} = \sum_{x_i} \alpha_{x_i|\mathbf{u_i}}$ and $M[\mathbf{u_i}] = \sum_{x_i} M[x_i, \mathbf{u_i}]$. The score is called *BDe* for *Bayesian Dirichlet equivalence* score. The "equivalence" part is due to a strong property that is satisfied by this score, called *score equivalence*: any two equivalent network structures will obtain the same score on a given data set. This property might be desired during structure search, in case we do not want to be biased towards certain structures in an equivalence class.

For linear Gaussian variables, Geiger and Heckerman (1994) develop a score with similar properties, called *BGe* (Bayesian Gaussian equivalence) score, by using the normal-Wishart parameter prior. The family scores, in this case also have a closed form expression.

Besides the multinomial and the Gaussian cases, the marginal likelihood $P(\mathcal{D} \mid \mathcal{G})$ usually does not have a closed form expression, even if it can be decomposed into separate family terms. In these cases we usually use some approximation to the Bayesian score. There are a number of approximations (Akaike, 1974; Cheeseman and Stutz, 1995; Schwarz), all of them use the maximum a-posteriori (MAP) or maximum likelihood (ML) parameters, and some complexity penalization term. The most widely used of these is the *Bayesian Information Criterion* (BIC) score (Schwarz):

$$\text{Score}_{BIC}(\mathcal{G} \ : \ \mathcal{D}) = \log L(\hat{\theta}, \mathcal{G} : \mathcal{D}) - \frac{\log M}{2} Dim(\hat{\theta}) \tag{2.16}$$

where $\hat{\theta}$ are the MAP parameters of $\mathcal{G}$ given $\mathcal{D}$ (or the maximum likelihood parameters, in case we do not use a prior). This score can also be decomposed into local family scores:

$$\text{Score}_{BIC}(\mathcal{G} \ : \ \mathcal{D}) = \sum_i [\ell_{X_i}(\mathcal{D} : \mathbf{U_i}, \hat{\theta}_i) - \frac{\log M}{2} Dim(\hat{\theta}_i)] \tag{2.17}$$

Input : $\mathcal{D}$ // training set
$\mathcal{G}_0$ // initial structure
Output : A final structure G

$\mathcal{G}_{best} \leftarrow \mathcal{G}_0$
repeat
  G$\leftarrow \mathcal{G}_{best}$
  foreach Add,Delete,Reverse *edge in G* **do**
    $\mathcal{G}' \leftarrow$ `ApplyOperator(`G`)`
    if $\mathcal{G}'$ *is cyclic* **then continue**
    if $\mathrm{Score}(\mathcal{G}' : \mathcal{D}) > \mathrm{Score}(\mathcal{G}_{best} : \mathcal{D})$ **then**
      $\mathcal{G}_{best} \leftarrow \mathcal{G}'$
    end
  end foreach
until $\mathcal{G}_{best} == G$
return $\mathcal{G}_{best}$

Algorithm 1: Greedy Hill-Climbing Structure Search for Bayesian Networks

where $\hat{\theta}_i$ are the MAP parameters for the CPD of X_i, and $\ell_{X_i}(\mathcal{D} : \mathbf{U_i}, \hat{\theta}_i)$ is the log local likelihood function of X_i given its parents $\mathbf{U_i}$. The BIC score is closely related to the *MDL (Minimum Description Length)* principle (Rissanen, 1989): it approximates the number of bits needed to describe the data using the model. This follows a well known principle from learning theory, that models which compress the data better, also have better generalization performance, or alternatively are closer to the generating distribution. It can be shown that as the size of the data set grows to infinity, the BIC score converges to the Bayesian score.

2.4.4 Search Algorithms

How do we use the Bayesian score we just defined? The Bayesian approach is to use the posterior over all structures $P(\mathcal{G} \mid \mathcal{D})$ as our "solution". This posterior can be used, for example, to estimate certain features in the networks $f(\mathcal{G})$ by computing their expected values over all networks:

$$E[f] = \sum_{\mathcal{G}} f(\mathcal{G}) P(\mathcal{G} \mid \mathcal{D})$$

A simple feature might be the existence of an edge between two variables. The expectation $E[f]$ in that case is the probability of a structure to have this edge given the data. Since the summation over all possible structures is usually prohibitively expensive, some methods were suggested to approximate these expectations (Attias, 1999; Friedman and Koller, 2003).

A more common approach, is to seek for the highest scoring structure. In the case of a Bayesian score, this would be the *Maximum a posteriori* scoring structure, or it can be any one of the alternative scores, such as the BIC score. This problem is known to be NP-hard (Chickering, 1996). Thus,

we resort to a heuristic search. We define a *search space* where each *state* in this space is a network structure. We define a set of *operators* that take us from one structure to another. This defines a graph structure on the states: neighboring states are those which are one operation away. We then start with some initial structure (usually the empty graph) and using the operators traverse this space searching for high scoring structures.

The simplest operators are those which involve a change of a single edge at a time: *Add*, *Remove* or *Reverse* an edge. The benefit of such simple operators is that they only affect the family score of one or two variables. Thus, if our score decomposes into family scores, we only need to recompute one or two terms. For instance, if we add an edge to the variable X_i, we only have to recalculate X_i's family score, as the other family scores remain unchanged. This significantly reduces the computation time. Note we only consider operations that result in legal networks. That is acyclic networks that satisfy any other constraints we specify (e.g. maximal indegree constraints).

Once we have evaluated the scores for all neighboring states, we need to decide which move to make. The simplest choice is the *greedy hill climbing* approach. Here we simply make a move to the highest scoring neighbor. We then continue until we reach a local maximum. Although this procedure does not necessarily find a global maximum, it often performs well in practice. Algorithm 1 describes the flow of greedy hill climbing search.

Some alternative heuristics to the greedy method try to overcome the problem of local maxima. These include stochastic hill climbing, simulated annealing, TABU search and random restarts. These techniques typically try to evade local maxima by allowing search steps which either decrease the score, or that move to structures beyond the immediate neighbors in structure space.

2.5 Assigning Causal Interpretations

A Bayesian network is a model of dependencies between multiple measurements. However, in many cases we are also interested in modeling the mechanisms that generated these dependencies. Thus, we want to model the flow of causality in the modeled domain. For example, we would like to draw conclusions such as "Gene A activates gene B", rather than merely "The expression levels of genes A and B are statistically dependent".

A *causal network* is a model of such causal processes. It models not only the distribution of the observations, but also the effects of *interventions*. If X causes Y, then manipulating the value of X affects the value of Y. On the other hand, if Y is a cause of X, then manipulating X will not affect Y. Thus, although $X \rightarrow Y$ and $X \leftarrow Y$ are equivalent Bayesian networks, they are not equivalent causal networks. While at first glance there seems to be no direct connection between probability distributions and causality, causal interpretations for Bayesian Networks have been proposed (Pearl, 2000; Pearl and Verma, 1991). A causal network is mathematically represented similarly to a Bayesian network, a DAG where each node represents a random variable along with a local probability model for each node. However, causal networks have a stricter interpretation of the meaning of edges: the parents of a variable are its *immediate causes*.

A causal network can be interpreted as a Bayesian network when we are willing to make the *Causal Markov Assumption*: given the values of a variable's immediate causes, it is independent of its earlier causes. When the causal Markov assumption holds, the causal network satisfies the Markov independencies of the corresponding Bayesian network. For example, this assumption is a natural one in models of genetic pedigrees: once we know the genetic makeup of the individual's parents, the genetic makeup of her ancestors is not informative about her own genetic makeup.

The central issue is: When can we learn a causal network from observations? This issue received a thorough treatment in the literature (Heckerman et al., 1999; Pearl and Verma, 1991; Spirtes et al., 1993, 1999). We briefly review the relevant results for our needs here. For a more detailed treatment of the topic we refer the reader to (Cooper and Glymour, 1999; Pearl, 2000).

To learn about causality we need to make several assumptions. The first one is a modeling assumption: we assume that the (unknown) causal structure of the domain satisfies the Causal Markov Assumption. Thus, we assume that causal networks can provide a reasonable model of the domain. Some of the results in the literature require a stronger version of this assumption, namely that causal networks can provide a perfect description of the domain (that is an independence property holds in the domain if and only if it is implied by the model). The second assumption is that there are no *latent* or hidden variables that effect several of the observable variables. We discuss relaxations of this assumption below.

If we make these two assumptions, then we essentially assume that one of the possible DAGs over the domain variables is the "true" causal network. However, as discussed above, from observations alone, we cannot distinguish between causal networks that specify the same independence properties, i.e., belong to the same equivalence class (see section 2.1.1). Thus, at best we can hope to learn a PDAG description of the equivalence class that contains the true model.

Once we identify such a PDAG, we are still uncertain about the true causal structure in the domain. However, we can draw some causal conclusions. For example, if there is a directed path from X to Y in the PDAG, then X is a causal ancestor of Y in *all* the networks that could have generated this PDAG including the "true" causal model. Thus, in this situation we can recover some of the causal directions. Moreover, by using Theorem 2.1.7, we can predict what aspects of a proposed model would be detectable based on observations alone.

When data is sparse, we can not identify a unique PDAG as a model of the data. In such a situation, we can use the posterior over PDAGs to represent *posterior* probabilities over causal statements. In a sense the posterior probability of "X causes Y" is the sum of the posterior of all PDAGs in which this statement holds (See Heckerman et al. (1999) for more details on this Bayesian approach). The situation is somewhat more complex when we have a combination of observations and results of different interventions. From such data we might be able to distinguish between equivalent structures. Cooper and Yoo (1999) show how to extend the Bayesian approach of Heckerman et al. (1999) for learning from such mixed data.

A possible pitfall in learning causal structure is the presence of latent variables. In such a situation the observations that X and Y depend on each other probabilistically might be explained

by the existence of an unobserved common cause. When we consider only two variables we cannot distinguish this hypothesis from the hypotheses "X causes Y" or "Y causes X". However, a more careful analysis shows that one can characterize all networks with latent variables that can result in the same set of independencies over the observed variables. Such equivalence classes of networks can be represented by a structure called *partial ancestral graph* (PAGs) (Spirtes et al., 1999). As can be expected, the set of causal conclusions we can make when we allow latent variables is smaller than the set of causal conclusions when we do not allow them. Nonetheless, in many cases causal relations can be recovered even in this case. The situation is more complicated when we do not have enough data to identify a single PAG. As in the case of PDAGs, we might want to compute posterior scores for PAGs. However, unlike PDAGs the question of scoring a PAG (which consists of many models with different number of latent variables) remains an open question.

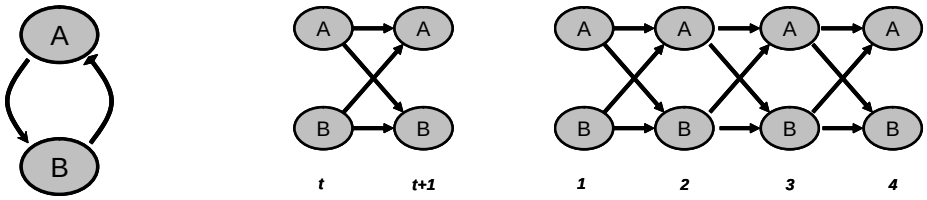
2.6 Modeling Time: Dynamic Bayesian Networks

So far we have assumed our data set is composed of *independent* samples from the generating distribution. Formally, this means that the likelihood over the samples can be decomposed as:

$$P(\mathcal{D} \mid \mathcal{G}, \theta) = \prod_{m=1}^{M} P(\mathbf{X}[m] \mid \mathcal{G}, \theta)$$

When our samples come from a time series of observations, taken from the domain in some intervals, this assumption no longer holds. Unless the process we are watching is totally memoryless, an observation at time t carries some information on observation at adjacent times (both in the future and in the past). For example, assume we want to model the concentrations of two enzymes, A and B. Assume also that A catalyzes the production of B, and B catalyzes the degradation of A. Such a type of interaction is termed a *negative feedback loop*, and is quite common in living cells: it usually results in either oscillatory behavior or in a stabilizing effect, depending on the time constants. Now, suppose we have a series of T observations, in some time intervals, of these two enzymes, and we index those observations with $t = 1, \ldots, T$. Clearly $A[t]$ is not independent of $A[t-1]$, since the concentrations have some persistence over time - they go up or down gradually. Can we suggest a Bayesian network like model to describe this data?

First we note that if we know the enzymes' concentrations at time t and the reaction parameters, their concentrations at earlier times do not help us predict their value at $t+1$. This is called the *Markov assumption*: given the current observation $\mathbf{X}[t]$, the next observation $\mathbf{X}[t+1]$ is independent of past observations, $\mathbf{X}[1], \ldots, \mathbf{X}[t-1]$. (Or more simply: the future is independent of the past given the present). This is a reasonable assumption in many domains. If the Markov assumption



(a) Interaction diagram (b) 2-TBN representation (c) Unrolled network

Figure 2.4: A dynamic model for the two enzyme system. (a) The interaction diagram (b) A 2-TBN representation of the dynamic Bayesian network model. (c) The unrolled network for 4 time slices.

holds, the data likelihood can be decomposed as:

$$P(\mathbf{X}[1], \ldots, \mathbf{X}[T]) = P(\mathbf{X}[1]) \prod_{t=2}^{T} P(\mathbf{X}[t] \mid \mathbf{X}[t-1])$$

Using this formulation, we still have to specify M sets of probability distributions, where M might be very large. We therefore usually make another simplifying assumption, that $P(\mathbf{X}[t] \mid \mathbf{X}[t-1])$ does not depend on t. That is, the probabilistic model is invariant over time, or *time invariant*. In the enzymes example, this means e.g. that A's effect on B does not depend explicitly on time, but rather on the concentrations themselves and the reaction parameters.

The Markov and time-invariance assumptions allow us to represent the probability distribution over the whole time series in a compact way. We note we only need two sets of parameters: one set to describe the initial distribution $P(\mathbf{X}[1])$, and another set to describe $P(\mathbf{X}[t] \mid \mathbf{X}[t-1])$. Each of these can be decomposed, similar to regular Bayesian networks, according to the dependency structure of the domain. The graphical representation of this model is consisted of two consecutive time slices. This representation is called a *2-TBN (2 time-slice BN)*.Figure 2.4(a) shows the interaction model for the two enzymes schematically. Figure 2.4(b) shows the 2-TBN representation of the probabilistic model. Since the time invariance assumption implies that the inter-time slice CPDs are correct for any t, this model defines a distribution over arbitrarily long sequences of time slices. For each length of time series T, we just need to reproduce the 2-TBN template $T-1$ times, and concatenate them, resulting in a legitimate Bayesian network. Figure 2.4(c) shows such an "unrolled" network for 4 time slices. Such a graphical model is called a *Dynamical Bayesian Network (DBN)*.

We note several properties of the DBN model. First, in our case both variables have *persistence edges* between time slices. This reflects the tendency of many variables to persist over time with high probability. Second, in our model all dependencies are between time slices. In a DBN there can also be connections within a time slice. These represent interactions (or dependencies) which are much faster than the inter-slice time interval, and are therefore modeled as instantaneous. Finally, we note that our example interaction model contains a cycle. Unlike static Bayesian networks, which do not allow cycles, the DBN model can capture cyclic dependencies by breaking the chain of dependencies according to time. This is a big advantage in modeling biological domains, which

often contain feedback loops.

Many popular dynamic models, such as hidden Markov models, or linear dynamical systems (also called Kalman filters) are special cases of DBNs. As for representing dependencies, any form of CPD we use in a BN can also be used in a DBN. Inference is essentially similar to the BN case, as it is done on the unrolled network model. It usually requires approximations, unless the domain is very small (Boyen and Koller, 1998; Doucet et al., 2000a; Murphy and Weiss, 2001). Parameter and structure learning are typically done on the 2-TBN representation (Friedman et al., 1998), but other than that are similar to what is done on static Bayesian networks.

Chapter 3

Gaussian Process Networks

3.1 Background

In the previous chapter we have introduced several options for representing dependencies in Bayesian networks. As our motivation comes from modeling data from gene expression experiments, we must ask ourselves: is any of these representations appropriate for our needs? What are the pros and cons of using each one of them in this context?

One simple approach to model expression data would be to treat each gene as being either in an "activated" or "suppressed" mode. This requires the discretization of the input data. Once the data is discretized we can use multinomial CPDs to model the dependencies between genes. As we shall see in Chapter 5, however, we loose much of the information in the data during the discretization process. Thus, we seek methods that can directly represent and learn interactions among continuous variables.

The best understood approach for modeling continuous distributions in Bayesian networks is based on linear Gaussian conditional densities (see Section 2.2.2). As we have shown, this form of continuous Bayesian network can be learned efficiently using exact Bayesian derivations. Unfortunately, the expressive power of Gaussian networks is limited. Formally, "pure" Gaussian networks can only learn *linear* dependencies among the measured variables. This is a serious restriction when learning in domains with non-linear interactions, or domains where the nature of the interactions is unknown. A common way of avoiding this problem is to introduce hidden variables that represent mixtures of Gaussians (e.g., Thiesson et al., 1998; Xu and Jordan, 1996).

Another problematic aspect of gene expression data is the large number of attributes (genes) that are measured (i.e., thousands) and the relatively few samples (i.e., dozens). Thus , we seek methods that are statistically robust and can detect dependencies among many possible alternatives.

In this chapter we address the problem of learning continuous networks by using *Gaussian Process* priors. This class of priors is a flexible semi-parametric regression model. We call the networks learned using this method *Gaussian Process Networks*. The resulting learning algorithm is capable of learning a large range of dependencies from data.

This approach has several important properties. First, the Gaussian Process regression method is inherently Bayesian. Thus, the integration of this form of regression into the Bayesian framework of model selection is natural and fairly straightforward. This allows us to interpret the learning results as posterior probabilities, and to assess the posterior probability of various networks structures (e.g., using methods such as presented in Friedman and Koller, 2003). Second, the semi-parametric nature of the Gaussian process prior allows to learn many continuous functional dependencies. This is crucial for exploratory data analysis where there is little prior knowledge on the form of interactions we may encounter in data. In addition, the Gaussian Process prior is biased to find functional dependencies among the variables in the domain. Thus, it is a useful prior for domains where we believe there is a direct causal dependency between attributes.

3.1.1 Why do we need non-parametric CPDs?

As we saw, common parametric forms for CPDs, such as linear Gaussian forms, impose constraints on the nature of interactions we can model. Multinomial CPDs, on the other hand, do not assume a specific form for the dependencies, but to use them we have to pay the price of discretizing the data.

An alternative approach, which tries to gain the benefits of both worlds, is modeling the dependencies using *non-parametric* forms of CPDs. In this approach, the conditional density of a variable X_i given its parents $\mathbf{U_i}$ is described using the instances of X_i and $\mathbf{U_i}$ in the data. In other words, the density can be written as:

$$P(X_i \mid \mathbf{U_i}) = f(X_i, \mathbf{U_i}; x[1], \ldots, x[M], \mathbf{u}_i[1], \ldots, \mathbf{u}_i[M], \theta) \tag{3.1}$$

where θ denote other parameters. The idea behind this description is, intuitively, assigning high density near regions which are highly represented in the data and low density in regions which have low representation. Typically, the additional parameters θ define how the density is smoothed (or interpolated) between data points, as well as outside their regions.

An example of such a non-parametric conditional density was suggested by Hofmann and Tresp (1996). Their method is based on a the well known practice of density estimation using kernel functions (Parzen, 1962; Rosenblatt, 1956). Roughly speaking, given training examples $\mathbf{x}[1], \ldots, \mathbf{x}[M]$, the kernel estimate for $P(\mathbf{X})$ is

$$P_{kernel}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} g\left(\frac{1}{\sigma} \left\|\mathbf{x} - \mathbf{x}[m]\right\|_2\right)$$

where $g()$ is a kernel function and σ is a "smoothing" parameter. A common choice is to take g to be the density function of a normal distribution with zero mean and unit variance. Figure 3.1 demonstrates kernel densities for two variables, X and U. Hofmann and Tresp use such estimates to find the conditional distribution by setting $P(x \mid \mathbf{u}) = P_{kernel}(x, u)/P_{kernel}(u)$.

Kernel methods are extremely flexible density estimators. As they are not committed to any

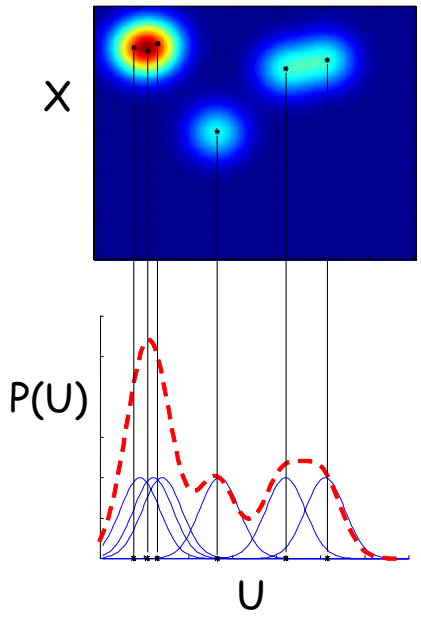


Figure 3.1: An illustration of *kernel* densities for two variables, X and U. The bottom plot shows the kernel density $P_{kernel}(u)$ (y-axis) as a function of U (x-axis). The density (dashed red) is composed of a sum of kernel functions centered around the data points (solid blue). The top plot shows the kernel estimate of the joint density $P_{kernel}(x, u)$.

parametric form, they can fit almost any density function. However, their performance depends crucially on the smoothness parameter, σ. When σ is too large, the density is over-smoothed, loosing much of the information in the samples. When σ is too small, the estimator overfits the data, resulting in a spiked density. We thus need to tune this parameter to avoid both overfitting and over-smoothing. This is usually done by *cross-validation* testing: Several partitions of the data set are made, where in each partition one part of the data set is used for training, or learning the model (in our case the density estimator), and the other part is used to test the performance of the model (in our case, by computing the log likelihood the estimator assigns to the test samples). We can now define the total log-likelihood obtained for all the test samples as an optimization objective, and optimize σ using this objective. Hofmann and Tresp use a *leave-one-out* cross-validation procedure: for a data set of size M they use M partitions, where in each partition $M - 1$ samples are used for training and the remaining sample is used for testing.

In addition to parameter optimization, we need to find a way of comparing the score of different network structures in this non-parametric setting. Hofmann and Tresp suggest to do so by comparing the cross-validated estimate of the logarithmic loss of each family. This is essentially an estimate of the out-of-sample loss the family will incur on new data. To summarize, for each family, Hofmann and Tresp's procedure searches for the parameters that minimize the log-loss in cross validation estimate, and then return this log-loss estimate as the score of the family.

3.2 Gaussian Process priors

In recent years, there has been much interest in the use of Gaussian Process priors for regression (Williams and Rasmussen, 1996) as well as for classification (Gibbs and MacKay, 1997). These priors possess some attractive features, and have a strong modeling power. For example, it can be shown that predictors like feed-forward neural networks and radial-basis function networks converge to Gaussian process predictors as the number of internal nodes goes to infinity (MacKay, 1998). The advantage of using Gaussian process predictors is that one can directly control properties of the learned functions (like periodicity, typical lengthscale and typical amplitude) directly through the prior's parameters, and that computations that involved complex parameter optimization in the network predictors are replaced by simple matrix operations. We now review the basics of Gaussian Processes, how they can be interpreted as priors and how they are used for regression.

Consider a set of variables $\mathbf{U}$. We want to model a prior over finite samples from an variable X which we believe to be a function of $\mathbf{U}$. We can treat the value of X for each value $\mathbf{u}$ as a random variable. Formally, a *stochastic process* over $\mathbf{U}$ is a function that assigns to each $\mathbf{u} \in Val(\mathbf{U})$ a random variable $X(\mathbf{u})$. Furthermore, for each finite set of coordinates $\mathbf{u}_{1:M} \equiv \{\mathbf{u}[1], \ldots, \mathbf{u}[M]\}$ the process assigns a joint probability distribution, $P(\mathbf{X}_{1:M} \mid \mathbf{u}_{1:M}, \mathbf{\Theta})$, where $\mathbf{X}_{1:M} \equiv \{X[1], \ldots, X[M]\}$ and $X[m] \equiv X(\mathbf{u}[m])$. One type of commonly used stochastic processes is discrete time processes. Here $\mathbf{U}$ takes discrete sequential values $t = 1, 2, \ldots$, usually referred to as time points. Examples of such processes are Bernoulli point processes and Markov processes of any order (see, e.g., Durrett (1991) for formal definitions). The joint probability supplied by the process can be used as a prior over finite samples from X.

A stochastic process is said to be a *Gaussian process* (GP) if for any finite sample $\mathbf{u}_{1:M}$, the assigned joint probability $P(\mathbf{X}_{1:M} \mid \mathbf{u}_{1:M}, \mathbf{\Theta})$ is a multivariate Gaussian distribution. We note that the domain $\mathbf{U}$ does not have to be a discrete set. It can equally consist of, for example, points in Euclidean space. Figure 3.2 illustrates the concept of a Gaussian Process.

To specify such a process, we need a way of describing the mean value of each variable $X(\mathbf{u})$ and the covariance matrix for each finite subset of values we choose. This is done, by specifying two functions:

- A mean function $\mu(\mathbf{u})$, so that $E[X(\mathbf{u})] = \mu(\mathbf{u})$.
- A covariance function $C(\mathbf{u}, \mathbf{u}')$, so that $Cov[X(\mathbf{u}), X(\mathbf{u}')] = C(\mathbf{u}, \mathbf{u}')$.

The joint distribution of $\mathbf{X}_{1:M}$ is therefore:

$$P(\mathbf{x}_{1:M}|\mathbf{u}_{1:M}) = \frac{1}{Z}\exp(-\frac{1}{2}(\mathbf{x}_{1:M} - \mu_{1:M})^T C_M^{-1}(\mathbf{x}_{1:M} - \mu_{1:M})) \tag{3.2}$$

where $\mu_{1:M}$ is the vector of means $\langle \mu(\mathbf{u}[1]), \ldots, \mu(\mathbf{u}[M]) \rangle$ and C_M is the M-by-M covariance matrix with the (i, j) entry $C(\mathbf{u}[i], \mathbf{u}[j])$. We note that by basic properties of Gaussian distributions, defining distributions in this manner guarantees consistency. For example, if $\mathbf{u}_{1:M}$ and $\mathbf{u}'_{1:N}$

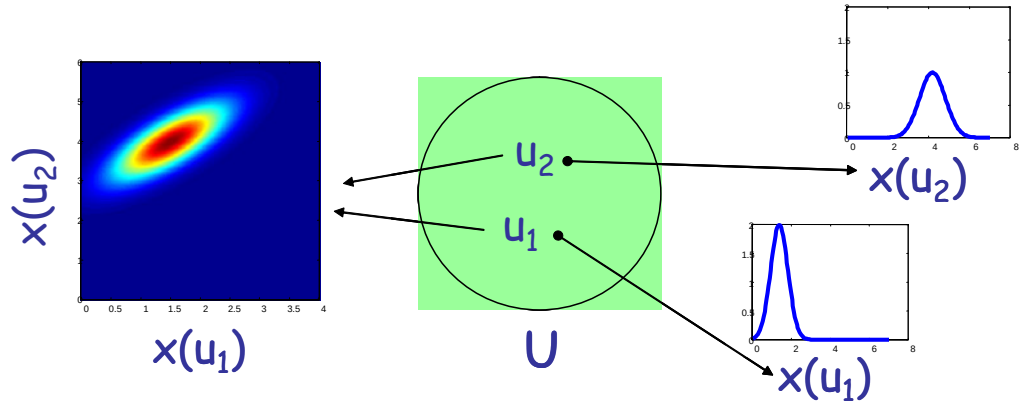


Figure 3.2: An illustration of a *Gaussian Process*. Each point u_i in the domain U (e.g. u_1, u_2) has a corresponding Gaussian random variable, $X(u_i)$. The joint density of a finite set of these variables (e.g. $P(X(u_1), X(u_2))$) is shown to be a multivariate Gaussian.

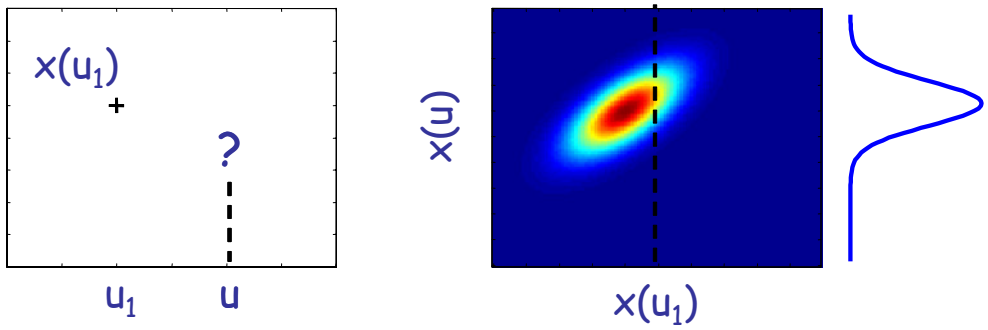


Figure 3.3: Prediction using a Gaussian Process prior. After seeing one sample $(u_1, x(u_1))$, we are interested in predicting the value of X at any given point u (*left*). The joint $P(X(u), X(u_1))$ is a two-dimensional Gaussian (*center*). The conditional $P(X \mid u, u_1, x_1)$, which is obtained by projecting the joint distribution on $x(u_1)$, is a univariate Gaussian (*right*).

are two samples with some overlapping points, then the marginals of their implied distributions $P(\mathbf{X}_{1:M}|\mathbf{u}_{1:M})$ and $P(\mathbf{X}'_{1:N}|\mathbf{u}'_{1:N})$ on the overlap set are the same.

3.2.1 Prediction

Before we discuss the covariance function C and its parameters, let us see how we use the GP to predict the value of the process at a new point. We shall assume $\mu(\mathbf{u}) = 0$ from now on.

Assume we already observed M points $\mathbf{x}_{1:M}$ given $\mathbf{u}_{1:M}$, and we are given a parametrized covariance function. By the definition of the Gaussian process $P(\mathbf{X}_{1:M}, X_{M+1} \mid \mathbf{U}_{1:M}, \mathbf{U}_{M+1})$ is an $M+1$-dimensional Gaussian distribution. Since we observed the values $\mathbf{X}_{1:M}$, we can compute the conditional distribution over X_{M+1} given these observations. A basic property of multivariate

Gaussian distributions is that the conditional distribution given the value of some of the variables is also a Gaussian distribution. Thus, the conditional distribution $P(X_{M+1} \mid \mathbf{X}_{1:M}, \mathbf{U}_{1:M}, \mathbf{U}_{M+1})$ is a univariate Gaussian distribution (see Figure 3.3). Using properties of Gaussian distributions we compute the mean and variance of this distribution using:

$$\begin{aligned} \mu_{M+1} &= \mathbf{k}^T C_M^{-1} \mathbf{x}_{1:M} \\ \sigma_{M+1} &= \kappa - \mathbf{k}^T C_M^{-1} \mathbf{k} \end{aligned} \tag{3.3}$$

where $\mathbf{k} = (C(\mathbf{u}[M+1], \mathbf{u}[1]), \ldots, C(\mathbf{u}[M+1], \mathbf{u}[M]))$ and $\kappa = C(\mathbf{u}[M+1], \mathbf{u}[M+1])$. In other words, having observed M values of the process we can represent the conditional density at any new coordinate x using C_M, the covariance matrix calculated for the first M points. Note that this falls under the definition of a non-parametric density we gave in Eq. (3.1): both C_M and $\mathbf{k}$ are parameterized using the first M data points.

3.2.2 Covariance Functions

We now deal with the issue of covariance functions. As we can see, this function determines a prior over functions. The only constraint on the covariance is that it should produce positive semidefinite matrices.

In general, if the covariance of two close points is large, then the prior prefers smooth functions. The covariance between points further away determines properties like periodicity, smoothness, and amplitude of the learned functions. These aspects of the covariance function are controlled by its *hyperparameters* θ. For example, Williams and Rasmussen (1996) suggest the following function:

$$C(\mathbf{u}, \mathbf{u}' : \theta) = \theta_0 \exp\left\{ -\frac{1}{2} \sum_{U_k \in \mathbf{U}}^{d} \frac{(u_k - u'_k)^2}{\lambda_k^2} \right\} + \theta_1 + \theta_2 \sum_{U_k \in \mathbf{U}}^{d} u_k u'_k + \theta_3 \delta_{\mathbf{u},\mathbf{u}'} \tag{3.4}$$

In this function each hyperparameter controls a different characteristic of the learned functions. The hyperparameter θ_0 controls the amplitude of variation of the function. The hyperparameter θ_1 controls how far can the whole function be shifted from the zero line. The hyperparameter θ_2 accounts for linear tendencies in the function. The hyperparameter θ_3 is the variance of a point-wise white noise term, which is uncorrelated between different points. The hyperparameters λ_k are the length scales of the different directions in $\mathbf{u}$, over which the function changes considerably.

What value of hyperparameters should we use in C when constructing the Gaussian Process density? The Bayesian approach is to assign the hyperparameters a prior, and then integrate over them. Let $\mathcal{D} = \{\mathbf{u}_{1:M}, \mathbf{x}_{1:M}\}$, then we should make predictions as

$$P(x[M+1] | \mathbf{u}[M+1], \mathcal{D}) = \int P(x[M+1] | \mathbf{u}[M+1], \mathcal{D}, \theta) P(\theta | \mathcal{D}) d\theta$$

As this integral is usually intractable, we can try to approximate it. One way is to use $\tilde{\theta}$, the *maximum a posteriori* estimator for θ, as suggested in Gibbs and MacKay (1997). Another option is to solve the integral numerically, e.g. by using a Monte Carlo method (as in Williams and Rasmussen, 1996). In this work we take the first approach, as we explain in the next section.

3.3 Learning Networks with Gaussian Process priors

We now examine Networks with Gaussian Process priors. We first note that given a network structure we can define conditional densities for any variable X given its parents in the network $\mathbf{U}$ using a Gaussian process prior, just as we discussed in the previous section. The structure along with these Gaussian process priors, one for each variable, comprise a full Bayesian network model, which we call a *Gaussian process network*.

We now turn to the problem of learning the structure of Gaussian process networks. We would like to define a Bayesian structure score, as described in Section 2.4.3, for this type of networks. After making the parameter independence and parameter and structure modularity assumptions described in Section 2.4.1, we can decompose this score into separate family scores. Recall that given a data set $\mathcal{D}$, the family score of X_i and its parents $\mathbf{U_i}$ is defined as

$$\text{FamScore}(X_i, \mathbf{U_i} \ : \ \mathcal{D}) \equiv P(x_i[1], \ldots, x_i[M] \mid \mathbf{u}_i[1], \ldots, \mathbf{u}_i[M], A[\mathbf{U}^G_{X_i} = \mathbf{U_i}])$$

These terms can be computed using the Gaussian Process prior for X_i as a function of $\mathbf{U_i}$. As before, we will assume that the mean function for this prior is 0, and thus we only need to choose a covariance function $C_{X_i|\mathbf{U_i}}$. Once we do that, the score is easy to compute. The Gaussian Process prior implies that $\mathbf{x}_{i,1:M} \equiv x_i[1], \ldots, x_i[M]$ are normally distributed with the covariance matrix C_M specified by the covariance function $C(\cdot, \cdot : \theta)$ and the parents' values $\mathbf{u_i}[1], \ldots, \mathbf{u}_i[M]$, and so:

$$\text{score}(X_i, \mathbf{U_i} \mid \mathcal{D}, \theta) = (2\pi)^{-\frac{M}{2}} |C_M|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \mathbf{x}^T_{i,1:M} C_M^{-1} \mathbf{x}_{i,1:M}\right)$$

We see that given a Gaussian Process prior, the computation of the marginal probability can be done in closed form. This prior is thus very appealing. It can learn a wide range of functional dependencies, and we can compute the Bayesian score exactly. In this sense, Gaussian Process priors fit well with the Bayesian model selection approach of learning Bayesian network structure.

In practice, we usually do not fix the parametrized covariance function in advance. Instead, we select a family of priors, such as the ones in Eq. (3.4), and define a prior over this family, $P(\theta)$. The proper Bayesian score would then require us to integrate over the hyperparameters:

$$\text{FamScore}(X_i, \mathbf{U_i} \ : \ \mathcal{D}) = \int \text{score}(X_i, \mathbf{U_i} \mid \mathcal{D}, \theta) P(\theta) d\theta$$

Unfortunately, whatever prior we choose, we do not know how to perform this integration in closed

form, since score$(X_i, \mathbf{U_i} \mid \mathcal{D}, \theta)$ is a complex function of θ. The approach we take, which is quite common in many other applications of Bayesian methods, is to approximate this integral with the MAP hyperparameters. This approximation is reasonable if the posterior probability over hyperparameters is sharply peaked over a single maximum. In such situations, most of the integral is determined by the area near the MAP parameters. A slightly better approximation is the Laplace approximation, where the posterior probability in the integral is approximated as a Gaussian distribution over the parameters θ (see, e.g., Chickering and Heckerman, 1997). This however requires the calculation of the Hessian of the log posterior probability, which can be time consuming. We therefore use an estimate for this term, which scales like $\frac{K}{2}log(N)$, where K in our case is the number of hyperparameters of the covariance function. The resulting estimate is in the spirit of the *Bayesian information criterion (BIC)* score we reviewed in Section 2.4.3, having a term which penalizes the model for over-complexity:

$$\text{FamScore}(X_i, \mathbf{U_i} \ : \ \mathcal{D}) = \max_{\theta} \text{score}(X_i, \mathbf{U_i} \mid \mathcal{D}, \theta) P(\theta) - \frac{K}{2} log(N)$$

To score a family X_i given $\mathbf{U_i}$, we perform conjugate gradient ascent to search for the MAP parameters. The evaluation of each point during the search requires to invert and to compute the determinant of an M-by-M matrix. Thus, the computational costs of this closed form equation is $O(M^3)$ in naïve implementations. This operation is repeated in each iteration of the hyperparameter optimization step. In practice this optimization converges quite rapidly (10-20 iterations). In the experiments reported here, we follow Rasmussen (1996) and use an inverse Gamma prior on the lengthscale hyperparameters, and a log-normal prior on the other hyperparameters. Finally, we can use the GP score defined above in conjunction with any structure search algorithm (such as greedy hill climbing described in Section 2.4.2), resulting in a procedure for learning Gaussian process networks.

3.4 Experimental Evaluation

We first want to test the GP score on the simplest case. We therefore ask the following question: Given two variables, X and Y, with some noisy functional dependence between them, will the GP network learner prefer the network where X is independent of Y, or the one in which they are dependent. Furthermore, we expect that, up to a certain noise level, the GP learner will prefer the direction for which it can fit a "nice" function, since such a function is more likely in a GP prior. For example, in Figure 3.4 we see a noisy quadratic dependence. The GP prior will assign a very low likelihood to the $X \rightarrow Y$ dependence, since it is hard to fit a function in this direction, while the dependence $Y \rightarrow X$ will get a higher probability, as it can be explained by a quadratic functional dependence with a certain noise width at each point.

To test this, we produced data sets of two variables with dependencies of linear, quadratic, cubic and sinusoidal nature. On top of the functional dependence, a non-correlated Gaussian noise

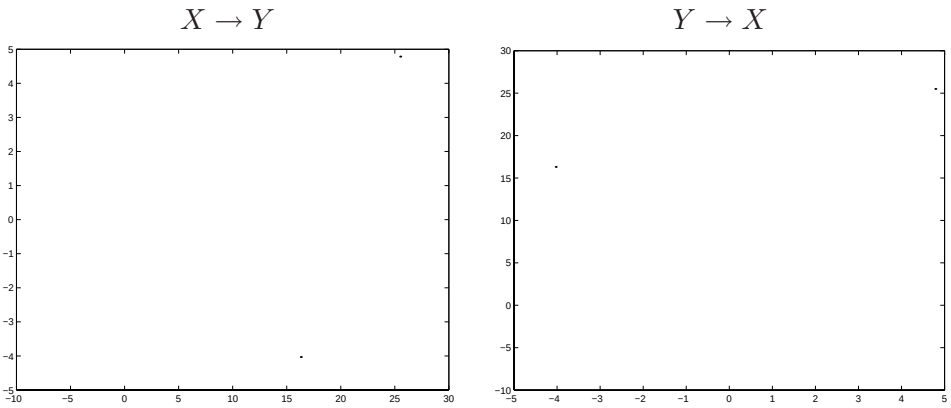


Figure 3.4: An example of a non-invertible dependence between X and Y. The explanation $X \rightarrow Y$ does not have a functional form, whereas $Y \rightarrow X$ can be explained as a noisy function.

was added. For each case we compared between the different network models, in terms of the GP network scores for the training set, and the log likelihood of the test set when that particular model was used for prediction. This was repeated for different noise levels, and different training set sizes. Figure 3.5 shows the dependence of those measures on the function noise level. We observe that for the true dependency model, the prediction quality and the GP family score rise as the level of noise drops. We see that even for noise levels as high as 1.5 times the dependent variable amplitude, the true dependence is still preferred over the no-dependence model. We also see that the direction of dependence is clear in the non-invertible cases, like the sinusoidal and the quadratic dependencies. In those cases, the score of the "wrong" direction dependency is as low as the no-dependency model. The cubic data set in our case is borderline-invertible, and so the distinction is less clear cut. For the linear case, if the slope is not too steep, both directions have a functional form, and so no one direction is preferred over the other. The Gaussian Process preference for functional direction can be useful when learning causal networks if we assume the interactions in our domain are functional.

We next compare the GP network learning method against two continuous variable models: the Gaussian network model with the BGe scoring metric described in Section 2.4.3, and the kernel network, described in Section 3.1.1. We start with two variable networks, with the same four types of functional relations as described before. Figure 3.6 shows the prediction quality of the three methods on those data sets, comparing the log loss of the predictions made by the dependent model to those made by the independent model. One can see that for the quadratic and sinusoidal relations, both far from linear, the Gaussian method prediction quality is the same for both models, while the GP learner continually performs better with the dependent model. The kernel method, which is insensitive to directionality or linearity, also performs better with the dependent model. This difference between the three methods is demonstrated in Figure 3.7, which compares the conditional density estimation of the methods on a data set with a clear functional direction. One can see that the GP estimator is the most sensitive to the direction of the dependency, both in terms of quality

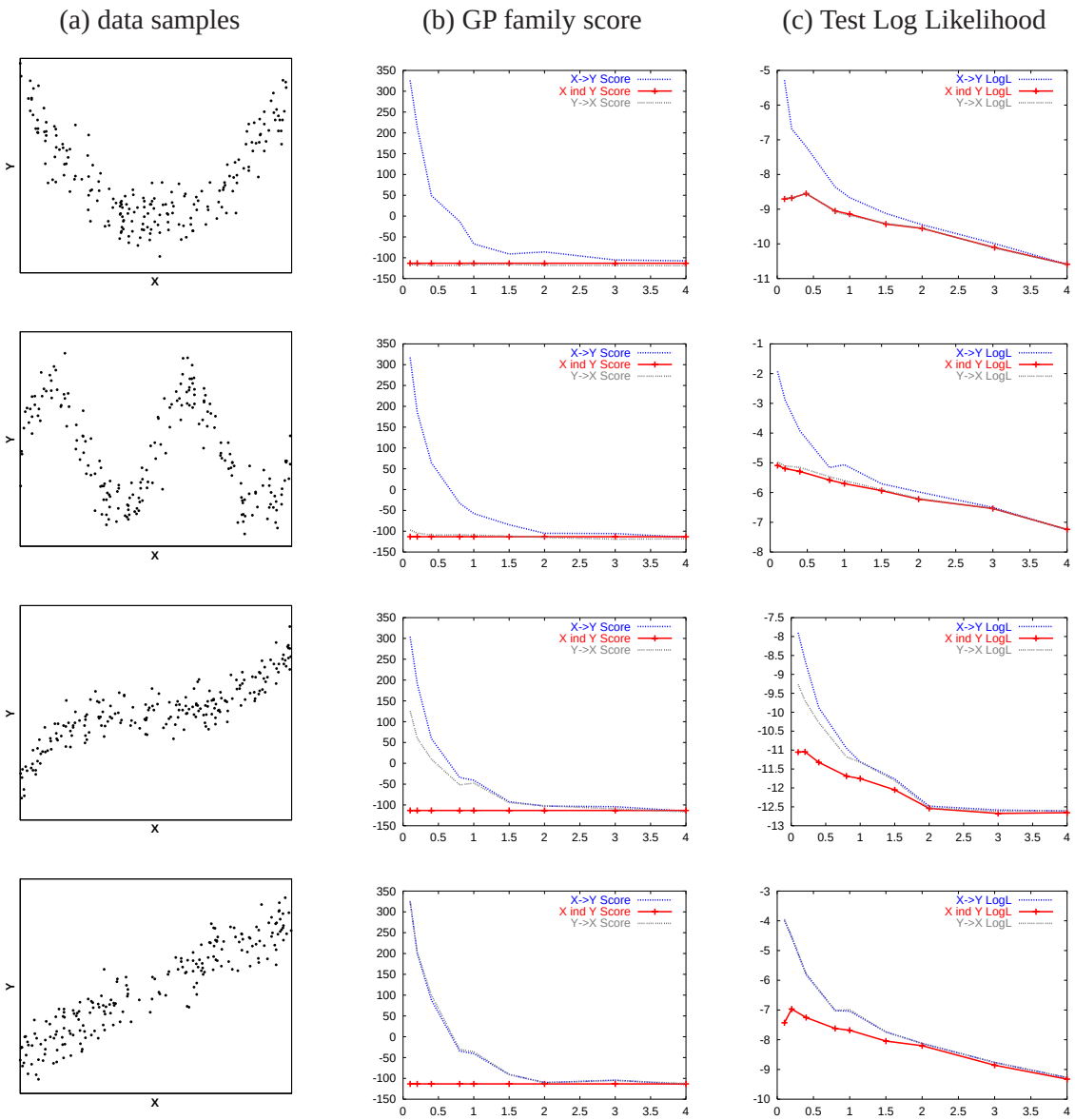


Figure 3.5: GP family scores and prediction accuracy as a function of sample noise, for different functional dependencies. (a) Data sets of quadratic, sinusoidal, cubic and linear dependencies of Y on X. In the shown sets the noise level on Y is 0.4 times its range. (b-c) Family scores and prediction accuracy for 3 network models: The no-dependency network, the $X \to Y$ "true" network and the $Y \to X$ "opposite direction" network. The X axis is the sample noise on Y in units of its range. The Y axis is the GP score for each family (b) or the average log loss of a test data set following parameter optimization on a different training set (c).

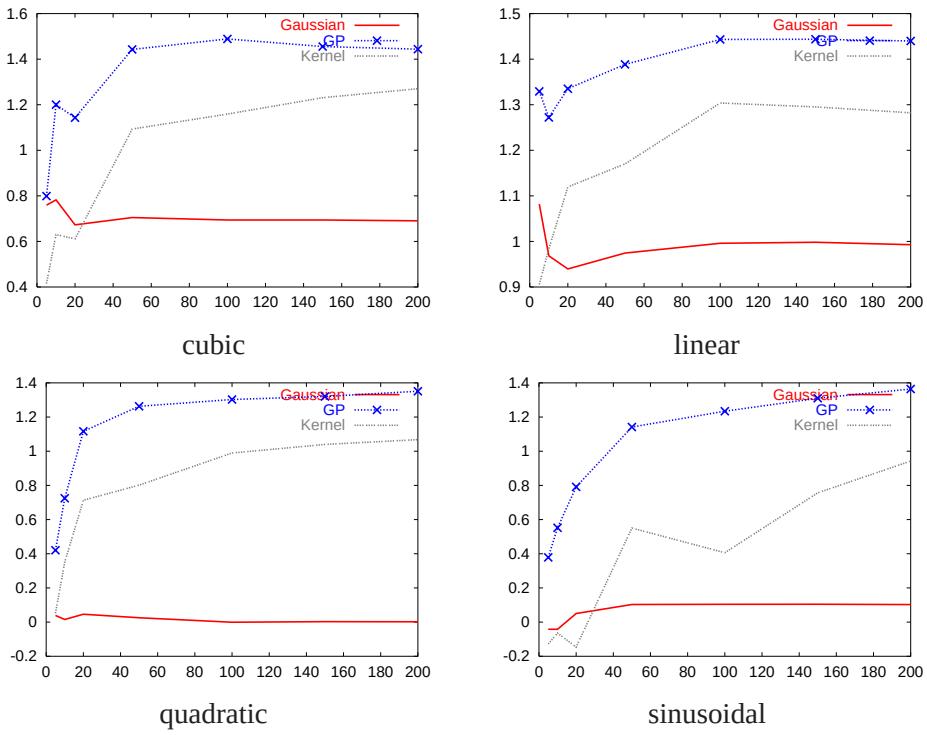


Figure 3.6: Sample complexity comparison for the Gaussian, Gaussian Process and Kernel methods. The plots show log likelihood ratio of the test set, between the no-dependency model and the $X \rightarrow Y$ model (*x-axis*) vs. number of training samples (*y-axis*). Four different functional relations between X and Y were tested (see Figure 3.5). Both training and test set have a noise level of 0.4 standard deviation of the dependent variable.

of fit, as well as in the qualitative prediction. Unlike the two other methods, it captures the noisy sinusoidal dependency, interpolating quite well into the middle range which is not represented in the data.

We now turn to comparing the reconstruction capability of the three methods. We start with small artificial networks with different functional relations, and check which method reconstructs the true network with higher accuracy. We sampled 50 and 100 instance data sets from 3 variable networks of all possible architectures, with linear, quadratic, sinusoidal or mixed functional relations. A non-correlated noise of width 0.4 of the variable's amplitude was added. We applied the three network learning methods on these data sets. Both GP and kernel methods performed well in reconstructing the correct skeleton of the generating network, with the GP performing only slightly better. However, the GP does significantly better in identifying the original DAG for data sets with non-invertible connections (quadratic and sinusoidal). In those cases, as expected, the GP learner orients the arcs in the "true" functional direction, while the kernel method does not necessarily do so. The Gaussian network model does not perform as well in this task, where in most of the cases

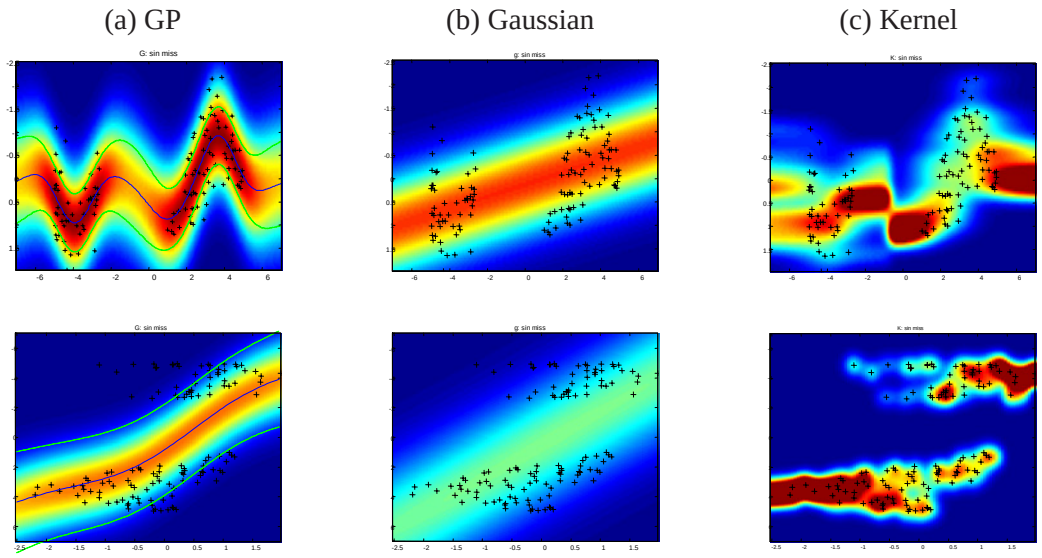


Figure 3.7: Conditional density estimation for Gaussian Process, linear Gaussian and kernel estimators. Plots on the top row show estimation of $P(Y|X)$ from a sample of points where the dependence between Y and X has a noisy sine-wave form, and the middle region in the joint space is not represented. The samples are shown as '+' marks (X is along the x-axis). The color scheme shows the estimated conditional density $P(Y|X)$ for each value (x, y). Bottom row: estimates of $P(X|Y)$ for the same data set (the axis are interchanged).

with non-linear connections, the learned networks are missing some of the arcs. This is not surprising, since the best linear-Gaussian model one can fit to a non-linear function often has a large variance, making this connection low scoring.

3.4.1 Real life data

We next tested Gaussian Process Networks on real world data sets of continuous attributes, comparing it to the other two methods. We use three data sets from the UCI machine learning repository (Blake and Merz, 1998). These data sets are:

- **Boston housing data set** - a data set describing different aspects of neighborhoods in the Boston area, and the median price of houses in those neighborhoods. The data set contains 506 samples with 14 attributes.
- **Abalone data set** - a data set of physical measurements of abalones. The data set contains 4177 samples with 9 attributes.
- **Glass identification data set** - a data describing the material concentrations in glasses, with a class attribute denoting the type of the glass. The data set contains 214 samples with 10 attributes.

Table 3.1: Average Log Loss on an independent test set achieved by the three methods for different training set sizes.

	Boston			Abalone			Glass		
Size	Gaussian	GP	Kernel	Gaussian	GP	Kernel	Gaussian	GP	Kernel
10	-53.7	-2.8×10^4	-56.2	-322.4	-319.8	-410.5	-43.8	-153.8	-72.8
20	-40.9	-447.8	-40.6	0.6	-0.1	-9.3	-10.4	-74.4	-52.8
50	-37.1	-44.7	-47.7	4.5	10.3	-8.1	-6.6	-51.3	-84.0
100	-34.4	-50.7	-132.3	7.6	11.5	-7.0	-3.3	-52.9	-35.4
150	-32.3	-70.3	4.4	9.3	13.1	-6.5	-2.5	-2.0	-42.8
200	-31.0	-43.5	8.3	10.5	13.1	-34.1			
300				12.0	12.9	-5.3			

For each data set, we performed structure learning with each method, using subsets of the original data set, which was permuted in a random order. We then used the learned structure and the optimized parameters to predict the likelihood of the corresponding test set, which consisted of a separate subset of the data set. Test set sizes varied between 64 samples (Glass data set) and 300 samples (Boston and Abalone sets). Some of the attributes in those data sets are either discrete (such as class attributes), or have only few values in the data. To accommodate these variables, we used the hybrid approach described in Section 2.2.3. In this approach, all discrete variables are forced to precede all continuous variables. For each continuous variable X having some mixture of continuous parents $\mathbf{U}_c$ and discrete parents $\mathbf{U}_d$, we model the distribution $P(X \mid \mathbf{U}_c)$ separately for each state of U_d. The score for such a family is given by

$$\text{score}(X, \mathbf{U}_c, \mathbf{U}_d \mid \mathcal{D}) = \sum_{\mathbf{u}_d \in \mathcal{U}_d} \text{score}(X, \mathbf{U}_c \mid \mathcal{D}_{\mathbf{u}_d})$$

where $\mathcal{U}_d$ is the set of values taken by $\mathbf{U}_d$, and $\mathcal{D}_{\mathbf{u}_d}$ is the subset of data where $\mathbf{U}_d$ have values $\mathbf{u}_d$.

Table 3.1 lists the average log likelihood of the test set, for each method and each training set size. We note that both in the glass and abalone domains the Gaussian process method performs well compared to the Gaussian model, while the kernel method does not do as well. On the Boston domain, however, the kernel method seems to rate quite high. This is due to one variable (index of accessibility to radial highways) which only has nine values appearing in the data set. The kernel method assigns no parents to this variable, and learns a distribution composed of sharp "delta" peaks around those values. In cases like this, the kernel method has to be bounded not to learn distributions which are too sharp. Another option is to treat those variables as discrete. In general, however, the Gaussian and GP methods, modeling only function-like relations, can not account for multi modal distributions.

Figure 3.8 shows two examples from the abalone domain of connections learned by the Gaussian Process network, plotted with the training samples. The GP learner clearly fits a non-linear function to the data, whose width varies according to the density of points in each area. Beyond the range of

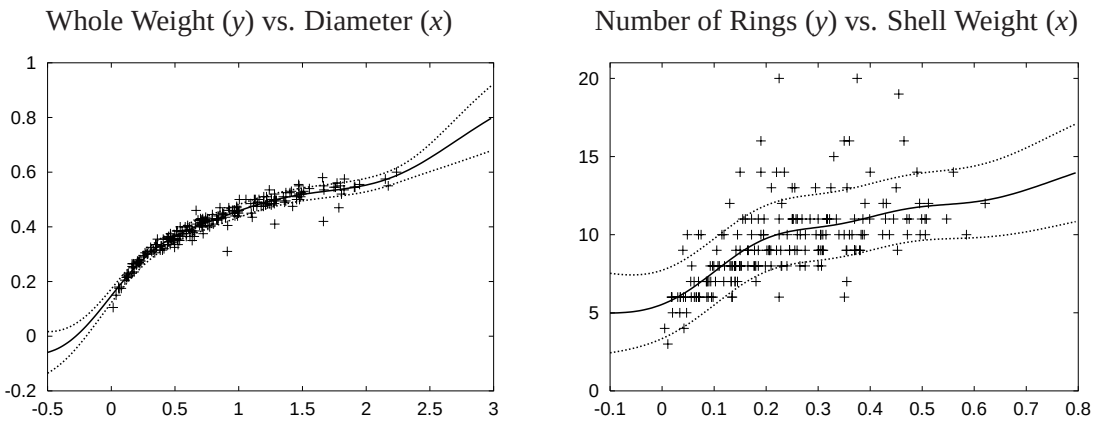


Figure 3.8: Two examples of function predictions made by the GP learner on the abalone data set. The data set points of the two attributes are shown as scattered '+' marks. The solid lines show the predicted mean function (mean of $P(y \mid x)$) and its width (one standard deviation) at each point.

sample points, the width of the predicted function rises, as the uncertainty increases. The figure on the right is an example where the dependent variable is semi-discrete (number of rings), showing that the method is capable of handling this type of data as well. These examples show that by learning Gaussian Process networks we can discover interesting relations even under noisy measurements.

3.5 Discussion

In this chapter we introduced the notion of Gaussian Process networks and developed the Bayesian score for learning them. We report on results that show that this method generalizes well from noisy data. The combination of this powerful regression technique with the flexible language of Bayesian networks seems like a promising tool for exploratory data analysis, causal structure discovery, prediction, and Bayesian classification.

There are several methods closely related to Gaussian Processes that are relevant to this work. Wahba (1990, 2000) makes a connection between Gaussian processes and reproducing kernel Hilbert spaces (RKHS), showing that the solution to the posterior Bayesian estimate of the Gaussian process (as in Eq. (3.3)) is also the solution to a spline smoothing problem posed as a variational minimization problem in an RKHS. The smoothing parameter is optimized using cross validation methods, whereas in the case of Gaussian process priors, we use a MAP estimate for the hyperparameters. In related works (e.g. Wahba, 1991), the relevance of the different components of the function is estimated from the learned smoothing parameters. In Gaussian process methods there is a similar notion, judging the relevance of different input dimensions by their estimated lengthscales in the covariance function (MacKay, 1998). Inputs with estimated large lengthscales are deemed less relevant, because the function hardly changes in those directions. A promising direction for future research is guiding the search in the network space by those learned lengthscales, resulting in a

more efficient and accurate search procedure. This is important when using the Gaussian process score method, as its computation is costly.

The main cost of computing Gaussian process conditional densities (or scores) comes from inverting the covariance matrix C_M, a computation which costs $O(M^3)$. For single family computations this is feasible up to around 1000 data points, but when performing structure search, combined with hyperparameter optimization, this computation has to be repeated many times. This makes the application of the method to data sets of more than a few hundred instances very hard. In recent years, the computational cost of GP regressors yielded a number of suggested solutions, including sparse representations (Csato and Opper, 2002; Lawrence et al., 2003; Smola and Bartlett, 2001), decomposed representations (Tresp, 2001) and approximate computations (Gibbs and MacKay, 1997; Williams and Seeger, 2001). Incorporating any one of these methods will enable learning from larger data sets.

Another issue is the approximation method used for evaluating the marginal likelihood. The assumption that the posterior probability has a unique sharp peak does not always hold. We therefore want to test methods which do not rely on the MAP estimate of the hyperparameters. Such methods include Monte Carlo sampling methods, as discussed in Williams and Rasmussen (1996).

In the current work we dealt only with learning from fully observed data. However, in case we are dealing with partial observations, or when our model includes hidden variables, we typically need to perform inference in the network for computing a posterior over the missing values. Though inference is well defined in Gaussian Process networks (similarly to other non-linear networks), it involves estimating integrals which do not have a closed form. We therefore have to devise appropriate approximations, such as particle-based inference methods (Doucet et al., 2001, 2000b), which can get quite costly.

Finally, when coming to select a family of models for a specific problem, such as modeling gene regulation networks, we must take care that this family is of the right complexity, or representation power, for the problem and the amount of data at hand. It is quite possible that the Gaussian process representation is too flexible in this context. If we are interested not only in generalization power, but also in learning a model which is close to the real biology, restricting ourselves to a specific parametric form of dependencies can work for us, given that we have a sound basis for this particular parametrization. In Chapter 6 we develop such a representation, which is based on the details of the biochemical reactions modeled.

Chapter 4

Structure Learning Methods for Bayesian Networks

In the previous chapters we have reviewed Bayesian networks, including several representations of dependencies in them as well as some standard methods for learning their structure and parameters. These learning methods have been applied successfully in many domains (e.g. Friedman and Goldszmidt, 1998; Geiger and Heckerman, 1994; Hofmann and Tresp, 1996; Lerner, 2002), and though the learning problem itself is NP-hard, heuristic methods often yield very good results. However, our domains of interest, namely gene regulatory systems, possess some properties which make the learning task especially hard.

First, these domains are usually quite large, in terms of the number of variables. Depending on the system studied, we typically want to learn structure over domains with several hundred to several thousand variables. Now, the size of structure space is super-exponential in N, the number of variables (Harary and Palmer, 1973). For example, for $N = 10$ there are in the order of 10^{12} distinct structures, while for $N = 13$ the order climbs to 10^{21}. It is clear then that with hundreds of variables, all heuristic methods cannot cover but a tiny fraction of structure space during the search.

Second, when coming to model regulatory networks in a realistic way, we will want to use conditional distribution models which can capture the behavior of regulatory interactions. These interactions, as we shall see in Chapter 6 are not expected to be linear. Whether they are modeled in a non-parametric way, as with Gaussian process priors, or with a specialized parametric model (which we will introduce in Chapter 6), the structure scores based on those models require intensive computation, as they involve non-linear parameter optimization.

In this chapter we try to cope with these difficulties by introducing two heuristic methods for speeding up structure search. Both methods try to focus the search on promising candidate structures. Both do this on a local scale, i.e. on a per-variable basis, using two different approaches.

The *"Sparse Candidate"* algorithm works iteratively. First, it restricts the parents of each variable to belong to a small subset of *candidates*. It then searches for a network that satisfies these

constraints. The learned network is then used as the basis for selecting better candidates for the next iteration.

The *"Ideal Parent"* method focuses on continuous variable network learning. It efficiently identifies good candidates for adding or replacing a parent to a variable. First, it computes what form an ideal parent profile would take, based on the parametric form of the variable's CPD. It then compares existing variables to this profile. Importantly, this method also facilitates the addition of new hidden variables into the network structure efficiently.

In the next sections we describe the two methods, and give some experimental results showing their advantages. At the end of this chapter we tie some knots between these methods, suggesting ways to combine them in order to further speed up structure search.

4.1 Learning from large domains: The "Sparse Candidate" Algorithm

In Section 2.4.2 we presented the problem of structure learning in Bayesian networks, and described some heuristic algorithms for learning structure. Such "generic" search procedures do not apply any knowledge about the expected structure of the network to be learned. For example, greedy hill-climbing search procedures examine all possible local changes in each step and apply the one that leads to the biggest improvement in score. The usual choice for "local" changes are edge addition, edge deletion, and edge reversal. Thus, there are approximately $O(N^2)$ possible changes where N is the number of variables.[1]

The cost of these evaluations becomes acute when we learn from massive data sets. Since the evaluation of new candidates requires collecting various statistics about the data, it becomes more expensive as the number of instances grows. To collect these statistics, we usually need to perform a pass over the data. Although, recent techniques (e.g., (Moore and Lee, 1997)) might reduce the cost of this collection activity, we still expect non trivial computation time for each new set of statistics we need.

It seems, however, that most of the candidates considered during the search can be eliminated in advance by using our statistical understanding of the domain. For example, in greedy hill-climbing, most possible edge additions might be removed from consideration. If X and Y are almost independent in the data, we might decide not to consider Y as a parent of X. Of course, this is a heuristic argument, since X and Y can be marginally independent, yet have strong dependence in the presence of another variable (e.g., X is the XOR of Y and Z). In many domains, however, it is reasonable to assume that this pattern of dependencies does not appear. For many continuous variable conditional density families, like linear Gaussians, these patterns are not possible at all.

The use of a measure of dependence, such as the *mutual information*, between variables to guide network construction is not new. For example, Chow and Liu's algorithm (Chow and Liu, 1968) uses the mutual information to construct a tree-like network that maximizes the likelihood

[1] Some of these changes introduce cycles, and thus are not evaluated. Nonetheless, the number of feasible operations is usually quite close to $O(N^2)$.

score. When we consider networks with larger in-degree, several authors use the mutual information to greedily select parents (Cheng et al., 1997, 2001). However, these authors do not attempt to maximize any statistically motivated score. In fact, it is easy to show situations where these methods can learn erroneous networks. This use of mutual information is a simple example of a statistical cue. Other cues can come about from examining algorithms of constraint-based approaches to learning. In this section, we incorporate similar considerations within a procedure that explicitly attempts to maximize a score. We provide an algorithm that empirically performs well in massive data sets.

The general idea is quite straightforward. We use statistical cues from the data, to restrict the set of networks we are willing to consider. This is done by restricting the possible parents of each node to a specific set of candidates. We then search for the highest scoring structure satisfying those restrictions. Any search techniques we use in this case will perform faster, since the search space is significantly restricted. Moreover, as we show, in some cases we can find the best scoring network satisfying these constraints. In other cases, we can use the constraints to improve our heuristics.

Of course, such a procedure might fail to find a high-scoring network: a misguided choice of candidate parents in the first phase can lead to a low scoring network in the second phase, even if we manage to maximize the score with respect to these constraints. The key idea of our algorithm is that we use the network we found at the end of the second stage to find better candidate parents. We then can find a better network with respect to these new restrictions. We iterate in this manner until convergence.

In the next section we outline the structure of our "Sparse Candidate" algorithm and show that there are two orthogonal issues that need to be resolved: how to select candidates in each iteration, and how to search given the constraints on the possible parents. We examine these issues in Section 4.1.2 and Section 4.1.3, respectively. In Section 4.1.4 we evaluate the performance of the algorithm on synthetic and real-life datasets. We conclude with a discussion of related work and future directions in Section 4.1.5.

4.1.1 The "Sparse Candidate" Algorithm

We now outline the framework for our *Sparse Candidate* algorithm The underlying principle for our algorithm is fairly intuitive. It calls for two variables with a strong dependency between them to be located "near" each other in the network. The strength of dependency between variables can often be measured using mutual information or correlation (Cover and Thomas, 1991). In fact, when restricting the network graph to a tree, Chow and Liu's algorithm (Chow and Liu, 1968) does exactly that. It measures the mutual information (formally defined below) between all pairs of variables and selects a maximal spanning tree as the required network.

We aim to use a similar argument for finding networks that are not necessarily trees. Here, the general problem is NP-hard (Chickering, 1995). However, a seemingly reasonable heuristic is to select pairs (X, Y) with high dependency between them and create a network with these edges. This approach however, does not take more complex interactions into account. For example, if the

Input : A data set $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$
An initial network B_0
A decomposable score: $\mathrm{Score}(B : \mathcal{D}) = \sum_i \mathrm{FamScore}(X_i, \mathbf{U}_i^B : \mathcal{D})$
A parameter k
Output : A network B

for $n = 1, 2, \ldots$ *until convergence* **do**
Restrict : Based on $\mathcal{D}$ and B_{n-1}, select for each variable X_i a set C_i^n ($|C_i^n| \leq k$) of candidate parents.
This defines a directed graph $H_n = (\mathcal{X}, E)$, where $E = \{X_j \rightarrow X_i | \forall i, j, X_j \in C_i^n\}$. (Note that H_n is usually cyclic.)
Maximize : Find network $B_n = \langle G_n, \Theta_n \rangle$ maximizing $\mathrm{Score}(B_n : \mathcal{D})$ among networks that satisfy $G_n \subseteq H_n$ (i.e., $\forall X_i$, $\mathbf{U}_i^{G_n} \subseteq C_i^n$,)
end
Return B_n

Algorithm 2: Outline of the *Sparse Candidate* algorithm

true structure includes a substructure of the form $X \rightarrow Y \rightarrow Z$, we might expect to observe a strong dependency between X and Y, Y and Z, and also between X and Z. However, once we consider both X and Y as parents of Z, we might recognize that X does not help in predicting Z given the value of Y.

Our approach is based on the same basic intuition of using mutual information, but we do so in a refined manner. We use measures of dependency between pairs of variables to *focus* our attention during the search. For each variable X, we find a set of variables $Y_1, \ldots, Y_k$ that are the most promising *candidate* parents for X. We then *restrict* our search to networks in which only these variables can be parents of X. Thus, instead of having $N - 1$ potential parents for a node, we only consider k possible parents, where $k \ll N$. This means a we work in a much smaller search space in which we can hope to find a good structure quickly. Though this procedure limits the number of parents of each variable by some constant k, in many real world situations it is reasonable limitation.

The main drawback of this procedure is, that once we choose the candidate parents for each variable, we are committed to them. Thus, a mistake in this initial stage can lead us to find an inferior scoring network. We therefore iterate the basic procedure, using the constructed network to reconsider the candidate parents and choose better candidates for the next iteration. In the example of $X \rightarrow Y \rightarrow Z$, if Y is detected as a parent of Z in the first iteration, X would not be chosen as a candidate for Z in the next iteration, allowing a variable with weaker dependency to replace it.

The resulting procedure has the general form shown in Algorithm 2. This framework defines a whole class of algorithms, depending on how we choose the candidates in the **Restrict** step, and how we perform the search in the **Maximize** step. The choice of methods for these two steps are mostly independent of one another. We examine each of these in detail in the next two sections.

Input : Data set $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$
A network B_n
A measure M
A parameter k
Output : For each variable X_i a set of candidate parents C_i of size k

forall X_i , $i = 1, \ldots, n$ **do**
Calculate $\mathrm{M}(X_i, X_j)$ for all $X_j \neq X_i$ such that $X_j \notin \mathbf{U_i}$
Choose $x_1, \ldots, x_{k-l}$ with highest ranking , where $l = |\mathbf{U_i}|$
Set $C_i = \mathbf{U_i} \cup \{x_1, \ldots, x_{k-l}\}$
end
Return $\{C_i\}$

Algorithm 3: Outline of the *Restrict* step

Before we go on to discuss these issues, we address the convergence properties of these iterations. Clearly, at this abstract level, we cannot say much about the performance of the algorithm. However, we can easily ensure its monotonic improvement. We require that in the **Restrict** step, the selected candidates for X_i's parents include X_i's current parents, i.e., the selection must satisfy $\mathbf{U}_i^{G_n} \subseteq C_i^{n+1}$ for all X_i.

This requirement implies that the network B_n is a legal structure in iteration $n+1$. Thus, if the search procedure at the **Maximize** step also examines this structure, it must return a structure that scores at least as well as B_n. Immediately, we get that $\mathrm{Score}(B_{n+1} : \mathcal{D}) \geq \mathrm{Score}(B_n : \mathcal{D})$.

Another issue is the stopping criteria for our algorithm. There are two types of stopping criteria: a *score based* criterion that terminates when $\mathrm{Score}(B_n : \mathcal{D}) = \mathrm{Score}(B_{n-1} : \mathcal{D})$, and a *candidate based* criterion that terminates when $C_i^n = C_i^{n-1}$ for all i. Since the score is a monotonically increasing bounded function, the score based criterion is guaranteed to stop. However, the candidate based criterion might be able to continue to improve after an iteration with no improvement in the score. It can also enter a non-terminating cycle, therefore we need to limit the number of iterations with no improvement in the score.

4.1.2 Choosing Candidate Sets

In this section we discuss possible measures for choosing the candidate set. To choose candidate parents for X_i, we assign each X_j some measure of relevance to X_i. As the candidate set of X_i, we choose those variables with the highest measure. This general outline is shown inAlgorithm 3. It is clear that in some cases, such as XOR relations, pairwise scoring functions are not enough to capture the dependency between variables. However, for computational efficiency we limit ourselves to this type of functions.

When considering each candidate, we essentially assume that there are no spurious independencies in the data. More precisely, if Y is a parent of X, then X is not independent (or "almost"

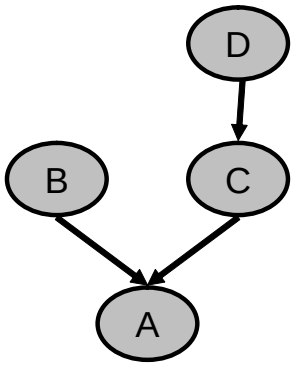


Figure 4.1: Network for Example 4.1.1

independent) of Y, given any subset of the other parents.

A simple and natural measure of dependence is *mutual information*:

$$I(X;Y) = \sum_{x,y} \hat{P}(x,y) \log \frac{\hat{P}(x,y)}{\hat{P}(x)\hat{P}(y)}$$

Where $\hat{P}$ denotes the observed frequencies in the dataset. The mutual information is always non-negative. It is equal to 0 when X and Y are independent. The higher the mutual information, the stronger the dependence between X and Y.

Researchers have tried to construct networks based on $I(X;Y)$, i.e., add edges between variables with high mutual information (Chow and Liu, 1968; Ezawa and Schuermann, 1995; Sahami, 1996). While in many cases mutual information is a good first approximation of the candidate parents, there are simple cases for which this measure fails.

Example 4.1.1: Consider the four-variable network illustrated in Figure 4.1. We can easily select parameters for this network such that $I(A;C) > I(A;D) > I(A;B)$. Thus, if we select only two parents based on mutual information, we would select C and D. These two, however, are redundant since once we know C, D adds no new information about A. Moreover, this choice does not take into account the effect of B on A. ∎

This example shows a general problem in pairwise selection, which our iterative algorithm overcomes. After we select C and D as candidates, and the learning procedure hopefully only sets C as a parent of A, we reestimate the relevance of B and D to A. How can this be done with the mutual information? We outline two possible approaches:

The first approach is based on an alternative definition of the mutual information. We can define the mutual information between X and Y as the distance between the distribution $\hat{P}(X,Y)$ and the distribution $\hat{P}(X)\hat{P}(Y)$, which assumes X and Y are independent:

$$I(X;Y) = D_{KL}(\hat{P}(X,Y) \| \hat{P}(X)\hat{P}(Y))$$

where $D_{KL}(P\|Q)$ is the *Kullback-Leibler divergence*, defined as:

$$D_{KL}(P(X)\|Q(X)) = \sum_X P(X) \log \frac{P(X)}{Q(X)}.$$

Thus, the mutual information measures the error we introduce if we assume that X and Y are independent. If we already have an estimate of a network B, we can use a similar test to measure the *discrepancy* between our estimate $P_B(X, Y)$ and the empirical estimate $\hat{P}(X, Y)$. We define

$$\mathrm{M}_{\mathrm{Disc}}(X_i, X_j \mid B) = D_{KL}(\hat{P}(X_i, X_j)\|P_B(X_i, X_j))$$

Notice that when B_0 is an empty network, with parameters estimated from the data, we get that $\mathrm{M}_{\mathrm{Disc}}(X, Y \mid B_0) = I(X : Y)$. Thus, our initial iteration in this case uses mutual information to select candidates. Later iterations use the discrepancy to find variables for which our modeling of their joint empirical distribution is poor. In our example, we would expect that $P_B(A, B)$ in the network, when only C is a parent of A, is quite different from $\hat{P}(A, B)$. Thus, B would measure highly relevant to A, while $P_B(A, D)$ would be a good approximation of $\hat{P}(A, D)$. Therefore, even "weak" parents have the opportunity to become candidates at some point.

One of the issues with this measure is that it requires us to compute $P_B(X_i, X_j)$ for pairs of variables. When learning networks over large number of variables this can be computationally expensive. However, we can easily approximate these probabilities by using a simple sampling approach. Unlike computation of posterior probabilities given evidence, the approximation of such prior probabilities is not hard. We simply sample N instances from the network, and from these we can estimate all pair-wise interactions. (In our experiments we use $N = 1000$.)

The second approach to extend the mutual information score is based on the semantics of Bayesian networks. Recall that in a Bayesian network X_i's parents *shield* it from its non-descendants. This suggests that we measure whether the conditional independence statement "X_i is independent of X_j given $\mathbf{U_i}$" holds. If it holds, then the current parents separate X_j from X_i and X_j is not a parent of X_i. On the other hand, if it does not hold, then either X_j is a parent of X_i, or X_j is a descendant of X_i.

Instead of testing whether the conditional independence statement holds or not, we estimate how strongly it is violated. The natural extension of mutual information for this task, is the notion of *conditional mutual information*:

$$I(X; Y|Z) = \sum_Z \hat{P}(Z) D_{KL}(\hat{P}(X, Y|Z)\|\hat{P}(X|Z)\hat{P}(Y|Z))$$

This measures the error we introduce by assuming that X and Y are independent given different values of Z. We define

$$\mathrm{M}_{\mathrm{Shield}}(X_i, X_j \mid B) = I(X_i; X_j|\mathbf{U_i})$$

Once again, we have that if B_0 is the empty network, then this measure is equivalent to $I(X_i; X_j)$. Although shielding can remove X's ancestors from the candidate set, it does not "shield" X from its descendants.

A deficiency of both these measures is that they do not take into account the cardinality of various variables. For example if both Y and Z are possible candidate parents of X, but Y has two values (one bit of information), while Z has eight values (three bits of information), we would expect that Y is less informative about X than Z. On the other hand, we can estimate $P(X|Y)$ more robustly than $P(X|Z)$ since it involves fewer parameters.

Such considerations lead us to use scores which penalize structures with more parameters, when searching the structure space, since the more complex the model is, the easier we are misled by the empirical distribution. We use the same considerations to design such a score for the Restrict step.

To see how to define a measure of this form, we start by reexamining the shielding property. Using the chain rule of mutual information:

$$I(X_i; X_j|\mathbf{U_i}) = I(X_i; X_j, \mathbf{U_i}) - I(X_i; \mathbf{U_i})$$

That is, the conditional mutual information is the additional information we get by predicting X_i using X_j and $\mathbf{U_i}$, compared to our prediction using $\mathbf{U_i}$. Since the term $I(X_i; \mathbf{U_i})$ does not depend on X_j, we don't need to compute it when we compare the information that different X_j's provide about X_i. Thus, an equivalent comparative measure is

$$\mathrm{M}_{\mathrm{Shield}}(X_i, X_j \mid B) = I(X_i; X_j, \mathbf{U_i})$$

Now, if we consider the score of the Maximize step as a cautious approximation of the mutual information, with a penalty on the number of parameters, we can get the *score* measure;

$$\mathrm{M}_{\mathrm{Score}}(X_i, X_j \mid B) = \mathrm{FamScore}(X_i; X_j, \mathbf{U_i} \ : \ \mathcal{D}).$$

This simply measures the score when adding X_j to the current parents of X_i.

Calculating $\mathrm{M}_{\mathrm{Shield}}$ and $\mathrm{M}_{\mathrm{Score}}$ is more expensive than calculating $\mathrm{M}_{\mathrm{Disc}}$. The calculation of $\mathrm{M}_{\mathrm{Disc}}$ only requires the joint statistics for all pairs X_i and X_j, which in turn require only one pass over the data and can then be cached for later iterations. The other measures require the joint statistics of X_i, X_j, and $\mathbf{U_i}$. In general $\mathbf{U_i}$ changes between iterations, and usually requires a new pass over the data set each iteration. The cost of calculating these new statistics can be reduced by limiting our attention to variables X_j that have large enough mutual information with X_i. Note that this mutual information can be computed using previously collected statistics.

Most of the measures we have developed up to now are tailored for discrete random variables. The information based measure do not translate well to most continuous conditional density families. The $\mathrm{M}_{\mathrm{Score}}$ measure, however, can be applied to any variable, with any conditional probability family, no matter if its parents are discrete, continuous or a hybrid of both types.

4.1.3 Learning with Small Candidate Sets

In this section we examine the problem of finding a constrained Bayesian network attaining a maximal score. We first show why the introduction of candidate sets improves the efficiency of standard heuristic techniques, such as greedy hill-climbing. We then suggest an alternative heuristic "divide and conquer" paradigm that exploits the sparse structure of the constrained graph.

Formally, we attempt to solve the following problem:

Maximal Restricted Bayesian Network (MRBN)
Input:

- A set $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ of instances
- A digraph H of bounded in-degree k
- A decomposable score S

Output: A network $B = \langle G, \Theta \rangle$ so that $G \subseteq H$, that maximizes S with respect to $\mathcal{D}$.

As can be expected, this problem has a hard combinatorial aspect.

Proposition 4.1.2: *MRBN is NP-hard.*

This follows from a slight modification of the NP-hardness of finding an optimal unconstrained Bayesian network (Chickering, 1996).

Standard Heuristics

Though MRBN is NP-hard, even standard heuristics are computationally more efficient and give a better approximation compared to the unconstrained problem. This is due to the fact that the search space is substantially smaller, as is the complexity of each iteration, and the number of counts needed.

The search space of possible Bayesian networks is extremely large. By searching in a smaller space, we can hope to have a better chance of finding a high-scoring network. Although the search space size for MRBN remains exponential, it is tiny in comparison to the space of all Bayesian networks on the same domain. To see this, note that even if we restrict the search to Bayesian networks with at most k parents, there are $O(\binom{n}{k})$ possible parent sets for each variable. On the other hand, in MRBN, we have only $O(2^k)$ possible parent sets for each variable. (Of course, the acyclicity constraints disallow many of these networks, but it does not change the order of magnitude in the size of the sets).

Examining the time complexity for each iteration in heuristic searches also points in favor of MRBN. In greedy hill climbing we initially compute a score for all possible edges, a total of $O(n^2)$ computations. In each subsequent iteration, we only need to compute a score for the variable whose family has changed in the previous iteration, meaning $O(n)$ new calculations. In MRBN we begin with $O(kn)$ initial calculations, after which each iteration only requires $O(k)$ calculations.

A large fraction of the learning time involves collecting the sufficient statistics from the data. Here again, restricting to candidate sets saves time. When k is reasonably small, we can compute the statistics for $\{X_i\} \cup C_i$ in one pass over the input. All the statistics we need for evaluating subsets of C_i as parents of X_i can then be computed by marginalization from these counts. Thus, we can dramatically reduce the number of statistics collected from the data.

Divide and Conquer Heuristics

In this section we describe algorithms that utilize the combinatorial properties of the candidate graph H in order to efficiently find the maximal scoring network, given the constraints. To simplify the following discussion, we abstract the details of the Bayesian network learning problem and focus on the underlying combinatorial problem. This problem is specified as follows:

Input: A digraph $H \equiv \{X_j \longrightarrow X_i : X_j \in C_i\}$, and a set of weights $w(X_i, \mathbf{Y})$ for each X_i and $\mathbf{Y} \subseteq C_i$

Output: An acyclic subgraph $G \subseteq H$ that maximizes $W_H[G] \equiv \sum_i w(X_i, \mathbf{U}_i^G)$

One of the most effective paradigms for designing algorithms is "Divide and Conquer". The main idea of this approach is to take a complex problem and decompose it into simpler problems that can be solved separately. In this particular problem, the global constraint we need to satisfy is acyclicity. Otherwise, we would have selected, for each variable X_i, the parents that attain maximal weight. Thus, we want to decompose the problem into components, so that we can efficiently combine their maximal solutions. We use standard graph decomposition methods to decompose H. Once we have such a decomposition, we can find acyclic solutions in each component and combine them into a global solution.

Strongly Connected Components: (SCC)

The simplest decomposition of this form is one that disallows cycles between components, i.e, *strongly connected components*. A subset of vertices $\boldsymbol{A}$ is *strongly connected* if for each $X, Y \in \boldsymbol{A}$, H contains a directed path from X to Y and a directed path from Y to X. The set $\boldsymbol{A}$ is *maximal* if there is no strongly connected superset of $\boldsymbol{A}$. It is clear that two maximal strongly connected components must be disjoint, and there cannot be a cycle that involves vertices in both of them (for otherwise their union would be a strongly connected component). Thus, we can partition the vertices in H into maximal strongly connected components. Every cycle in H will be contained within a single component. Thus, once we ensure acyclicity "locally" within each component, we get an acyclic solution over all the variables. This means we can search for a maximum on each component independently.

To formalize this idea, we begin with some definitions. Let $\boldsymbol{A}_1, \ldots \boldsymbol{A}_m$ be a partition of $\{X_1, \ldots, X_n\}$. We define the following subgraphs: $H_{X_i} = \{Y \rightarrow X_i | Y \in C_i\}$, $H_j = \bigcup_{X_i \in \boldsymbol{A}_j} H_{X_i}$. For $G \subset H_j$, let $W_{\boldsymbol{A}_j}[G] = \sum_{X_i \in \boldsymbol{A}_j} w(X_i, \mathbf{U}_i^G)$.

foreach *possible order* σ *on* S **do**
  for $i = 1, 2$ **do**
    find $G_{i,\sigma} \subset H_i$, that maximizes $W_{H_i}[]$ among graphs that respect σ ;
  end
  $G_\sigma \leftarrow G_{1,\sigma} \cup G_{2,\sigma}$;
end
Return $G = \arg\max_{G_\sigma} W[G_\sigma]$;

Algorithm 4: Outline of using a separator to efficiently solve MRBN

Proposition 4.1.3: *For* $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ *strongly connected components of* H*, if for each* j*,* $G_j \subset H_j$ *is the acyclic graph that maximizes* $W_{A_j}[G]$ *then*

- *The graph* $G = \cup_j G_j$ *is acyclic.*
- G *maximizes* $W_H[G]$.

Decomposing H into strongly connected components takes linear time (see, e.g., Cormen et al., 1990), therefore we can apply this decomposition, and search for the maxima on each component separately. However, when the graph contains large connected components, we still face a hard combinatorial problem of finding the graphs G_j. For the remainder of this section we will focus on further decomposition of such components.

Separator Decomposition

We now decompose strongly connected graphs, therefore we must consider cycles between the components. However, our goal is to find small "bottlenecks" through which these cycles must go through. We then consider all possible ways of breaking the cycles at these bottlenecks.

Definition 4.1.4: A *separator* of H is a set S of vertices so that:

1. $H \setminus S$ has two components H'_1 and H'_2 with no edges between them. For $j \in \{1, 2\}$ let $H_i = H'_i \cup S$.
2. For each X_i,$\exists j \in \{1, 2\}$ so that $\{X_i \cup C_i\} \subseteq H_j$

∎

Intuitively, S separates the vertices into two sets, such that no potential family crosses the border. For each vertex we therefore search for the maximal choice of parents in only one component (H_1 or H_2). Let A_1 and A_2 be a disjoint partition of all vertices into two sets, so that if $X_i \in A_j$, then $X_i \cup C_i \subset H_j$. The second property of the separator ensures that such a partition exists. Unlike the SCC decomposition, however, this decomposition does not allow us to maximize W for each H_j independently. Suppose that we find two acyclic graphs G_1 and G_2 that maximize $W_{A_1}[]$ and

$W_{A_2}[]$, respectively. If the combined graph $G = G_1 \cup G_2$ is acyclic, then it must maximize $W_H[]$. Unfortunately, G might be cyclic. However, the first property of separators ensures that the source of potential conflicts between G_1 and G_2 involves vertices in the separator S: For $X, Y \in S$, if there is a path from X to Y in G_1 and in addition there is a path from Y to X in G_2, then the combined graph will be cyclic. Conversely, it is also easy to verify, that any cycle in G must involve at least two vertices in S.

This suggests a way of ensuring that the combined graph will be acyclic. If we force some order on the vertices in S, and require both G_1 and G_2 to respect this order, then we disallow cycles. Formally, let σ be a partial order on $\{X_1, \ldots, X_n\}$. We say that a graph G *respects* σ, if whenever there is a directed path $X_j \longrightarrow \ldots \longrightarrow X_i$ in G, then $X_i \not\prec_\sigma X_j$.

Proposition 4.1.5: *Let S be a separator in H and let σ be a complete order on S. Let $G_1 \subset H_1$ and $G_2 \subset H_2$ be two acyclic graphs that respect σ. Then, $G = G_1 \cup G_2$ is acyclic.*

Given S, a small separator in H, this suggests a simple approach described in Algorithm 4. This approach considers $|S|!$ pairs of independent sub-problems. If the cost of finding a solution to each of the sub-problems is smaller than for the whole problem, and if $|S|$ is relatively small, this procedure can be more efficient.

Proposition 4.1.6: *Using the same notation as in the separator-algorithm, if $\forall \sigma$ for $i \in \{1, 2\}$, $G_{i,\sigma}$ maximizes $W_{H_i}[]$ among the graphs that respect σ then:*

- *G_σ maximizes $W_H[]$ among the graphs that respect σ*
- *$G = \arg\max_{G_\sigma} W[G_\sigma]$ maximizes $W_H[]$.*

Proposition 4.1.6 implies that Algorithm 4 returns the optimal solution.

Cluster-Tree Decomposition

In this section we present *cluster trees*, which are representations of the candidate graphs, implying a recursive separator decomposition of H into *clusters*. The idea is similar to those of standard clique-tree algorithms used for Bayesian network inference (e.g., (Jensen, 1996)). We use this representation to discuss a class of graphs for which $W_H[]$ can be found in polynomial time.

Definition 4.1.7: A *Cluster Tree* [2] of H is a pair $(\mathbf{K}, T)$, where $T = (J, F)$ is a tree and $\mathbf{K} = \{K_j | j \in J\}$ is a family of *clusters*, subsets of $\{X_1, \ldots, X_n\}$, one for each vertex of T, so that:

- For each X_i, there exists $j(i) \in J$ such that $\{X_i \cup C_i\} \subseteq K_{j(i)}$.
- For all $i, j, k \in J$, if j is on the path from i to k in T, then $K_i \cap K_k \subset K_j$. This is called the *running intersection property*.

[2] This definition coincides with that of a *Tree Decomposition* for the moralized graph of H (Bodlaender, 1997), as defined in graph theory. We use our modified definition also when referring to *tree width* at the end of this section.

We introduce some notation: Let (i, j) be an edge in T. Then $S_{i,j} = K_i \cap K_j$ is a separator in T, breaking it into two subtrees T_1 and T_2. Define A_j to be the set of vertices assigned (with their parents) to K_j: $A_j = \{X_i | j(i) = j\}$. For $i = 1, 2$, define $A[T_i] = \bigcup_{j \in T_i} A_j$.

Whenever $|S_{i,j}|$ is small, and the two subtrees are not too big, we can use $S_{i,j}$ efficiently as in Algorithm 4. But when one of the subtrees T_1 or T_2 is big, we would like to decompose it further. This can be done in a recursive manner, using the tree structure and the other separators. We now devise a dynamic programming algorithm for computing the optimal graph using the cluster tree separators. First, let us root the cluster tree at an arbitrary $K_0 \in \mathbf{K}$, inducing an order on the tree vertices. Each cluster $K_j \in \mathbf{K}$ is the root of a subtree T_j, spanning away from K_0. S_j is the tree separator, separating T_j from the rest of T ($S_0 = \emptyset$). The *sub-vertices* of K_j are its neighbors in T_j. Figure 4.2 shows a candidate graph H for a domain with seven variables, and a corresponding possible cluster tree. In this tree A and F are assigned to K_0, G and E are assigned to K_1, B,C and D are assigned to K_2. The separator S_1, for example, separates the subtree T_1 (composed of K_1) from the rest of the tree. The sub-vertices of K_0 are K_1 and K_2, both of which have no sub-vertices of their own.

Define for each cluster K_j and each total order σ on S_j the weight $W[T_j, \sigma]$ of the maximal partial solution for the subtree T_j which respects σ

$$W[T_j, \sigma] = \max_{\substack{\text{acyclic } G \subset H[A[T_j]] \\ \text{respecting } \sigma}} W_{A[T_j]}[G]. \tag{4.1}$$

The crux of the algorithm is that finding these weights can be done in a recursive manner, based on previously computed maxima.

Proposition 4.1.8: *For each cluster $K_j \in \mathbf{K}$ and order σ over S_j: Let $K_1, \ldots, K_k$ be the sub-vertices of K_j. Then:*

$$W[T_j, \sigma] = \max_{\sigma'} \Big(\max_{\substack{\textit{acyclic } G \subset H[A_j] \\ \textit{respecting } \sigma'}} W_{A_j}[G] + \sum_{i=1}^{k} W[T_i, \sigma'|_{S_i}] \Big)$$

where σ' ranges on all orders on K_j that are consistent with σ, ,and $\sigma'|_{S_i}$ is the restriction of σ' to an order over S_i.

Proposition 4.1.8 facilitates rapid evaluation of all the tables $W[T_j, \sigma]$ in one phase, working our way from the leaves inwards towards K_0. At the end of this traversal, we have computed the weight of each ordering on all separators adjacent to the root cluster K_0. A second phase then traverses T

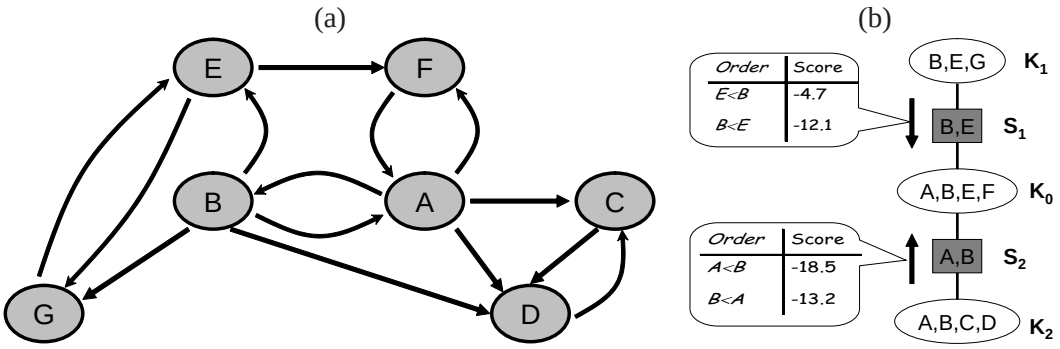


Figure 4.2: Illustration of the Cluster Tree algorithm. (a) The graph H, showing all candidate parents of each variable. (b) A corresponding cluster tree, along with possible weight "messages" $W[T_j, \sigma]$

from the root outwards, in order to back-trace the choices made during the first phase, leading to the maximum total weight $W_H[G]$. Figure 4.2(b) shows the computed weights for T_1 and T_2 at the end of the first phase in our example. The weights computed for S_1, for example, correspond to the subtree consisting of K_1, and so $W[T_1, \sigma]$ here sums the scores over the families of G and E. In this case, the ordering $E < B$ allows the consideration of B as a parent of E, resulting in a higher score (weight).

Examining the complexity of this algorithm, we see that each cluster K_j is visited twice, the first (more expensive) visit requiring $O(|S_j|! \cdot |A_j| \cdot 2^k)$ operations, where k is the size of the candidate sets. Thus, we get the following result:

Theorem 4.1.9: If c is the size of the largest cluster in the cluster tree, and s the size of the largest separator, then finding G that maximizes $W[G]$ can be done in $O(2^k \cdot c \cdot s! \cdot |J|)$.

In summary, the algorithm is *linear* in the size of the cluster tree but worse than exponential in the size of the largest separator in the tree. This, in turn, is of course bounded by the size of the largest cluster, c. This size is called the *width* of the cluster tree decomposition of the moralized graph of H (Bodlaender, 1997). The discussion until now assumed a fixed cluster tree. In practice we also need to select the tree decomposition. Our analysis shows that a good choice would be the tree with the minimal width. This size is called the *tree width* of the original graph. Determining the tree width of a graph, and finding a tree decomposition with a minimal width are both well-known and hard problems that are beyond the scope of this section (for a review of related results, see Bodlaender (1997)). However, we note that if there is a cluster tree of small width, it can be found in polynomial time.

We note that this algorithm is another example of turning an NP-complete graph problem to a polynomial (or even linear) time one, given a tree decomposition with a bounded tree width. Such methods were shown for several classic graph problems. Bodlaender (1993) gives a survey of some of these results, as well as applications of bounded tree width to real life problems.

Cluster-Tree Heuristics

Although the algorithm of the previous section is linear in the number of clusters, it is worse than exponential in the size of the largest cluster. Thus, in many situations we expect it to be hopelessly intractable. Nonetheless, this algorithm provides some intuition on how to decompose the heuristic search for our problem. The key idea is that although after computing a cluster tree, many of the clusters might be large, we can use a mixture of the exact algorithm on small clusters and heuristic searches such as greedy hill climbing on the larger clusters. We now briefly outline the main ideas of this approach.

When K_j is sufficiently small, we can efficiently store the tables $W[T_j, \sigma]$ used by the exact cluster tree algorithm. However, if the clusters are large, then we cannot do the maximization of Proposition 4.1.8. Instead, we perform a heuristic search, such as greedy hill-climbing, over the space of parents for vertices in A_j to find a partial network that is consistent with the ordering induced by the current assignment.

By proceeding in this manner, we approximate the exact algorithm. This approximation examines a series of small search spaces, that are presumably easier to deal with than the original search space. This approach can be easily extended to deal with cluster trees in which only some of the separators are small.

4.1.4 Experimental Evaluation

In this section we illustrate the effectiveness of the sparse candidate algorithm. We examine both a synthetic example and a real-life dataset. Our current experiments are designed to evaluate the effectiveness of the general scheme and to show the utility of various measures for selecting candidates in the **Restrict** phase. In the experiments described here we use greedy hill-climbing for the **Maximize** phase.

The basic heuristic search procedure we use is a greedy hill-climbing that considers local moves in the form of edge addition, edge deletion, and edge reversal. At each iteration, the procedure examines the change in the score for each possible move, and applies the one that leads to the biggest improvement. These iterations are repeated until convergence. To escape local maxima, the procedure is augmented with a simple version of TABU search. It keeps a list of the N last candidates seen, and instead of applying the best local change, it applies the best local change that results in a structure not on the list. Note that because of the TABU list, the best allowed change might actually reduce the score of the current candidate. We terminate the procedure after some fixed number of changes failed to result in an improvement over the best score seen so far. After termination, the procedure returns the best scoring structure it encountered.

In the reported experiments we use this hill-climbing procedure both for the Maximize phase of the sparse candidate algorithm, and as a search procedure by itself. In the former case, the only local changes that are considered are those allowed by the current choice of candidates. In the latter case, the procedure considers all possible local changes. This latter case serves as a reference point

Method	Iter	Time	Score	KL	Stats
Greedy		40	-15.35	0.0499	2656
Disc 5	1	14	-18.41	3.0608	908
	2	19	-16.71	1.3634	1063
	3	23	-16.21	0.8704	1183
Disc 10	1	20	-15.53	0.2398	1235
	2	26	-15.43	0.1481	1512
	3	32	-15.43	0.1481	1733
Shld 5	1	14	-17.50	2.1675	915
	2	29	-17.25	1.8905	1728
	3	36	-16.92	1.5632	1907
Shld 10	1	20	-15.86	0.5357	1244
	2	35	-15.50	0.1989	1968
	3	41	-15.50	0.1974	2109
Score 5	1	12	-15.94	0.6756	893
	2	27	-15.34	0.0550	1838
	3	34	-15.33	0.0479	2206
Score 10	1	17	-15.54	0.2559	1169
	2	30	-15.31	0.0352	1917
	3	34	-15.31	0.0352	2058

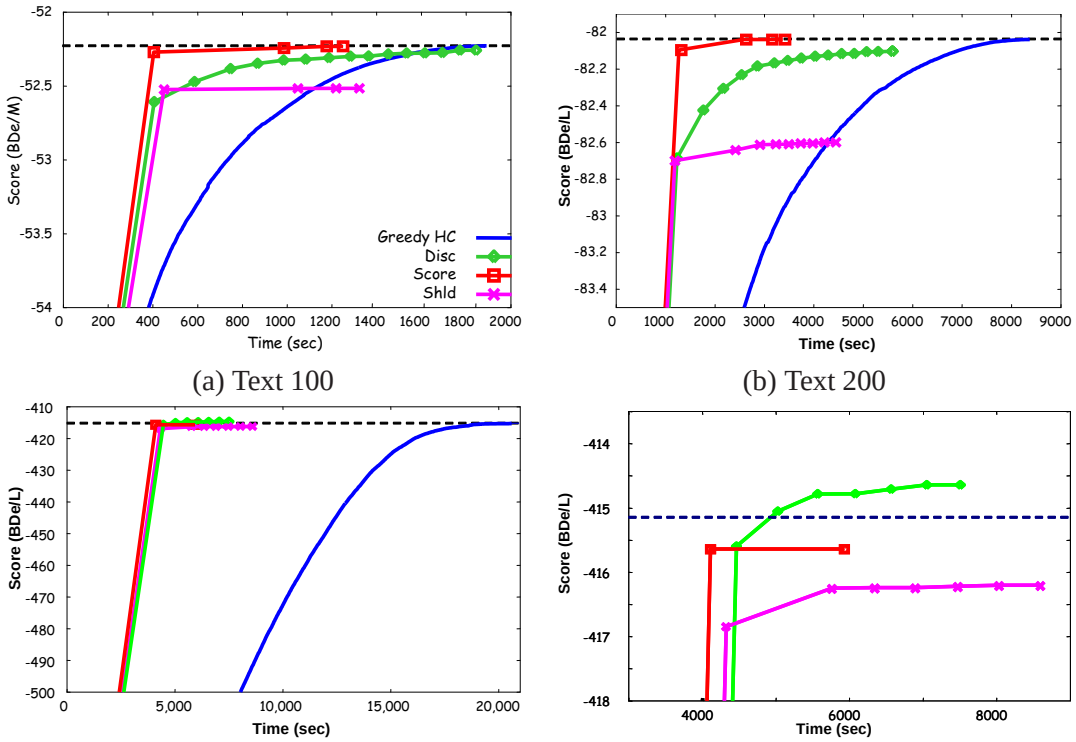
Table 4.1: Summary of results on synthetic data from the alarm domain. These results report the quality of the network, measured both in terms of the score (BDe score divided by number of instances), and KL divergence to the generating distribution. The other columns measure performance both in terms of execution time (seconds) and the number of statistics collected from the data. The methods reported are **Disc** – discrepancy measure, **Shld** – shielding measure, and **Score** – score based measure.

against which we compare our results.

To compare these search procedures we need to measure both their performance in the task at hand, and their computational cost. Performance evaluation is based on the score assigned to the network found by each algorithm. In addition, for synthetic data , we can also measure the true error with respect to the generating distribution. This allows us to assess the significance of the differences between the scores. Evaluating the computational cost is more complicated. The simplest approach is to measure running time. We report running times on an unloaded Pentium II 300MHz machines running Linux. These running times, however, depend on various coding issues in our implementation. We attempted to avoid introducing bias within our code for either procedure, by using the same basic library for evaluating the score of candidates and for computing and caching of sufficient statistics. Moreover, the actual search is carried by the same code for greedy-hill climbing procedure.

As additional indication of computational cost, we also measured the number of sufficient statistics computed from the data. In massive datasets these computations can be the most significant portion of the running time. To minimize the number of passes over the data we use a cache that allows us to use previously computed statistics, and to marginalize statistics to get the statistics of subsets. We report the number of actual statistics that were computed from the data.

Finally, in all of our experiments we used the BDe score of (Heckerman et al., 1995a) with a uniform prior with equivalent sample size of ten. This choice is a fairly uninformed prior that does not code initial bias toward the correct network. The strength of the equivalent sample size was set



(a) Text 100 (b) Text 200

(c) Cell Cycle 800 (d) Cell Cycle 800 (cloesup)

Figure 4.3: Performance of the different algorithms on the text and biological domains. Each graph plots the score (y-axis) vs. running time (x-axis). The reported methods vary in terms of the candidate selection measure: (**Disc** – discrepancy measure, **Shld** – shielding measure, **Score** – score based measure); **Greedy HC** – greedy hill-climbing. Candidate set sizes used in the experiments: text – $k = 15$; cell cycle – $k = 20$. The points on each curve for the sparse candidate algorithm are the end result of an iteration. The dashed line marks the maximal score obtained by the greedy method.

prior to the experiments and was not tuned.

In the first set of experiments we used a sample of 10000 instances from the "alarm" network (Beinlich et al., 1989). This network has been used for studies of structure learning in various papers, and is treated as a common benchmark in the field. This network contains 37 variables, of which 13 have 2 values, 22 have 3 values, and 2 have 4 values. Although this data set is not particularly massive, it does allow us to observe the behavior of our search procedure.

The results for this small data set are reported in Table 4.1. In this table we measure both the score of the networks found and their error with respect to generating distributions. The results on this toy domain show that our algorithm, in particular with the Score selection heuristic, finds networks with comparable score to the one found by greedy hill climbing. Although the timing results for this small scale experiments are not significant, we do see that the sparse candidate algorithm usually requires fewer statistics records. Finally, we note that the first iteration of the algorithm finds reasonably high scoring networks. Nonetheless, subsequent iterations improve the score. Thus, the

re-estimation of candidate sets based on our score does lead to important improvements.

To test our learning algorithms on more challenging domains we examined data from text and from a biological domain. In the text domain, we used a data set that contains messages from 20 newsgroups (approximately 1000 from each) (Jochims, 1997). We represent each message as a vector containing one attribute for the newsgroup and attributes for each word in the vocabulary. We constructed data sets with different numbers of attributes by focusing on subsets of the vocabulary. We did this by removing common stop words, and then sorting words based on their frequency in the whole data set. The data sets included the group designator and the 99 (*text 100* set) or 199 (*text 200* set) most common words. We trained on 10,000 messages that were randomly selected from the total data set. In the biological domain, we devised another synthetic dataset, which originates in real biological data. We used gene expression data from (Spellman et al., 1998). The data describes expression levels of 800 cell-cycle regulated genes, over 76 experiments (we return to this data in Chapter 5). We learned a network from this dataset, and then sampled 5000 instances from the learned network. We then used this synthetic dataset.

The results of these experiments are reported in Figure 4.3. As we can see, in the case of 100 attributes, by using the Score selection method with candidate sets of sizes 15, we can learn networks that are reasonably close to the one found by greedy hill-climbing in about half the running time. When we have 200 attributes, the speedup is larger than 3, and in the cell cycle domain (having 800 variables), the speedup is up to 3.5 fold (the Score selection method finishes in under 6000 seconds). We expect that as we consider data sets with larger number of attributes, this speedup ratio will grow. The ratios between the number of statistics collected during the runs are similar to the ratios of running time (data not shown). After the first iteration, where the initial $O(N^2)$ statistics are collected, each iteration adds only a modest number of new statistics, since we only calculate the measure for pairs of variables that initially had a significant mutual information. We also note that the discrepancy measure has a slower learning curve than the score measure. This might be due to better candidate selection of the score measure, which reflects the most in the first iterations, where most changes take place.

4.1.5 Discussion

The "Sparse Candidate" algorithm offers a simple heuristic for improving search efficiency. By restricting our search to examine only a small number of candidate parents for each variable, we can find high-scoring networks efficiently. Furthermore, we showed that we can improve the choice of the candidates by taking into account the network we learned, thus getting higher scoring networks. We demonstrated both of these effects in our experimental section. These results show that our procedure can lead to dramatic reduction in the learning time with a small loss of quality.

Second, we showed that by restricting each variable to a small group of candidate parents, we can sometimes get theoretical guarantees on the complexity of the learning algorithm. This result is of theoretical interest: to the best of our knowledge, this is the first non-trivial case for which one can find a polynomial time learning algorithm for networks with in-degree greater than

one. This theoretical argument might also have practical ramifications. As we showed, even if the exact polynomial algorithm is too expensive, we can use it as a guide for finding good approximate solutions.

In addition to the experimental results we describe here, the "Sparse Candidate" algorithm was already applied in several other projects. In (Boyen et al., 1999), the sparse candidate method is combined with the structural EM procedure (Friedman, 1997) for learning structure from incomplete data. In that setup, the cost of finding statistics is much higher, since instead of counting number of instances, we have to perform inference for each of the instances. As a consequence the reduction in the number of requested statistics (as shown in our results) leads to significant saving in run time. Similar cost issues occur in (Getoor et al., 1999), where a variant of the algorithm is used for learning probabilistic models from relational databases. Finally, as we shall see in Chapter 5, this procedure is a crucial component in our analysis of real-life *gene expression data* that contains thousands of attributes.

There are several directions for future research. When coming to learn a network over several thousand variables, the cost of the initial **Restrict** step of the algorithm is prohibitive (since it is quadratic in the number of variables). Developing heuristic methods for finding good candidates would enable learning in such domains. Once we learn a network based on these candidates, we can use it to help focus on other variables that should be examined in the next **Restrict** step. Another direction of interest is the combination of our methods with other recent ideas for efficient learning from large datasets, such as (Moore and Lee, 1997). Finally, the usage of the method on continuous variable networks is currently limited only to the score measure. At the end of this chapter we point at additional measures which can be tailored to specific conditional densities, and can also speed up significantly the initial quadratic step.

4.2 Learning in Continuous Variable Networks: The "Ideal Parent" Method

The "Sparse Candidate" algorithm presents a heuristic method for choosing candidate parents based on conditional pairwise scores. In each iteration, however, we still have to perform greedy structure search, though in a much more restricted space. Greedy search involves computing a family score for each suggested edge change. This score computation, as we saw, can be costly in many cases. Can we avoid the necessity to compute a full score? Can we estimate the score using a cheaper computation?

In this section we focus on learning continuous variable networks, which are crucial for a wide range of real-life applications. When we learn *linear Gaussian* networks (Geiger and Heckerman, 1994), we can use sufficient statistics to summarize the data, and a closed form equation to evaluate the score of candidate families. In general, however, we are also interested in non-linear interactions. These usually require non-linear parameter optimization to evaluate the score of a candidate family.

This severely limits the applicability of standard heuristic structure search procedures to rich non-linear models.

We present a general method for speeding search algorithms for structure learning in continuous variable networks. Our method can be applied to many forms of a unimodal parametric conditional distribution, including the linear Gaussian model as well as many non-linear models. The ideas are inspired from the notion of *residues* in regression (McCullagh and Nelder, 1989), and involve the notion of "ideal parents". For each variable, we construct an *ideal parent profile* of a new hypothetical parent that would lead to the best possible prediction of the variable. We then use this profile to efficiently select potential candidate parents that have a similar profile of values. Using basic principles, we derive a similarity measure that can be computed efficiently and that approximates the improvement in score that would result from the addition of a candidate hidden parent. This provides us with a fast method for scanning many potential parents and focus more careful evaluation (scoring) to a smaller number of promising candidates.

The ideal parent profiles we construct during search also provide new leverage on the problem of introducing new hidden variables during structure learning. Basically, if the ideal parent profiles of several variables are sufficiently similar, and are not similar to the profile of an existing variable in the current model, we can consider adding a new hidden variable that serves as a parent of all these variables. The ideal profile allows us to estimate the impact this new variable will have on the score, and suggest the values it takes in each instance. And so, the method provides a guided approach for introducing new variables during search and allows to contrast them with alternative search steps in a computationally efficient manner.

4.2.1 The "Ideal Parent" Concept

Structure learning typically involves a traversal of a super-exponential search space. Even when using a greedy local search procedure, structure learning is extremely time consuming. This problem is even more acute when considering parameterizations that require using non-linear optimization for estimating the maximum likelihood parameters. Our goal in this section is to efficiently select the approximately best candidate parent during the search procedure for Bayesian networks with continuous variables, and to facilitate the addition of new and effective hidden variables into the network structure. We start by characterizing the notion of an ideal parent that will enable us to do so.

The basic idea is straightforward — for a given variable, we want to construct a hypothetical "ideal parent" that would best predict the variable. We will then compare existing candidate parents to this imaginary one and find the ones that are most similar. Throughout this discussion, we focus on a single variable X that in the current network has parents $\mathbf{U}$. To make our discussion concrete, we focus on networks where we represent X as a function of its parents $\mathbf{U} = \{U_1, \ldots, U_k\}$ with CPDs that have the following general form:

$$X = g(\alpha_1 \mathbf{u}_1, \ldots, \alpha_k \mathbf{u}_k : \theta) + \epsilon \tag{4.2}$$

where g is a function that integrates the contributions of the parents with additional parameters θ, α_i that are scale parameters applied to each of the parents, and ϵ that is a noise random variable with zero mean. In here, we assume that ϵ is Gaussian with variance σ^2. In Section 4.2.4 we deal with more general forms of noise modeling.

When the function g is the sum of its arguments, this CPD is the standard linear Gaussian CPD. However, we can also consider non-linear choices of g. For example,

$$g(y_1, \ldots, y_k : \theta) \equiv \theta_1 \frac{1}{1 + e^{-\sum_i y_i}} + \theta_0 \tag{4.3}$$

is a sigmoid function where the response of X to its parents' values is saturated when the sum is far from 0. Both of these examples of CPDs are instances of *generalized linear models* (GLMs) (McCullagh and Nelder, 1989). This class of CPDs uses a function g that is applied to the sum of its arguments, called the *link function* in the GLM literature. However, we can also consider more complex functions, as long as they are well defined for any desired number of parents. For example, in Chapter 6 models based on chemical reactions are considered, where the function g does not have a GLM form. An example of a two variable function of this type is:

$$g(y_1, y_2 : \theta) = \theta \frac{y_1 y_2}{(1 + y_1)(1 + y_2)}$$

With this definition for a CPD, we now can define the ideal parent for X.

Definition 4.2.1: Given a dataset $\mathcal{D}$, and a mode function g for the CPD of X given its parents $\mathbf{U}$ with parameters θ and α, the *ideal parent* Y of X is such that for each instance m,

$$x[m] = g(\alpha_1 u_1[m], \ldots, \alpha_k u_k[m], y[m] : \theta) \tag{4.4}$$

∎

Under mild conditions, the *ideal parent profile* (i.e., value of Y in each instance) can be computed for almost any unimodal parametric conditional distribution. The only requirement from g is that it should be invertible w.r.t. each one of the parents.[3]

The resulting profile for the ideal parent Y is the optimal set of values for the $k+1$'th parent, in the sense that it would maximize the likelihood of the child variable X. Intuitively, if we can efficiently find a candidate parent that is similar to the optimal parent, we can improve the model by adding an edge from this parent to X.

We now develop the computation of the similarity measure between the ideal profile and a candidate parent for the case of a linear Gaussian distribution. In Section 4.2.2 we describe how to use this measure to perform the actual search.

[3] In Definition 4.2.1 we implicitly assume $x[m]$ lies in the image of g. If this is not the case, we can substitute $x[m]$ with $x_g[m]$, the point in g's image closest to $x[m]$. This guarantees the prediction's mode for the current set of parents and parameters is as close as possible to X.

Linear Gaussian

Let X be a variable in the network with a set of parents $\mathbf{U}$, and a *linear Gaussian* conditional distribution. In this case, g in Eq. (4.2) takes the form

$$g(\alpha_1 \mathbf{u}_1, \ldots, \alpha_k \mathbf{u}_k : \theta) \equiv \sum_i \alpha_i \mathbf{u}_i + \theta_0$$

In using the BIC score (see Section 2.4.3), whenever we consider a change in the structure, such as adding Z as a new parent of X whose current parents are $\mathbf{U}$, we need to compute the change in likelihood

$$\Delta_{X|\mathbf{U}}(Z) = \max_{\theta'} \ell_X(\mathcal{D} : \mathbf{U} \cup \{Z\}, \theta') - \ell_X(\mathcal{D} : \mathbf{U}, \theta)$$

where θ is the current maximum likelihood parameters for the family. The change in the BIC score is this difference combined with the change in penalty terms. To evaluate this difference, we need to compute the maximum likelihood parameters of X given the new choice of parents. In the "ideal parent" approach we use a fast method for choosing the promising candidates for additional parent, and then compute the BIC score only for these candidates. This reduces the time complexity of the scoring step.

To choose promising candidate parents to add, we start by computing the ideal parent Y for X given its current set of parents. This is done by inverting the linear link function g with respect to this additional parent Y (note that we can assume, without loss of generality, that the scale parameter of Y is 1). This results in

$$y[m] = x[m] - \sum_j \alpha_j u_j[m] - \theta_0 \tag{4.5}$$

We can summarize this in vector notation, by using $\vec{x} = \langle x[1], \ldots, x[M] \rangle$, and so we get

$$\vec{y} = \vec{x} - \mathcal{U}\vec{\alpha}$$

where $\mathcal{U}$ is the matrix of the parent values on all instances, and $\vec{\alpha}$ is the vector of scale parameters.

Having computed the *ideal parent profile*, we now want to efficiently evaluate its similarity to profiles of candidate parents. Intuitively, we want the similarity measure to reflect the likelihood gain by adding Z as a parent of X. Ideally, we want to evaluate $\Delta_{X|\mathbf{U}}(Z)$ for each candidate parent Z. However, instead of reestimating all the parameters of the CPD after adding Z as a parent, we approximate this difference by only fitting the scaling factor associated with the new parent and freezing all other parameters of the CPD.

Proposition 4.2.2 *Suppose that X has parents $\mathbf{U}$ with a set $\vec{\alpha}$ of scaling factors. Let Y be the ideal parent as described above, and Z be some candidate parent. Then the change in likelihood of X in the data, when adding Z as a parent of X, while freezing all parameters except the scaling factor*

of Z, is

$$\begin{aligned} C_1(\vec{y},\vec{z}) &\equiv \max_{\alpha_Z} \ell_X(\mathcal{D}:\mathbf{U}\cup\{Z\},\theta\cup\{\alpha_Z\}) - \ell_X(\mathcal{D}:\mathbf{U},\theta) \\ &= \frac{1}{2\sigma^2}\frac{(\vec{y}\cdot\vec{z})^2}{\vec{z}\cdot\vec{z}} \end{aligned}$$

Proof: We can write the difference in log-likelihood after and before the addition of the parent p as (with the α's of the other parents held fixed):

$$\begin{aligned} \log P(\vec{x}|\mathcal{U},\vec{z},\alpha_z) - \log P(\vec{x}|\mathcal{U}) &= \frac{1}{2\sigma^2}\vec{y}\cdot\vec{y} - \frac{1}{2\sigma^2}(\vec{y}\cdot\vec{y} - 2\alpha_z\vec{z}\cdot\vec{y} + \alpha_z^2\vec{z}\cdot\vec{z}) \\ &= \frac{1}{2\sigma^2}\left[2\alpha_z\vec{z}\cdot\vec{y} + \alpha_z^2\vec{z}\cdot\vec{z}\right] \end{aligned}$$

The first equality follows from the substitution of $\vec{y}$ as defined in Eq. (4.5). After substituting α_z with its *maximum likelihood* estimate which is $\frac{\vec{z}\cdot\vec{y}}{\vec{z}\cdot\vec{z}}$, we get the desired result. ∎

Note that by definition of Y, the maximum likelihood estimator of σ^2 is $\frac{1}{M}\vec{y}\cdot\vec{y}$.[4] If we substitute it in the definition of C_1, we get an intuitive result:

$$C_1(\vec{y},\vec{z}) = \frac{M}{2}\frac{(\vec{y}\cdot\vec{z})^2}{(\vec{z}\cdot\vec{z})(\vec{z}\cdot\vec{z})} = \frac{M}{2}\cos^2\phi_{\vec{y},\vec{z}}$$

where $\phi_{\vec{y},\vec{z}}$ is the angle between $\vec{y}$ and $\vec{z}$. The smaller the angle between these two vectors, the higher the similarity, regardless of their norms: It can easily be shown that $\vec{z} = c\vec{y}$ (for any constant c) maximizes this similarity measure.

Note that $C_1(\vec{y},\vec{z})$ is a *lower bound* on $\Delta_{X|\mathbf{U}}(Z)$, the improvement on the log-likelihood by adding Z as a parent of X. When we add the parent we optimize all the parameters, and so we expect to attain a likelihood as high, or higher, than the one we attain by freezing some of the parameters. This is illustrated in Figure 4.4(a) that plots C_1 vs. the true likelihood improvement for several thousand edge modifications.

We can get a better lower bound by optimizing additional parameters. In particular, after adding a new parent, the errors in predictions change, and so we can readjust the variance term. As it turns out, we can perform this readjustment in closed form.

Proposition 4.2.3 *Suppose that X has parents $\mathbf{U}$ with a set $\vec{\alpha}$ of scaling factors. Let Y be the ideal parent as described above, and Z be some candidate parent. Then the change in likelihood of X in the data, when adding Z as a parent of X, while freezing all other parameters except the variance*

[4] We choose to use σ^2 explicitly in the definition of C_1 for compatibility with the developments below.

of X, is

$$\begin{aligned} C_2(\vec{y},\vec{z}) &\equiv \max_{\alpha_Z,\sigma} \ell_X(\mathcal{D}:\mathbf{U}\cup\{Z\},\theta\cup\{\alpha_Z\}) - \ell_X(\mathcal{D}:\mathbf{U},\theta) \\ &= -\frac{M}{2}\log\,\sin^2\phi_{\vec{y},\vec{z}} \end{aligned}$$

Proof: The maximum likelihood estimator of the new variance parameter is:

$$\begin{aligned} \sigma_z^2 &= \frac{1}{M}\sum_m (x[m]-g(\mathbf{u}[m],\alpha_z z[m])^2 \\ &= \frac{1}{M}(\alpha_z\vec{z}-\vec{y})\cdot(\alpha_z\vec{z}-\vec{y}) \\ &= \frac{1}{M}\left(\vec{y}\cdot\vec{y}-\frac{(\vec{z}\cdot\vec{y})^2}{\vec{z}\cdot\vec{z}}\right) \end{aligned}$$

The second equality comes from plugging in the definition of $\vec{y}$, and the third from incorporating the maximum likelihood estimate of α_z. The log likelihood difference is therefore:

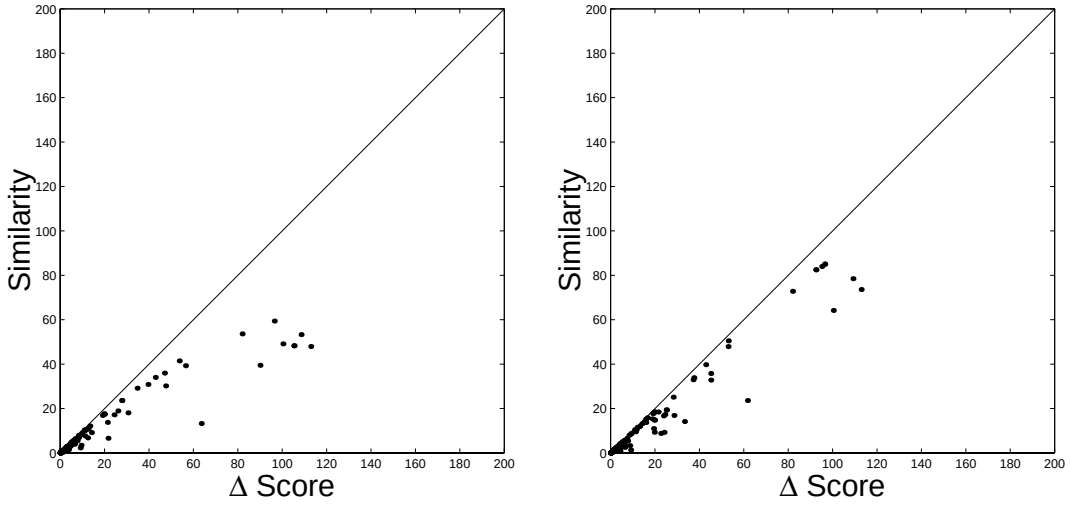
$$\begin{aligned} \log P(\vec{x}|\mathcal{U},\vec{z};\alpha_z,\sigma_z) - \log P(\vec{x}|\mathcal{U};\sigma) &= -M\log(\sigma_z) + M\log(\sigma) - \frac{M}{2} + \frac{M}{2} \\ &= -\frac{M}{2}\log\left(\vec{y}\cdot\vec{y}-\frac{(\vec{z}\cdot\vec{y})^2}{\vec{z}\cdot\vec{z}}\right) + \frac{M}{2}\log(\vec{z}\cdot\vec{z}) \\ &= \frac{M}{2}\log\frac{1}{1-\frac{(\vec{z}\cdot\vec{y})^2}{(\vec{z}\cdot\vec{z})(\vec{y}\cdot\vec{y})}} \end{aligned}$$

The last two terms after the first equality are the sum of squares terms after we have substituted both σ and σ_z with their maximum likelihood estimates. In the second equality we have plugged those estimators in the first two terms. The result follows immediately from the last equality.∎

We can easily show that

$$C_1(\vec{y},\vec{z}) \leq C_2(\vec{y},\vec{z}) \leq \Delta_{X|\mathbf{U}}(Z)$$

due to the set of parameters we optimize in each quantity. As seen in Figure 4.4 C_2 is clearly a tighter bound than C_1, particularly for promising candidates. It is important to note that both C_1 and C_2 are monotonic functions of $\frac{(\vec{y}\cdot\vec{z})^2}{\vec{z}\cdot\vec{z}}$, and so they consistently rank candidate parents of the same variable. When we compare changes that involve different ideal parents, such as adding a parent to X_1 compared to adding a parent to X_2, the ranking by these two measures might differ.



(a) C_1 score (b) C_2 score

Figure 4.4: Demonstration of the C_1 (a) and C_2 (b) similarity measures for linear Gaussian CPDs. The similarity measures (y-axis) are shown against the change in log likelihood resulting from the corresponding edge modifications (x-axis). Points shown correspond to several thousand edge modifications in a run of the ideal parent method on real-life yeast gene expressions data.

4.2.2 Ideal Parents in Search

The technical developments of the previous section show that we can approximate the score of candidate parents for X by comparing them to the ideal parent Y. Is this approximate evaluation useful? We now discuss two ways of using this approximation during structure search.

Guiding Heuristic Search

When performing a local heuristic search, at each iteration we have a current candidate structure and we consider some operations on that structure. These operations might include edge addition, edge replacement, edge reversal and edge deletion. We can readily use the ideal profiles and similarity measures developed to speed up two of these: edge addition and edge replacement. These two modification form the bulk of edge changes considered by a typical search algorithm, since edge deletions and reversals can only be applied to the relatively small number of existing edges.

When considering adding an edge $Z \rightarrow X$, we use the ideal parent profile for X and compute its similarity to Z. We repeat this for every other candidate parent for X. We then compute the full score only for the K most similar candidates, and insert them (and the associated change in score) to a queue of potential operations. In a similar way, we can utilize the ideal parent profile for considering edge replacement for X. Suppose that Z is a parent of X. We can define the ideal profile for replacing Z while freezing all other parameters of the CPD of X. The difference here is that for each current parent of X we compute a separate ideal profile - one corresponding to replacement of that parent with a new one. We then use the same policy as above for examining

replacement of each one of the parents.

We note that we can tradeoff between the accuracy of our move evaluations and the speed of the search, by changing K, the number of candidate changes per family for which we compute a full score. Using $K = 1$, we only score the best candidate according to the ideal parent method ranking, thus achieving the largest speedup. Since our ranking only approximates the true score difference, this strategy might miss good moves. Using higher values of K brings us closer to the standard search both in terms of move selection quality but also in terms of computation time.

In the experiments in Section 4.2.5, we integrated the changes described above into a greedy hill climbing heuristic search procedure. This procedure also examines moves that remove an edge, which we evaluate in the standard way. The greedy hill climbing procedure applies the best available move at each iteration (among these that were chosen for full evaluation).

Adding New Hidden Variables

One of the hardest challenges in learning graphical models is dealing with *hidden* variables. Such variables pose several problems, the hardest of which is detecting when and how should one add a new hidden variable into the network structure. When we learn networks in a domain with a large number of variables, and each hidden variable influences a relatively small subset of these variables, this becomes a major issue (see, e.g., Elidan and Friedman, 2003; Elidan et al., 2001; Martin and VanLehn, 1995; Zhang, 2002).

The ideal parent profiles provide a straightforward way to find when and where to add hidden variables to the domain. The intuition is fairly simple: if the ideal parents of several variables are similar to each other, then we know that a similar input is predictive of all of them. Moreover, if we do not find a variable in the network that is close to these ideal parents, then we can consider adding a new hidden variable that will serve as their combined input, and, in addition, have an informed initial estimate of its profile.

When introducing a new hidden variable Z, it can be connected as a parent to any subset, or cluster, of the variables. Ideally, we would like it to be beneficial for several children at once. The difference in log-likelihood due to adding a new parent with profile $\vec{z}$ is the difference between the log-likelihood of families it is involved in:

$$\Delta_{X_1,\ldots,X_L}(Z) = \sum_{i}^{L} \Delta_{X_i|\mathbf{U}_i}(Z)$$

where we assume, without loss of generality, that the members of the cluster are $X_1, \ldots, X_L$. To score the network with Z as a new hidden variable, we also need to deal with the difference in the complexity penalty term, and the likelihood of Z as a root variable. These terms, however, can be readily evaluated. The difficult term is finding the profile $\vec{z}$ that maximizes $\Delta_{X_1,\ldots,X_L}(Z)$.

Using the ideal parent approximation, we can lower bound this improvement:

$$\Delta_{X_1,\ldots,X_L}(Z) \geq \sum_i^L C_1(\vec{y}_i, \vec{z}) \equiv \sum_i \frac{1}{2\sigma_i^2} \frac{(\vec{z} \cdot \vec{y}_i)^2}{\vec{z} \cdot \vec{z}} \tag{4.6}$$

and so we want to find the profile $\vec{z}$ that maximizes this bound. We will then use this optimized bound as our cluster score, and search for the best cluster for connecting Z. Rewriting the last expression from Eq. (4.6) in matrix notation, we want to find $\vec{z^*}$ that maximizes

$$\vec{z^*} = \arg\max_{\vec{z}} \frac{\vec{z}^T \mathcal{Y} \mathcal{Y}^T \vec{z}}{\vec{z}^T \vec{z}} \tag{4.7}$$

where $\mathcal{Y}$ is the matrix whose columns are y_i/σ_i.

It is easy to see that $\vec{z^*}$ must lie in the span of $\mathcal{Y}$: any component orthogonal to this span increases the denominator of Eq. (4.7), but leaves the numerator unchanged, and therefore does not obtain a maximum. We can therefore express the solution as:

$$\vec{z^*} = \sum_i \lambda_i y_i/\sigma_i = \mathcal{Y}\vec{\lambda} \tag{4.8}$$

Furthermore, the objective in Eq. (4.7) is known as the *Rayleigh quotient* of the matrix $\mathcal{Y}\mathcal{Y}^T$ and the vector $\vec{z}$, and its optimum is achieved when $\vec{z}$ equals the eigenvector of $\mathcal{Y}\mathcal{Y}^T$ corresponding to the largest eigenvalue (Wilkinson, 1965). We can express this eigenvector problem as follows:

$$\mathcal{Y}\mathcal{Y}^T \vec{z^*} = \gamma \vec{z^*}$$

Plugging in Eq. (4.8) we get:

$$\mathcal{Y}\mathcal{Y}^T \mathcal{Y}\vec{\lambda} = \gamma \mathcal{Y}\vec{\lambda}$$

Multiplying both sides by $\mathcal{Y}^T$ and defining $A \equiv \mathcal{Y}^T\mathcal{Y}$, we can write:

$$AA\vec{\lambda} = \gamma A\vec{\lambda}$$

We can now either solve this reduced generalized eigenvalue problem directly, or, if A is non-singular , we can multiply both sides by A^{-1} and end up with a simple eigenvalue problem:

$$A\vec{\lambda} = \gamma\vec{\lambda}$$

which is easy to solve as the dimension of A is L, the number of variables in the cluster, which is typically relatively small. Once we find the L dimensional eigenvector λ^* with the largest eigenvalue γ^*, we can express with it the desired parent profile $\vec{z^*}$.

We can get a better bound of $\Delta_{X_1,\ldots,X_L}(Z)$ if we use C_2 similarity rather than C_1. Unfortunately, optimizing the profile $\vec{z}$ with respect to this similarity measure is a harder problem that is not solvable in closed form. Since the goal of the cluster identification is to provide a good starting point for the following iterations that will eventually adapt the structure, we use the closed form solution for Eq. (4.7). Note that once we optimized the profile z using the above derivation, we can still use the C_2 similarity score to provide a better bound on the quality of this profile as a new parent for $X_1, \ldots, X_L$.

Now that we can approximate the benefit of adding a hidden variable to a cluster of variables, we still need to consider different clusters to find the most beneficial one . As the number of clusters is exponential, we adapt a heuristic *agglomerative clustering* approach (Duda and Hart, 1973, e.g.,) to explore different clusters. We start with each ideal parent profile as an individual cluster and at each point we merge the two clusters that lead to the best expected improvement in the BIC score (combining the above approximation with the change in penalty terms). This procedure potentially involves $O(N^3)$ merges, where N is the number of variables. We save much of the computations by pre-computing the matrix $\mathcal{Y}^T\mathcal{Y}$ only once, and then using the relevant sub-matrix in each merge In practice, the time spent in this step is insignificant in the overall search procedure.

Learning with Missing Values

Once we consider learning structure with hidden variables, we have to deal with the issue of missing values while considering subsequent structure changes. Similar considerations can arise if the dataset contains partial observations of some of the variables.

To deal with this problem, we use an Expectation Maximization approach (Dempster et al., 1977) and its application to network structure learning (Friedman, 1997). At each step in the search we have a current network that provides an estimate of the distribution that generated the data, and use it to compute a distribution over possible completions of the data. Instead of maximizing the BIC score, we attempt to maximize the expected BIC score

$$\boldsymbol{E}_Q[BIC(\mathcal{D}, G) \mid \mathcal{D}_o] = \int Q(\mathcal{D}_h \mid \mathcal{D}_o) BIC(\mathcal{D}, G) d\mathcal{D}_h$$

where $\mathcal{D}_o$ is the observed data, $\mathcal{D}_h$ is the unobserved data, and Q is the distribution represented by the current network. As the BIC score is a sum over local terms, we can use linearity of expectations to rewrite this objective as a sum of expectations, each over the scope of a single CPD. This implies that when learning with missing values, we need to use the current network to compute the posterior distribution over the values of variables in each CPD we consider. Using these posterior distributions we can estimate the expectation of each local score, and use them in standard structure search (discussed above). Once the search algorithm converges, we use the new network for computing expectations and reiterate until convergence (see (Friedman, 1997)).

How can we combine the ideal parent method into this structural EM search? Since we do not necessarily observe neither X nor all of its parents, the definition of ideal parent cannot be applied

directly. Instead, we define the ideal parent to be the profile that will match the expectations given Q. That is, we choose $y[m]$ so that

$$\boldsymbol{E}_Q[x[m] \mid \mathcal{D}_o] = \boldsymbol{E}_Q[g(\alpha_1 u_1[m], \ldots, \alpha_k u_k[m], y[m] : \theta) \mid \mathcal{D}_o]$$

In the case of linear Gaussian CPDs, this implies that

$$\vec{y} = \boldsymbol{E}_Q[\vec{x} \mid \mathcal{D}_o] - \boldsymbol{E}_Q[\mathcal{U} \mid \mathcal{D}_o]\vec{\alpha}$$

Once we define the ideal parent, we can use it to approximate changes in the expected BIC score (given Q). For the case of linear Gaussian, we get terms that are similar to C_1 and C_2 of Proposition 4.2.2 and Proposition 4.2.3, respectively. The only change is that we apply the similarity measure on the expected value of $\vec{z}$ for each candidate parent Z. This is in contrast to exact evaluation of $\boldsymbol{E}_Q\left[\Delta_{X|\mathbf{U}} Z \mid \mathcal{D}_o\right]$, which requires the computation of the expected sufficient statistics of $\mathbf{U}$, X, and Z.

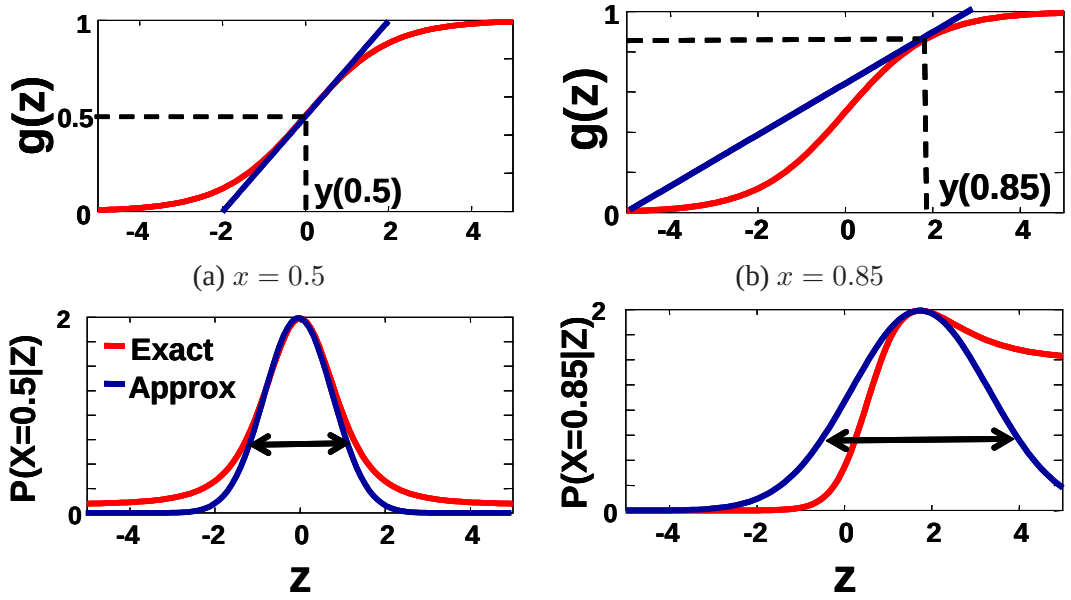
To facilitate efficient computation of the expectations, we adopt an approximate variational *mean-field* form (e.g., Jordan et al., 1998; Murphy and Weiss, 1999) for the posterior. This approximation is used both for the ideal parent method and the standard greedy approach used in Section 4.2.5. This results in computations that require only the first and second moments for each instance $z[m]$, and thus can be easily obtained from Q.

Finally, we note the structural EM iterations are still guaranteed to converge to a local maximum. In fact, this does *not* depend on the fact that C_1 and C_2 are lower bounds of the true change to the score, since these measures are only used to pre-select promising candidates which are scored before actually being considered by the search algorithm. Indeed, the ideal parent method is a modular structure candidate selection algorithm and can be used as a black-box by any search algorithm.

4.2.3 Non-linear CPDs

We now turn to dealing with the case of non-linear CPDs. In the class of CPDs we are considering, this non-linearity is mediated by the function g, which we assume here to be invertible. Examples of such function include the sigmoid function shown in Eq. (4.3) and hyperbolic functions that are suitable for modeling gene transcription regulation, such as we develop in Chapter 6, among many others. When we learn with non-linear CPDs, parameter estimation is harder. To evaluate a potential parent Z for X we have to perform non-linear optimization (*e.g.*, conjugate gradient) of all of the α coefficients of all parents as well as other parameters of g. In this case, a fast approximation can boost the computational cost of the search significantly.

As in the case of linear CPDs, we compute the ideal parent profile $\vec{y}$ by inverting g (We assume that the inversion of g can be performed in time that is proportional to the calculation of g itself as is the case in CPDs considered above.) Suppose we are considering the addition of a parent to X in addition to its current parents $\mathbf{U}$, and that we have computed the value of the ideal parent $y[m]$ for



(a) $x = 0.5$ (b) $x = 0.85$

(c) $x = 0.5$ (d) $x = 0.85$

Figure 4.5: Likelihood approximations for a non linear CPD. The top plots show the sigmoid function $g(z)$, and a linear approximation of it around $y = g^{-1}(0.5) = 0$ (a) and $y = g^{-1}(0.85)$ (b). The bottom plots show the resulting Gaussian approximation to the likelihood $P(X|Z)$. Exact functions are shown in red; Approximations are shown in blue.

each sample m by inversion of g. Now consider a particular candidate parent Z whose value at the m'th instance is $Z[m]$. How will the difference between the ideal value and the value of Z reflect in prediction of X for this instance?

In the linear case, the difference $z[m] - y[m]$ translated through g to a prediction error. In the non-linear case, the effect of difference on predicting X depends on other factors, such as the values of the other parents, despite the fact that their parameters are held fixed: Consider again the sigmoid function g of Eq. (4.3). If the sum of the arguments to g are close to 0, then g locally behaves like a sum of its arguments. On the other hand, if the sum is far from 0, the function is in one of the saturated regions, and big differences in the input almost do not change the prediction.

Our solution to this problem is to approximate g by a linear function around the value we predict. We use a first-order Taylor expansion of g around the value of $\vec{y}$ and write

$$g(\mathbf{u}, \vec{z}) \approx g(\mathbf{u}, \vec{y}) + (\vec{z} - \vec{y}) \frac{\partial g(\mathbf{u}, \vec{y})}{\partial \vec{y}}$$

As a result, the "penalty" for a distance between $\vec{z}$ and $\vec{y}$ depends on the gradient of g at the particular value of $\vec{y}$, given the value of the other parents. In instances where the derivative is small, larger deviations between y and z have little impact on the likelihood of X, and in other instances where the derivative is large, the same deviations may lead to worse likelihood. Figure 4.5 shows the

linear approximation for a sigmoid function with a single parent, Z, for two values of the dependent variable X. When $x = 0.5$, the value of the ideal parent is $y = g^{-1}(0.5) = 0$. The derivative of g around this point is the highest, and the approximation of $P(X|Z)$ is sensitive to small changes in z. On the other hand, the linear approximation is the best at this point, and so even when the deviation of z from y is large, the likelihood approximation is still good. When $x = 0.85$, the gradient at the corresponding y value is smaller, and the sensitivity to the deviations is smaller. As we can see, the quality of likelihood approximation is not as good for this point.

With this linear approximation, we can develop similarity measures that parallel the developments for the linear case. There are two main differences. First, due to the use of Taylor expansion, we can no longer prove that these are underestimates of the likelihood. Second, due to the different value of the gradient at different instances, the contribution of distance at each instance will be weighted differently.

Proposition 4.2.4 *Suppose that X has parents $\mathbf{U}$ with a set $\vec{\alpha}$ of scaling factors. Let Y be the ideal parent as described above, and Z be some candidate parent. Then the change in log-likelihood of X in the data, when adding Z as a parent of X, while freezing all other parameters, is approximately*

$$C_1(\vec{y} \star g'(y), \vec{z} \star g'(y)) - \frac{1}{2\sigma^2}(c_1 - c_2). \tag{4.9}$$

where $g'(y)$ is the vector whose m'th component is $\partial g(\vec{\alpha}\mathbf{u}, y)/\partial y \mid_{\mathbf{u}[m], y[m]}$, and $\star$ denotes component-wise product. Similarly, if we also optimize the variance, then the change in log-likelihood is approximately

$$C_2(\vec{y} \star g'(y), \vec{z} \star g'(y)) - \frac{M}{2} \log \frac{c_1}{c_2} \tag{4.10}$$

In both cases,

$$c_1 = (\vec{y} \star g'(y)) \cdot (\vec{y} \star g'(y)) \; ; \; c_2 = (\vec{x} - g(\mathbf{u})) \cdot (\vec{x} - g(\mathbf{u}))$$

do not depend on $\vec{z}$.

Thus, we can use exactly the same measures as before, except that we "distort" the geometry with the weight vector $g'(y)$ that determines the importance of different instances. Instances where the gradient is higher have a higher weight. These instances are more sensitive to the value of z, but the likelihood approximation for them is more accurate, as we have seen. In this sense, the weighting also compensates for inaccuracies caused by the approximation.

To approximate the likelihood difference, we also add the correction term which is a function of c_1 and c_2. This correction is not necessary when comparing two candidates for the same family, but is required for comparing candidates from different families, or when adding hidden values. Note that if g is linear then the correction term vanishes altogether and Proposition 4.2.2 and Proposition 4.2.3 are recovered.

We can now efficiently perform the ideal *add* and *replace* steps in the structure search. The complexity is again $O(M(d-1))$ for computing the ideal profile and $O(KM)$ for computing the similarity measure for each candidate parent. As before, the significant gain in speed is that we only perform few parameter optimizations (that are expected to be costly as the number of parents grows), rather than $O(N)$ such optimizations.

Adding a new hidden variable with non-linear CPDs introduces further complication. We want to use, similarly to the case of a linear model, the structure score of Eq. (4.6) with the distorted C_1 measure. Optimizing this measure has no closed form solution in this case and we need to resort to an iterative procedure or an alternative approximation. In here, we approximate Eq. (4.9) with a form that is similar to the linear Gaussian case, with the "distorted" geometry of $\vec{y}$.

4.2.4 Other Noise Models

Up to now we have only handled CPDs where the uncertainty is modeled using an additive Gaussian term. In some cases we wish to use different forms of noise. For example, as we discussed in Chapter 1, the biological processes related to regulation as well as the related measurement methods have many noise components which are multiplicative. Such a noise process can be modeled using CPDs of the form

$$X = g(\alpha_1 \mathbf{u}_1, \ldots, \alpha_k \mathbf{u}_k : \theta)(1+\epsilon) \tag{4.11}$$

where, as in Eq. (4.2) ϵ is a zero mean noise random variable. Another popular choice for modeling multiplicative noise is the log-normal form:

$$\log(X) = \log(g(\alpha_1 \mathbf{u}_1, \ldots, \alpha_k \mathbf{u}_k : \theta)) + \epsilon \tag{4.12}$$

that is, the log of the random variable is distributed normally. In this section we try to generalize the concepts introduced so far to such general CPD forms, and present explicit results for the multiplicative noise CPD of Eq. (4.11).

Let us first denote the conditional density function of X in the following general form:

$$P(X \mid \mathbf{U}) = q(X; g(\alpha_1 u_1[m], \ldots, \alpha_k u_k[m] : \theta), \phi)$$

where g is the link function as before, and ϕ denotes parameters of the "noise" part of the function (e.g. variance parameters). In the additive case of Eq. (4.2) we have $q = \mathcal{N}(X; g, \sigma^2)$. In the multiplicative case above we have $q = \mathcal{N}(X; g, (g\sigma)^2)$. We first note that in some unimodal CPD forms, like the multiplicative noise CPD, g is not the actual mode of q. Since our motivation in defining the ideal parent profile $\vec{y}$ is to maximize the likelihood of the child variable instances $\vec{x}$, we now redefine it in a more generalized way:

Definition 4.2.5: Given a dataset $\mathcal{D}$, and a CPD $P(X; \mathbf{U}) = q(X; g(\mathbf{U}), \phi)$ for X given its parents

$\mathbf{U}$ with parameters θ, α and ϕ, the *ideal parent* Y of X is such that for each instance m,

$$\left.\frac{\partial q(x[m];g)}{\partial g}\right|_{g=g(\alpha_1 u_1[m],\ldots,\alpha_k u_k[m],y[m]:\theta)} = 0 \tag{4.13}$$

■

That is, $\vec{y}$ is the vector which makes $g(\mathbf{u},\vec{y})$ maximize the likelihood of the child variable at each instance. Since $\frac{\partial q}{\partial y} = \frac{\partial q}{\partial g}\frac{\partial g}{\partial y} = 0$, this definition also means that the ideal parent maximizes the likelihood w.r.t. the values of a new parent. We note that in case the distribution is a simple Gaussian with whatever form of g, this definition coincides with Definition 4.2.1. Of course, the definition is useful only if $\vec{y}$ can be extracted efficiently from Eq. (4.13).

The new definition motivates us to use a different approximation than before when computing the score difference. We now choose to approximate the log likelihood function directly around $\vec{y}$. As we shall see, for the case of additive Gaussian noise modeling, the results coincide with those obtained when approximating g. The second order approximation is:

$$\log P(\vec{x} \mid \mathbf{u}, \alpha_z z) \approx \log P(\vec{x} \mid \mathbf{u}, y) + (\alpha_z \vec{z} - \vec{y}) \cdot \frac{\partial \log P(\vec{x} \mid \mathbf{u}, \vec{y})}{\partial \vec{y}} + \frac{1}{2}(\alpha_z \vec{z} - \vec{y})^T H (\alpha_z \vec{z} - \vec{y})$$

where H is the Hessian matrix of $\log P(\vec{x} \mid \mathbf{u}, \vec{z})$ at the point $\alpha_z \vec{z} = \vec{y}$. The first order term vanishes since as we noted $\frac{\partial q}{\partial y} = 0$, which implies also $\frac{\partial \log(q)}{\partial y} = 0$. Using the chain rule, we can derive the expression for the Hessian:

$$\begin{aligned} H_{m,n} &= \frac{\partial^2 \log P(\vec{x} \mid \mathbf{u}, \vec{y})}{\partial y[m] \partial y[n]} \\ &= \delta_{mn} \frac{1}{q_m^2} \left(- \left(\frac{\partial q_m}{\partial g_m} \frac{\partial g_m}{\partial y[m]} \right)^2 + q_m \left\{ \frac{\partial^2 q_m}{\partial {g_m}^2} \left(\frac{\partial g_m}{\partial y[m]} \right)^2 + \frac{\partial q_m}{\partial g_m} \frac{\partial^2 g_m}{\partial y[m]^2} \right\} \right) \end{aligned}$$

where we used the abbreviations $g_m \equiv g(\mathbf{u}[m], y[m])$ and $q_m \equiv q(x[m]; g_m)$. The Hessian matrix is always diagonal, since each term in the log likelihood involves $y[m]$ from a single sample m. After eliminating terms involving $\frac{\partial q}{\partial g}$, the Hessian simplifies to:

$$H_{m,n} = \delta_{mn} \frac{1}{q_m} \frac{\partial^2 q_m}{\partial {g_m}^2} \left({g'}_m \right)^2$$

where ${g'}_m \equiv \frac{\partial q_m}{\partial y[m]}$, and the approximation equation boils down to:

$$\log P(\vec{x} \mid \mathbf{u}, \alpha_z \vec{z}) \approx \log P(\vec{x} \mid \mathbf{u}, \vec{y}) + \frac{1}{2} \sum_m \frac{(\alpha_z z[m] - y[m])^2}{q_m} \frac{\partial^2 q_m}{\partial {g_m}^2} \left({g'}_m \right)^2 \tag{4.14}$$

As in the linear Gaussian case, we managed to write the likelihood difference using some distance between the new parent $\alpha_z \vec{z}$ and the ideal parent $\vec{y}$, and as in the non-linear case, this distance is

deformed by different weighting for different samples. Using this deformation, we can define the C_1 measure for the generalized CPD case similarly to previous definitions:

Proposition 4.2.6 *Suppose that X has parents $\mathbf{U}$ with a set $\vec{\alpha}$ of scaling factors. Let Y be the ideal parent as defined in Eq. (4.13), and Z be some candidate parent. Then the change in log-likelihood of X in the data, when adding Z as a parent of X, while freezing all other parameters, is approximately*

$$\begin{aligned} C_1(\vec{y},\vec{z}) &\approx \log P(\vec{x} \mid \mathbf{u},\vec{y}) - \max_{\alpha_Z} \frac{1}{2} K(\alpha_z\vec{z}-\vec{y},\alpha_z\vec{z}-\vec{y}) - \log P(\vec{x} \mid \mathbf{u}) \\ &= \log P(\vec{x} \mid \mathbf{u},\vec{y}) - \frac{1}{2}K(\vec{y},\vec{y}) + \frac{1}{2}\frac{(K(\vec{y},\vec{z}))^2}{K(\vec{z},\vec{z})} - \log P(\vec{x} \mid \mathbf{u}) \end{aligned}$$

where $K(.,.)$ is an inner product of two vectors defined as:

$$K(\vec{a},\vec{b}) = \sum_m a[m]b[m]\frac{-1}{q_m}\frac{\partial^2 q_m}{\partial {g_m}^2}\left({g'}_m\right)^2$$

The first equation follows directly from Eq. (4.14), and the second one follows from replacing α_z with its maximum likelihood estimator, which is $\frac{K(\vec{y},\vec{z})}{K(\vec{z},\vec{z})}$. The inner product K entails the deformation for the general case: The factor $({g'}_m)^2$ weights each vector by the gradient of g, as explained in Section 4.2.3. The new factor $\frac{-1}{q_m}\frac{\partial^2 q_m}{\partial {g_m}^2}$ measures the sensitivity of q_m to changes in g_m for each instance. Since we are dealing with a maximum point of q_m, this factor is always positive. In the Gaussian noise models we have considered in the previous sections, it equals a constant: $\frac{1}{\sigma^2}$. In non-Gaussian models, this sensitivity can vary between instances.

It is easy to see the generalized definition of C_1 coincides with the two previous ones given for linear and for non-linear Gaussian CPDs: For a linear Gaussian CPD we have ${g'}_m = 1$, and so $K(\vec{a},\vec{b}) = \frac{1}{\sigma^2}\vec{a}\cdot\vec{b}$. All terms which do not depend on $\vec{z}$ cancel out in this case, resulting in our original definition of C_1 from Proposition 4.2.2. For the non-linear Gaussian case, we get $K(\vec{a},\vec{b}) = \frac{1}{\sigma^2}(\vec{a}\star g'(y))\cdot(\vec{b}\star g'(y))$, and the original form of Proposition 4.2.4 is obtained. The power of the new definition is that it is applicable to any link function and unimodal noise model. The difference between different distributions and different link functions will be in the form of the derivatives inside the kernel function K, and in the additional $\log P$ terms. We note that we cannot give a similarly general expression for C_2, since it requires optimizing both σ and α_z simultaneously. The solution to this problem depends on the form of the distribution.

We now solve the case of the multiplicative noise density of Eq. (4.11). First, let's write it in an explicit density form:

$$P(x \mid \mathbf{u}) = \frac{1}{\sqrt{2\Pi}\sigma|g(\mathbf{u})|}\exp\left(-\frac{1}{2\sigma^2}\left(\frac{x}{g(\mathbf{u})}-1\right)^2\right)$$

To avoid singularity, we will restrict the values of g to be positive. The partial derivatives of q_n are:

$$\begin{aligned}\frac{\partial q_m}{\partial g_m} &= \left[-\frac{1}{g_m}+\frac{1}{\sigma^2}\left(\frac{x}{g_m}-1\right)\frac{x}{g_m^2}\right]q_m \\ \frac{\partial^2 q_m}{\partial {g_m}^2} &= \left[-\frac{1}{g_m}+\frac{1}{\sigma^2}\left(\frac{x}{g_m}-1\right)\frac{x}{g_m^2}\right]^2 q_m + \left[\frac{1}{g_m^2}+\frac{1}{\sigma^2}\left(\frac{x}{g_m}-1\right)\frac{-2x}{g_m^3}-\frac{1}{\sigma^2}\frac{x^2}{g_m^4}\right]q_m\end{aligned}$$

By definition of $\vec{y}$ the first derivative zeros out, and so:

$$-\frac{1}{g_m}+\frac{1}{\sigma^2}\left(\frac{x}{g_m}-1\right)\frac{x}{g_m^2}=0 \tag{4.15}$$

which results in the following requirement for $\vec{y}$:

$$g(\alpha_1 u_1[m],\ldots,\alpha_k u_k[m], y[m]:\theta) = x[m]\frac{-1+\sqrt{1+4\sigma^2}}{2\sigma^2} \tag{4.16}$$

(The negative solution is discarded due to the constraint $g>0$). Note that now the link function should equal a scaled version of $x[m]$. We can now extract $y[m]$ as before by inverting g_n.

The remaining terms for the second derivative are:

$$\begin{aligned}\frac{\partial^2 q}{\partial g^2} &= \left[\frac{1}{g^2}-\frac{1}{\sigma^2}\left(\frac{x}{g}-1\right)\frac{2x}{g^3}-\frac{1}{\sigma^2}\frac{x^2}{g^4}\right]q \\ &= -\frac{1}{g^2}\left[1+\frac{1}{\sigma^2}\frac{x^2}{g^2}\right]q \\ &= -\frac{1}{g^2}k_\sigma q\end{aligned}$$

The second equation comes from substituting in Eq. (4.15) and the third equation from plugging in Eq. (4.16), where k_σ is a positive constant function of σ. We can now express K similarly to the dot product expressions we used before:

$$K(\vec{a},\vec{b}) = k_\sigma(\vec{a}\star\frac{g'(y)}{g(y)})\cdot(\vec{b}\star\frac{g'(y)}{g(y)}) \tag{4.17}$$

where $\frac{g'(y)}{g(y)}$ is the vector whose m'th component is $\frac{g'_m}{g_m}$. This weighting is similar to the one we used for the non-linear Gaussian case, but additionally scales down each instance m by g_n. This has an intuitive explanation: Since in the multiplicative density the noise level is expected to go up with g, this rescaling brings all samples to the same noise scale.

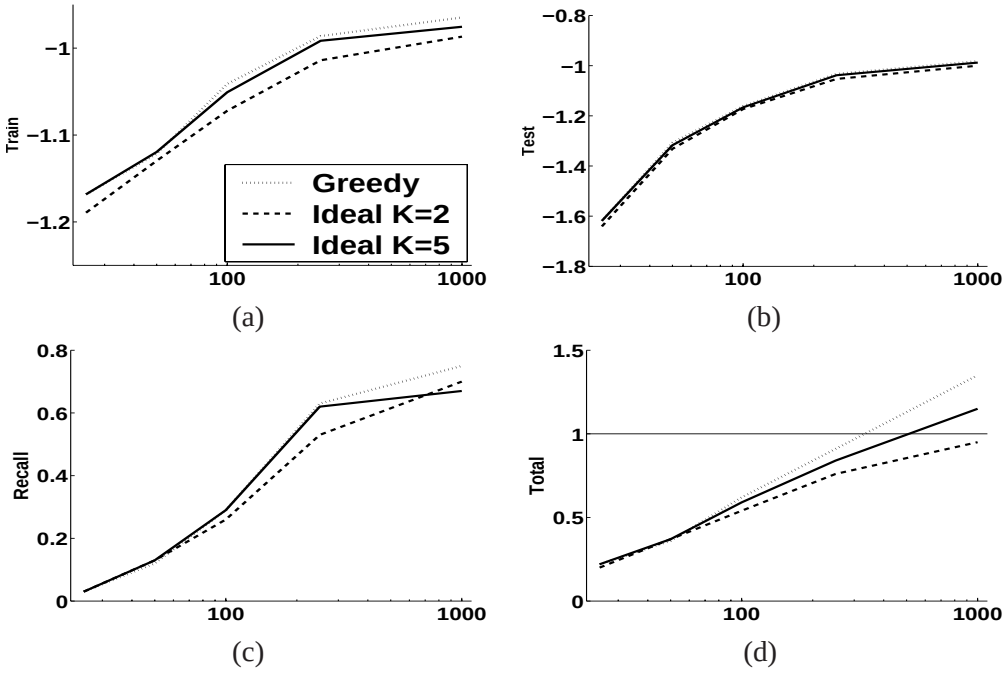


Figure 4.6: Evaluation of using Ideal search on synthetic data with 44 variables. We compare Ideal search with $K = 2$ (dashed) and $K = 5$ (solid), against the standard Greedy procedure (dotted). The figures show, as a function of the number of instances (x-axis), for linear Gaussian CPDs: (a) average training log likelihood per instance per variable; (b) same for test; (c) percent of true edges obtained in learned structure; (d) total number of edges learned as percent of true number.

For completeness, we write the additional $\log P$ terms in the expression of C_1 for the multiplicative density:

$$\begin{aligned}
\log P(\vec{x} \mid \mathbf{u}, \vec{y}) - \log P(\vec{x} \mid \mathbf{u}) &= -\sum \log(\sigma' g(\mathbf{u}[m], y[m])) + \sum \log(\sigma g(\mathbf{u}[m])) - \\
&\quad \frac{1}{2\sigma'^2} \sum (\frac{x[m]}{g(\mathbf{u}[m], y[m])} - 1)^2 + \frac{1}{2\sigma^2} \sum (\frac{x[m]}{g(\mathbf{u}[m])} - 1)^2 \\
&= -M \log \frac{-1 + \sqrt{1 + 4\sigma'^2}}{2\sigma'} - \sum \log x[m] + \sum \log \sigma g(\mathbf{u}[m]) - \\
&\quad \frac{M}{2\sigma'^2} \left(\frac{2\sigma'^2}{-1 + \sqrt{1 + 4\sigma'^2}} - 1 \right)^2 + \frac{1}{2\sigma^2} \sum (\frac{x[m]}{g(\mathbf{u}[m])} - 1)^2
\end{aligned}$$

where σ' denotes the new variance parameter. For C_1 it is the same as σ. For C_2, in this case there is no closed form solution for the joint optimization problem. A possible alternative is to optimize α_z and σ' iteratively.

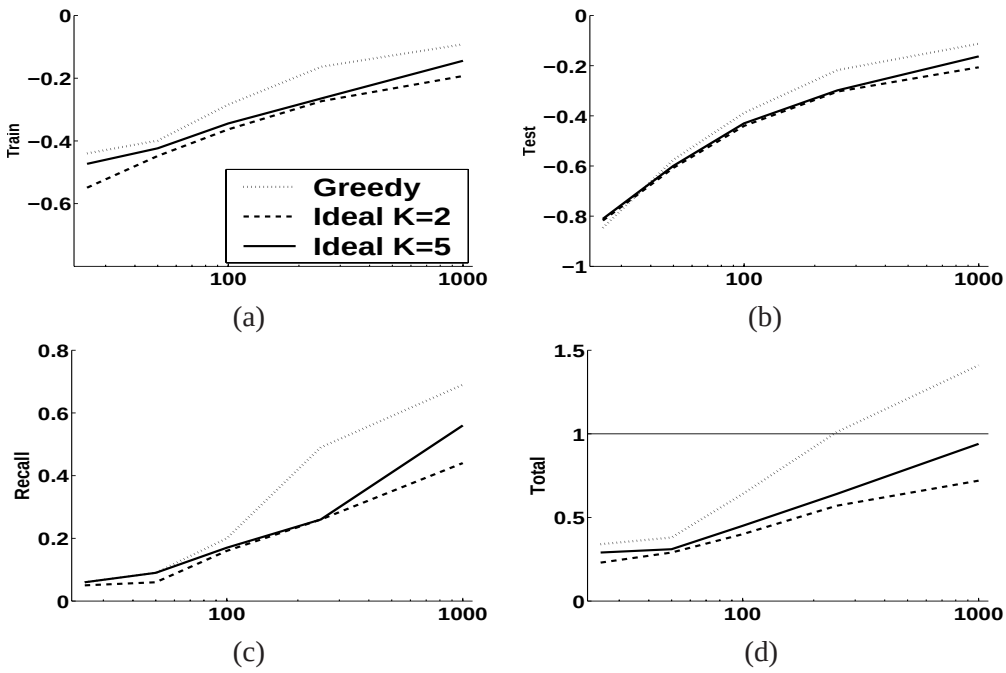


Figure 4.7: Same as Figure 4.6, for Sigmoid CPDs.

4.2.5 Experimental Evaluation

We now examine the impact of the ideal parent method in two settings. In the first setting, we use this method for pruning the number of potential moves that are evaluated by greedy hill climbing structure search. We apply this learning procedure to complete data (and data with some missing values) to learn dependencies between the observed variables. In the second setting, we use the ideal parent method as a way of introducing new hidden variables, and also use it as a guide to reduce the number of evaluations when learning structure that involves hidden variables and observed ones with a Structural EM search procedure.

In the first setting, we applied standard greedy hill climbing search (Greedy) and greedy hill climbing supplemented by the ideal parent method as discussed in Section 4.2.2 (Ideal). In using the ideal parent method, we used the C_2 similarity measure (Section 4.2.1) to rank candidate edge additions and replacements, and then applied full scoring only to the top K ranking candidates per variable.

To evaluate the impact of the method, we start with a synthetic experiment where we know the true underlying network structure. In this setting we can evaluate the magnitude of the performance cost resulting from the approximation we use. We used a network learned from real data (see below) with 44 variables. From this network we can generate datasets of different sizes and apply our method with different values of K. Figure 4.6 and Figure 4.7 compare the ideal parent method and the standard greedy procedure for linear Gaussian CPDs and Sigmoid CPDs. Using $K = 5$ is, as we expect, closer to the performance of the standard greedy method both in terms of training set and

			Ideal $K = 2$ vs Greedy						Ideal $K = 5$ vs Greedy					
Dataset	N	M	train	test	edges	moves	eval	speed	train	test	edges	moves	eval	speed
Linear Gaussian														
AA full	44	173	-0.024	0.006	87.1	96.5	3.6	2	-0.008	0.007	94.9	96.5	9.3	2
AA full Cond	173	44	-0.038	0.082	92.2	92.6	1.2	2	-0.009	0.029	96.9	98.2	2.9	2
Met full	89	173	-0.033	-0.024	88.7	91.5	1.6	3	-0.013	-0.016	94.5	96.9	4.4	2
Met full Cond	173	89	-0.035	-0.015	91.3	98.0	1.0	2	-0.007	-0.023	98.9	98.5	2.4	2
Linear Gaussian with missing values														
AA	354	173	-0.101	-0.034	81.3	95.2	0.4	5	-0.048	-0.022	90.7	96.0	0.9	5
AA Cond	173	354	-0.066	-0.037	74.7	87.5	0.4	14	-0.033	-0.021	86.3	101.1	1.6	11
Sigmoid														
AA full	44	173	-0.132	-0.065	49.7	59.4	2.0	38	-0.103	-0.046	60.4	77.6	6.1	18
AA full Cond	173	44	-0.218	0.122	62.3	76.7	1.0	36	-0.150	0.103	73.7	79.4	2.3	21
Met full	89	173	-0.192	-0.084	47.9	58.3	0.9	65	-0.158	-0.059	56.6	69.8	2.6	29
Met full Cond	173	89	-0.207	-0.030	60.5	69.5	0.8	53	-0.156	-0.042	69.8	77.7	2.2	29

Table 4.2: Performance comparison of the Ideal parent search with $K = 2$ or $K = 5$ and Greedy on real data sets. *vars* - number of variables in the dataset; N - number of variables; M - number of instances; *train* - average difference in training set log likelihood per instance per variable; *test* - same for test set; *edges* - percent of edges learned by Ideal with respect to those learned by Greedy. *moves* - percent of structure modifications taken during the search; *eval* - percent of moves evaluated; *speed* - speedup of Ideal over greedy method. All figures are average over 5 fold cross validation sets.

test set performance then $K = 2$. For linear Gaussian CPDs test performance is essentially the same with a slight advantage for the standard greedy method using Sigmoid CPDs. When considering the percent of true edges recovered , again the standard method shows some advantage over the ideal method with $K = 5$. However, looking at the total number of edges learned , we can see that the standard greedy method achieves this by using close to 50% more edges then the original structure for Sigmoid CPDs. Thus, advantage in performance comes at a high complexity price (and as we demonstrate below, at a significant speed cost).

We now examine the effect of the method on learning from real-life datasets. We base our datasets on a study that measures the expression of the baker's yeast genes in 173 experiments (Gasch et al., 2000). In this study, researchers measured expression of 6152 yeast genes in its response to changes in the environmental conditions, resulting in a matrix of 173×6152 measurements. In the following, for practical reasons, we use two sets of genes. The first set consists of 639 genes that participate in general metabolic processes (Met), and the second is a subset of the first with 354 genes which are specific to amino acid metabolism (AA). We choose these sets since part of the response of the yeast to changes in its environment is in altering the activity levels of different parts of its metabolism. For some of the experiments below, we focused on subsets of genes for which there are no missing values (full, consisting of 89 and 44 genes, respectively).

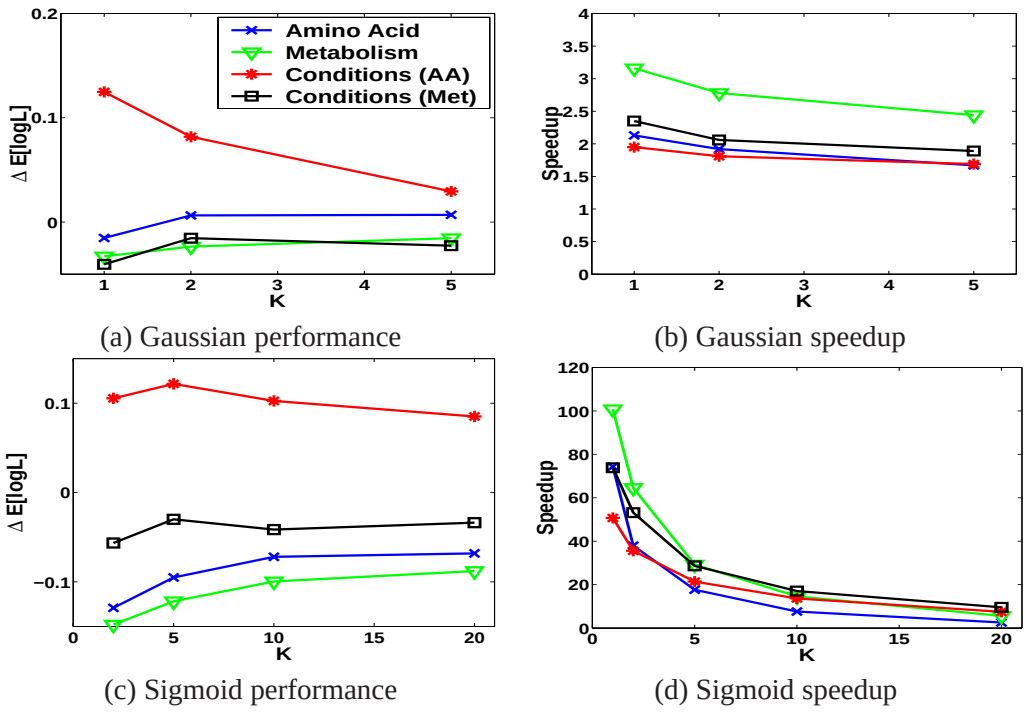
On these datasets we can consider two tasks. In the first, we treat genes as variables and experiments as instances. The learned networks indicate possible regulatory or functional connections between genes, as we discuss in Chapter 5. A complementary task is to treat experiments as variables (Cond). A learned network in this scenario indicates dependencies between the responses to different conditions.

In Table 4.2 we summarize differences between the Greedy search and the Ideal search with K set to 2 and 5, for the linear Gaussian CPDs as well as sigmoid CPDs. Since the C_2 similarity is only a lower bound of the BIC score difference, we expect the candidate ranking of the two to be different. As most of the difference comes from freezing some of the parameters, a possible outcome is that the Ideal search is less prone to overfitting. Indeed as we see, though the training set log likelihood in most cases is lower for Ideal search, the test set performance is comparable or better.

Of particular interest is the tradeoff between accuracy and speed when using the ideal parent method. In Figure 4.8 we examine this tradeoff in four of the data sets described above using linear Gaussian and sigmoid CPDs. In both cases, the performance of the ideal parent method approaches that of standard greedy as K is increased. As we can expect, in both types of CPDs the ideal parent method is faster even for $K = 5$. However, the effect on total run time is much more pronounced when learning networks with non-linear CPDs. In this case, most of the computation is spent in optimizing the parameters for scoring candidates. And so, reducing the number of candidates evaluated results in a dramatic effect. As Table 4.2 shows, the number of score evaluations with the ideal heuristic is usually a small fraction of the number of evaluations carried out by the standard search. This speedup in non-linear networks makes previously "intractable" real-life learning problems (like gene regulation network inference) more accessible.

In the second experimental setting, we examine the ability of our algorithm to learn structures that involve hidden variables and introduce new ones during the search. In this setting, we focus on *two layered networks* where the first layer consists of hidden variables, all of which are assumed to be roots, and the second layer consists of observed variables. Each of the observed variables is a leaf and can depend on one or more hidden variables. Learning such networks involves introducing different hidden variables, and determining for each observed variable which hidden variables it depends on.

To test the performance of our algorithm, we used a network topology that is curated from biological literature for the regulation of cell-cycle genes in yeast (seeSection 6.4.1). This network involves 7 hidden variables and 141 observed variables. We learned the parameters for the network (using either linear Gaussian CPDs or sigmoid CPDs) from a cell cycle gene expression dataset (Spellman et al., 1998). From the learned network we then sampled datasets of varying sizes, and tried to recreate the regulation structure using either greedy search or ideal parent search with the corresponding type of CPDs. In both search procedures we introduce hidden variables in a gradual manner. We start with a network where a single hidden variable is connected as the only parent to all observed variables. After parameter optimization, we introduce another hidden variable - either as a parent of all observed variables (in greedy search), or to members of the highest scoring cluster (in ideal parent search, as explained in Section 4.2.2). We then let the structure search modify edges (subject to the two-layer constraints) until no beneficial moves are found, at which point we introduce another hidden variable, and so on. The search terminates when it is no longer beneficial to add a new variable.



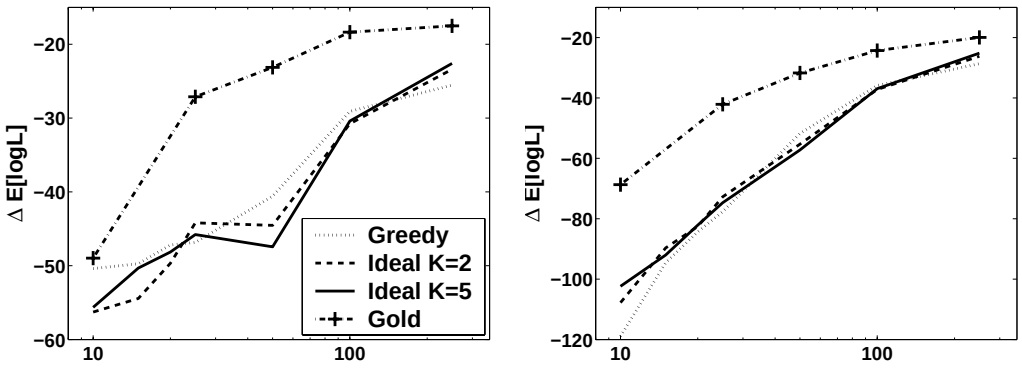
(a) Gaussian performance (b) Gaussian speedup

(c) Sigmoid performance (d) Sigmoid speedup

Figure 4.8: Evaluation of Ideal search on real-life data. (a) average log likelihood per instance on test data (based on 5-fold cross validation) relative to Greedy when learning with linear Gaussian CPDs (y-axis) against K (x-axis). (b) Speedup over Greedy (y-axis) against K (x-axis). (c),(d) same for sigmoid CPDs.

Figure 4.9 shows the performance of the ideal parent search and the standard greedy procedure as a function of the number of instances. As can be seen, although there are some differences in training set likelihood, the performance on test data is essentially similar, and approaches that of the Golden model (true structures with trained parameters) as the number of training instances grows. Thus, as in the case of the yeast experiments considered above, there was no degradation of performance due to the approximation made by our method.

We then considered the application of our algorithm to the real-life cell-cycle gene expression data described above with linear Gaussian CPDs. Although this data set contains only 17 samples, it is of high interest from a biological perspective to infer from it as much as possible on the structure of regulation. We performed leave-one-out cross validation and compared the Ideal parent method with $K = 2$ and $K = 5$ to the Greedy. To avoid overfitting, we limited the number of hidden parents for each observed variable to 2. Although the standard greedy procedure achieved higher train log-likelihood performance, its test performance is significantly worse as a result of overfitting for two particular instances. As we have demonstrated in the synthetic example above, the ability of the ideal method to avoid overfitting via a guided search does not come at the price of diminished performance when data is more plentiful. When the observed variables were allowed to have up to 5 parents, all methods demonstrated overfitting which for Greedy was by far more severe.



(a) train (b) test

Figure 4.9: Evaluation of performance in **two-layer network** experiments on synthetic data. The data was generated from a model learned from real data with 143 observable variables. (a) average log likelihood per instance on *training* data (y-axis) for Greedy , Ideal search with $K = 2$ and Ideal search with $K = 5$, and the true structure with trained parameters Golden, when learning with linear Gaussian CPDs against the number of training samples (x-axis). (b) Same for *test* set.

Finally, as we expect the nature of biological interactions in the cell cycle regulation domain to be non-linear, we set out to learn a model for this data using the richer but computationally demanding Sigmoid CPDs. To make computations tractable, we used in both methods a variational mean field approximation for computing posteriors over the hidden variables in the E-step of Structural EM. Unfortunately, even in this limited setting the standard greedy method proved non-feasible. The ideal method produced networks with interesting structure which, as we show inChapter 6 can be subject to biological analysis.

4.2.6 Discussion

In this part we set out to learn continuous variable networks. We addressed two fundamental challenges: First, we show how to speed up structure search, particularly for non-linear conditional probability distributions. This speedup is essential as it makes structure learning feasible in many interesting real life problems. Second, we show a principled way of introducing new hidden variables into the network structure. We use the concept of an "ideal parent" for both of these tasks and show its benefits on both synthetic and real-life biological domains.

The unique aspect of the Ideal parent approach is that it leverages on the parametric structure of the conditional distributions. In here, we applied this in conjunction with a greedy search algorithm. However, it can be supplemented to many other search procedures, such as simulated annealing, as a way of speeding up evaluation of candidate moves. Of particular interest is how the "Ideal Parent" method can help algorithms that inherently limit the search to promising candidates such as the "Sparse Candidate" algorithm.

Few works touched on the issue of when and how to add a hidden variable in the network structure (e.g. Elidan and Friedman, 2003; Elidan et al., 2001; Martin and VanLehn, 1995; Zhang,

2002). Only some of these methods are potentially applicable to continuous variable networks, and none have been adapted to this context. To our knowledge, this is the first work to address this issue in a general context of continuous variable networks.

Many challenges remain. First, to combine the Ideal Parent method within other search procedures as a plug-in for candidate selection. Second, to apply the method to additional and more complex conditional probability distributions, and to leverage the connection to Generalized Linear Models (McCullagh and Nelder, 1989), where a variety of optimization methods for specific types of CPDs exist. Finally, to use better approximations for adding new hidden variables in the non-linear case.

4.3 Discussion: Comparing the Two Methods

The "Sparse Candidate" and "Ideal Parent" methods share the same motivation. Both offer heuristic methods for speeding structure search. The "Sparse Candidate" is a complete algorithm for structure search, which treats the parametric representation and choice of family score as a black box. The "Ideal Parent" idea supplies a heuristic method for fast estimation of family scores.

There are several ways in which these two methods can be married. The most obvious one is applying the "Ideal Parent" method within the Maximize step of the "Sparse Candidate" search. Instead of doing full greedy search within the restricted space of networks, we can use one of the Ideal method variants. This would significantly speed up that step, which is still exponential in cost, though in a much more reduced space.

Another called for application of the "Ideal Parent" method is within the Restrict step of the search. As we noted in Section 4.1.2, of all the measures we suggested for choosing candidate parents, the score measure is the only one directly applicable to most continuous variable conditional density representations. For non-linear dependencies this step becomes very costly, due to non-linear optimization involved in score computation. This is especially true for the first iteration, where $O(N^2)$ measure computations are needed. We can therefore suggest a new measure, based on the ideal parent method, which we call the *ideal* measure:

$$\mathrm{M}_{\mathrm{Ideal}}(X_i, X_j \mid B) = C_2(y_{X_i}^{\vec{}}, x_j^{\vec{}})$$

where $y_{X_i}^{\vec{}}$ is the ideal parent profile of X_i given its current parents in the network B.

We note that the Restrict step using $\mathrm{M}_{\mathrm{Ideal}}$ is similar to the candidate selection stage in the supplemented greedy search we described in Section 4.2.2. When using the same K in both cases, we end up with the same list of candidate additional parents for each variable. The difference is in what we do next: in the supplemented greedy search we fully score all candidates and take a single structure move (the highest scoring one). In "Sparse Candidate" we run full greedy search restricted to the candidate parent lists, until convergence.

These two approaches are two extremes over a continuum, and so we can naturally suggest

intermediate approaches. For example, after the Restrict step with M_{Ideal}, we can run only a fixed number of greedy steps in the Maximize step, or run the search until score improvement drops to a certain level. These choices, together with the usage of the Ideal method in the Maximize step, lead to different tradeoffs between efficiency and accuracy.

Chapter 5

Discrete and Linear Modeling of Regulatory Networks

The main research question posed in this thesis is that of modeling gene regulatory networks using gene expression data. In the last three chapters we introduced our basic modeling language, as well as some advanced representation and search techniques. We are now ready to present the first use of this language to our stated purpose. In this chapter, we introduce a new approach for analyzing gene expression patterns which uncovers properties of the transcriptional program by examining statistical cues in the data. We base this approach on Bayesian networks. The approach, probabilistic in nature, is capable of handling noise as well as estimating the confidence in the different features of the network. We are therefore able to focus on interactions whose signal in the data is strong.

Why are Bayesian networks suitable for analyzing gene expression patterns? First, they are particularly useful for describing processes composed of *locally* interacting components; that is, the value of each component *directly* depends on the values of a relatively small number of components. Second, as we showed in the previous chapters, there are well understood statistical foundations for learning Bayesian networks from observations, and computational algorithms to do so, which have been used successfully in many applications. Finally, Bayesian networks provide models of causal influence: Although they are mathematically defined strictly in terms of probabilities and conditional independence statements, a connection can be made between this characterization and the notion of *direct causal influence*, as we discussed in Section 2.5. Although this connection depends on several assumptions that do not necessarily hold in gene expression data, the conclusions of Bayesian network analysis might be indicative about some causal connections in the data.

The remainder of this chapter is organized as follows. In Section 5.1 we describe how Bayesian networks can be applied to model interactions among genes and discuss the technical issues that are posed by this type of data. In Section 5.2 we apply our approach to the gene-expression data of Spellman et al. (1998), analyzing the statistical significance of the results and their biological

plausibility. Finally, in Section 5.3, we conclude with a discussion of related approaches and future work.

5.1 Analyzing Expression Data

We start with our modeling assumptions and the type of conclusions we expect to find. Our aim is to understand a particular *system* (a cell or an organism and its environment) that is the subject of investigation. At each point in time, the system is in some *state*. For example, the state of a cell can be defined in terms of the concentration of proteins and metabolites in the various compartments, the amount of external ligands that bind to receptors on the cell's membrane, the concentration of different mRNA molecules in the cytoplasm, etc. The cell (or other biological system) prefers certain states to others. Thus, some states are more probable if we consider random sampling of the system. The probability distribution over states captures the "region" of state space the cell attempts to stay in (other states are either visited for only brief periods of time, or are never reached).

Our aim is to estimate such a probability distribution and understand its structural features. Of course, a state of a system can be infinitely complex. Thus, we describe a distribution over only some of the attributes of states. In this chapter, we are mainly dealing with random variables that denote the mRNA expression level of specific genes. However, we can also consider other random variables that denote other aspects of the system state, such as the phase of the system in the the cell-cycle process. Other examples include measurements of experimental conditions, temporal indicators (i.e., the time/stage that the sample was taken from), background variables (e.g., which clinical procedure was used to get a biopsy sample), and exogenous cellular conditions.

We thus attempt to build a model which is a joint distribution over a collection of random variables that describe the system states. If we had such a model in a form of a Bayesian network, we could answer a wide range of queries about the system. For example, does the expression level of a particular gene depend on the experimental condition? Is this dependence direct, or indirect? If it is indirect, which genes mediate the dependency? Not having a model at hand, we want to learn one from the available data and use it to answer questions about the system.

The most abundant form of data available is mRNA expression measurements, mostly from microarray experiments. In order to learn such a model from expression data, we need to deal with several important issues that arise when learning in the this domain. These involve statistical aspects of interpreting the results, algorithmic complexity issues in learning from the data, and the choice of local probability models. In the following sections we describe how we handle each of these issues.

Most of the difficulties in learning from expression data revolve around the following central point: Contrary to most situations where one attempts to learn models (and in particular Bayesian networks), expression data involves transcript levels of thousands of genes while current data sets typically contain a few dozen samples. This raises problems in both computational complexity and the statistical significance of the resulting networks. On the positive side, genetic regulation networks are believed to be sparse, i.e., given a gene, it is assumed that no more than a few genes

directly affect its transcription. Bayesian networks are especially suited for learning in such sparse domains.

5.1.1 Representing Partial Models

When learning models with many variables, small data sets are not sufficiently informative to significantly determine that a single model is the "right" one. Instead, many different networks should be considered as reasonable explanations of the given data. From a Bayesian perspective, we say that the posterior probability over models is not dominated by a single model (or equivalence class of models).

One potential approach to deal with this problem is to find all the networks that receive high posterior score. Such an approach is outlined by Madigan and Raftery (1994). Unfortunately, due to the combinatoric aspect of networks the set of "high posterior" networks can be huge (i.e., exponential in the number of variables). Thus, in a domain such gene expression with many variables and diffused posterior we cannot hope to explicitly list all the networks that are plausible given the data.

Our solution is as follows. We attempt to identify properties of network that might be of interest. For example, are X and Y "close" neighbors in the network. We call such properties *features*. We then try to estimate the posterior probability of features given the data. Since each feature f is either present or not present in a given network $\mathcal{G}$, we can write:

$$P(f \mid \mathcal{G}) = f(\mathcal{G}) = \begin{cases} 1 & \text{if } f \text{ is a feature of } \mathcal{G} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

and the posterior probability of a feature f given the data is:

$$P(f \mid D) = \sum_{\mathcal{G}} f(\mathcal{G}) P(\mathcal{G} \mid D). \tag{5.2}$$

Of course, exact computation of such a posterior probability is as hard as processing all networks with high posterior. However, as we shall see below, we can estimate these posteriors by finding representative networks. Since each feature is a binary attribute, this estimation is fairly robust even from a small set of networks (assuming that they are an unbiased sample from the posterior).

Before we examine the issue of estimating the posterior in such features, we briefly discuss two classes of features involving pairs of variables. While in this chapter we handle only pairwise features, it is clear that this type of analysis is not restricted to them. Pe'er et al. (2001) extends this type of analysis to more feature types, including whole subnetworks.

The first type of feature is *Markov relations*: Is Y in the *Markov blanket* of X? The Markov blanket of X is the minimal set of variables that *shield* X from the rest of the variables in the model. More precisely, X given its Markov blanket is independent from the remaining variables in the network. It is easy to check that this relation is symmetric: Y is in X's Markov blanket if and

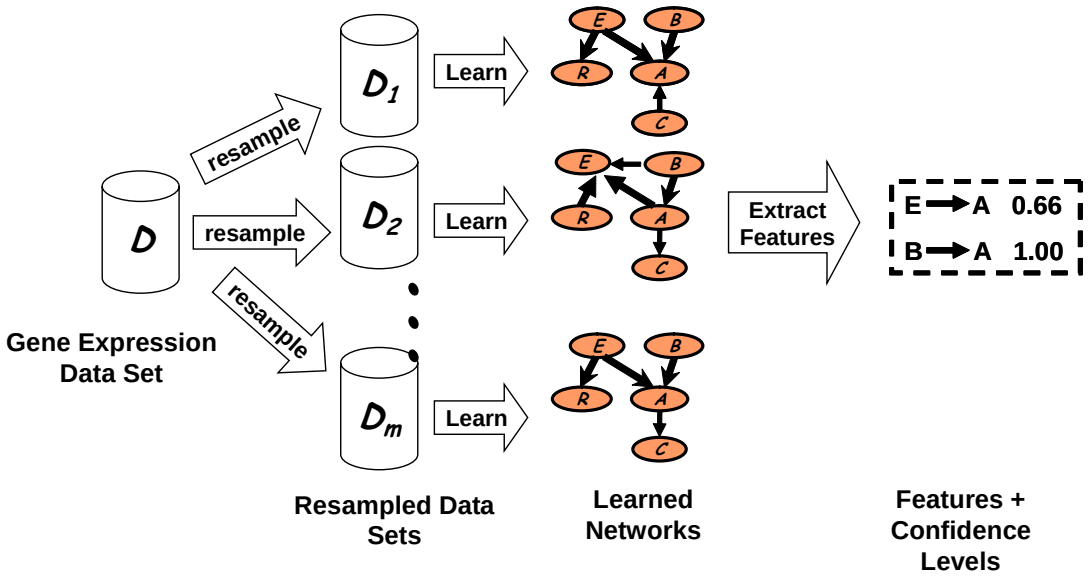


Figure 5.1: Suggested flow for analysis of gene expression data.

only if there is either an edge between them, or both are parents of another variable (Pearl, 1988). In the context of gene expression analysis, a Markov relation indicates that the two genes are related in some joint biological interaction or process. Note that two variables in a Markov relation are directly linked in the sense that no variable *in the model* mediates the dependence between them. It remains possible that an unobserved variable (e.g., protein activation) is an intermediate in their interaction.

The second type of features is *order relations*: Is X an ancestor of Y in all the networks of a given equivalence class? That is, does the given PDAG contain a path from X to Y in which all the edges are directed? This type of feature does not involve only a close neighborhood, but rather captures a global property. Recall that under the assumptions discussed inSection 2.5, learning that X is an ancestor of Y would imply that X is a cause of Y. However, these assumptions are quite strong (in particular the assumption of no latent common causes) and thus do not necessarily hold in the context of expression data. Thus, we view such a relation as an indication, rather than evidence, that X might be a causal ancestor of Y.

5.1.2 Estimating Statistical Confidence in Features

We now face the following problem: To what extent does the data support a given feature? More precisely, we want to estimate the posterior of features as defined in (5.2). Ideally, we would like to sample networks from the posterior and use the sampled networks to estimate this quantity. Unfortunately sampling from the posterior is hard problem. The general approach to this problem is to build a *Markov Chain Monte Carlo* (MCMC) sampling procedure (Madigan and York, 1995) (see (Gilks et al., 1996) for a general introduction to MCMC sampling). However, it is not clear

how these methods scale up for large domain.

Although recent developments in MCMC methods (such as Friedman and Koller, 2000) show promise for scaling up, we choose here to use an alternative method as a "poor man's" version of Bayesian analysis. An effective, and relatively simple, approach for estimating confidence is the *bootstrap* method (Efron and Tibshirani, 1993). The main idea behind the bootstrap is simple. We generate "perturbed" versions of our original data set, and learn from them. In this way we collect many networks, all of which are fairly reasonable models of the data. These networks reflect the effect of small perturbations to the data on the learning process.

In our context, we use the bootstrap as follows:

- For $i = 1 \dots m$.
  - Construct a dataset D_i by sampling, with replacement, N instances from D.
  - Apply the learning procedure on D_i to induce a network structure $\mathcal{G}_i$.
- For each feature f of interest calculate

$$\text{conf}(f) = \frac{1}{m} \sum_{i=1}^{m} f(\mathcal{G}_i)$$

  where $f(\mathcal{G})$ is the indicator function defined in Eq. (5.1).

We refer the reader to Friedman et al. (1999) for more details, as well as large-scale simulation experiments with this method. These simulation experiments show that features induced with high confidence are rarely false positives, even in cases where the data sets are small compared to the system being learned. This bootstrap procedure appears especially robust for the Markov and order features described in section 5.1.1. In addition, simulation studies by Friedman and Koller (2000) show that although the confidence values computed by the bootstrap are not equal to the Bayesian posterior they correlate well with estimates of the Bayesian posterior for features.

Figure 5.1 shows the suggested bootstrap flow for analysis of gene expression data on a sample domain consisting of 5 genes. The two feature examples given are *edge features*, meaning simply the existence of a specific edge. In this example, an edge from gene E to gene A is present in 0.66 of the learned networks, and so is assigned a confidence level of 0.66. An edge from B to A is present in all learned networks, and therefore is assigned the maximal confidence level. In a similar manner, confidence levels can be assigned to Markov and order features we described above.

5.1.3 Local Representations and Learning Algorithms

In order to specify a Bayesian network model, we need to choose the type of the local probability models we learn. In this chapter, we consider two of the simplest approaches:

- **Multinomial model**. In this model we treat each variable as discrete and learn a multinomial distribution that describes the probability of each possible state of the child variable given the state of its parents, as described in Section 2.2.1.
- **Linear Gaussian model**. In this model we learn a linear regression model for the child variable given its parents, as described in Section 2.2.2.

These models were chosen since their posterior and marginal distributions can be efficiently calculated in closed form. For the multinomial model we use the Dirichlet parameter prior, and accordingly the BDe structure score. For the linear Gaussian model we use normal-Wishart priors and the BGe structure score (see Section 2.4.3 for details). We note here several properties of these models which are important for our task. First, the priors contain a notion of a *prior network*, which reflects our prior belief on the structure and parameters of the underlying distribution of the domain, and a notion of an *effective sample size*, which reflects how strong is our belief in the prior network. Intuitively, setting the effective sample size to K, is equivalent to having seen K samples from the distribution defined by the prior network. Second, when learning from complete data (i.e. with no missing values) these scores are *structure equivalent*, i.e., if $\mathcal{G}$ and $\mathcal{G}$ are equivalent graphs they are guaranteed to have the same score. Third, these network scores are *decomposable* to local family scores. Finally, these local family scores can be computed using a closed form equation. (see Section 2.4.3 for more details).

To apply the multinomial model we need to discretize the gene expression values. We choose to discretize these values into three categories: *under-expressed* (-1), *normal* (0), and *over-expressed* 1, depending on whether the expression rate is significantly lower than, similar to, or greater than control, respectively. The control expression level of a gene can be either determined experimentally (as in the methods of DeRisi et al., 1997), or it can be set as the average expression level of the gene across experiments. We discretize by setting a threshold to the ratio between measured expression and control. In our experiments we choose a threshold value of 0.5 in logarithmic (base 2) scale. Thus, values with ratio to control lower than $2^{-0.5}$ are considered under-expressed, and values higher than $2^{0.5}$ are considered over-expressed.

Each of the two models has benefits and drawbacks. On one hand, it is clear that by discretizing the measured expression levels we are loosing information. The linear-Gaussian model does not suffer from the information loss caused by discretization. On the other hand, the linear-Gaussian model can only detect dependencies that are close to linear. In particular, it is not likely to discover combinatorial effects (e.g., a gene is over expressed only if several genes are jointly over expressed, but not if at least one of them is not over expressed). The multinomial model is more flexible and can capture such dependencies.

In section 2.4.2, we formulated learning Bayesian network structure as an optimization problem in the space of directed acyclic graphs. As we showed, the number of such graphs is super-exponential in the number of variables. Since in our domain we consider hundreds of variables, we must deal with an extremely large search space. Therefore, we need to use efficient search algorithms. In this chapter we use the "Sparse Candidate" algorithm described in Chapter 4. For the

candidate selection step we use the score measure M_{Score}, which proved successful on synthetic domains.

5.2 Application to Cell Cycle Expression Patterns

We applied our approach to the data of Spellman et al. (1998). This data set contains 76 gene expression measurements of the mRNA levels of 6177 *S. cerevisiae* ORFs. These experiments measure six time series under different cell cycle synchronization methods. Spellman et al. identified 800 genes whose expression varied over the different cell-cycle stages.

In learning from this data, we treat each measurement as an independent sample from a distribution, and do not take into account the temporal aspect of the measurement. Since it is clear that the cell cycle process is of temporal nature, we compensate by introducing an additional variable denoting the cell cycle phase. This variable is forced to be a root in all the networks learned. Its presence allows to model dependency of expression levels on the current cell cycle phase.

We used the Sparse Candidate algorithm with a 200-fold bootstrap in the learning process. We performed two experiments, one with the discrete multinomial distribution, the other with the linear Gaussian distribution. The learned features show that we can recover intricate structure even from such small data sets. It is important to note that our learning algorithm uses *no prior biological knowledge nor constraints*. All learned networks and relations are based solely on the information conveyed in the measurements themselves. Figure 5.6 at the end of this chapter illustrates the graphical display of some results from this analysis.

5.2.1 Robustness Analysis

We first performed a number of tests to analyze the statistical significance and robustness of our procedure. Some of these tests were carried on a smaller data set with 250 genes for computational reasons.

To test the credibility of our confidence assessment, we created a random data set by randomly permuting the order of the experiments independently for each gene. Thus for each gene the order was random, but the composition of the series remained unchanged. In such a data set, genes are independent of each other, and thus we do not expect to find "real" features. As expected, both order and Markov relations in the random data set have significantly lower confidence. We compare the distribution of confidence estimates between the original data set and the randomized set in Figure 5.2. Clearly, the distribution of confidence estimates in the original data set have a longer and heavier tail in the high confidence region. In the linear-Gaussian model we see that random data does not generate any feature with confidence above 0.3. The multinomial model is more expressive, and thus susceptible to overfitting. For this model, we see a smaller gap between the two distributions. Nonetheless, randomized data does not generate any feature with confidence above 0.8, which leads us to believe that most features that are learned in the original data set with

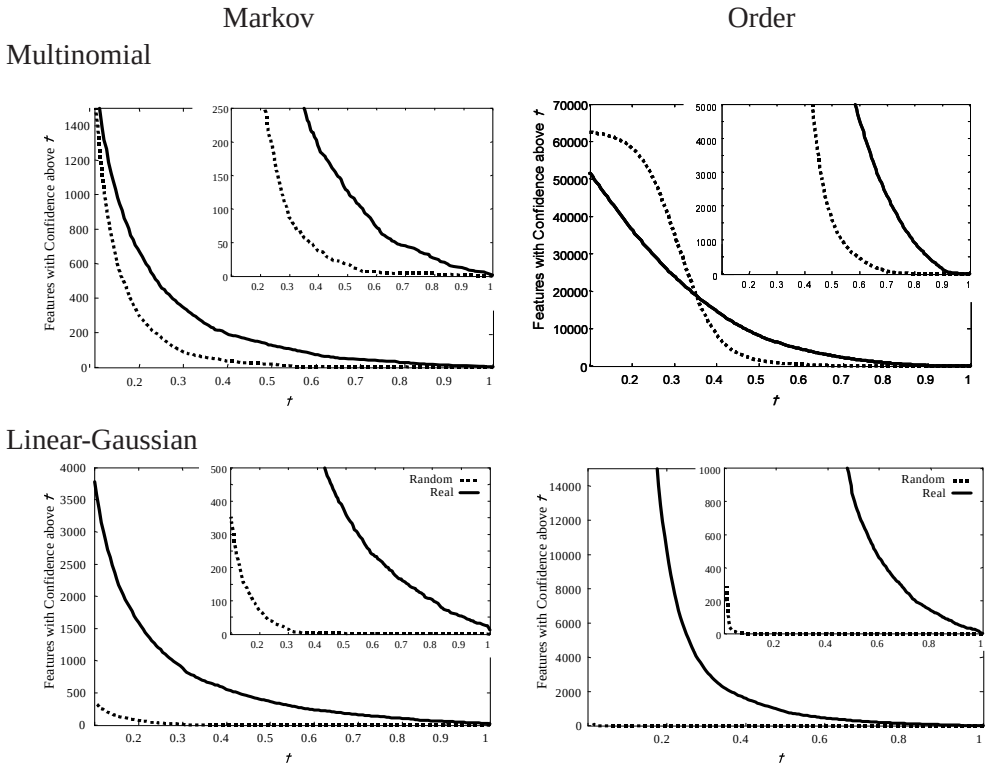


Figure 5.2: Plots of abundance of features with different confidence levels for the cell cycle data set (solid line), and the randomized data set (dotted line). The x-axis denotes the confidence threshold, and the y-axis denotes the number of features with confidence equal or higher than the corresponding x-value. The graphs on the left column show Markov features, and the ones on the right column show Order features. The top row describes features found using the multinomial model, and the bottom row describes features found by the linear-Gaussian model. Inset in each graph is plot of the tail of the distribution.

such confidence are not an artifact of the bootstrap estimation.

Since the analysis was not performed on the whole *S. cerevisiae* genome, one concern is that the results are highly sensitive to the number and choice of genes. We therefore tested the robustness of our analysis to the addition of more genes, comparing the confidence of the learned features between the 800 gene dataset and a smaller 250 gene data set that contains genes appearing in eight major clusters described by Spellman et al.. Figure 5.3 compares feature confidence in the analysis of the two datasets for the multinomial model. As we can see, there is a strong correlation between confidence levels of the features between the two data sets. The comparison for the linear-Gaussian model gives similar results.

A crucial choice for the multinomial experiment is the threshold level used for discretization of the expression levels. It is clear that by setting a different threshold, we would get different discrete expression patterns. Thus, it is important to test the robustness and sensitivity of the high confidence features to the choice of this threshold. This was tested by repeating the experiments

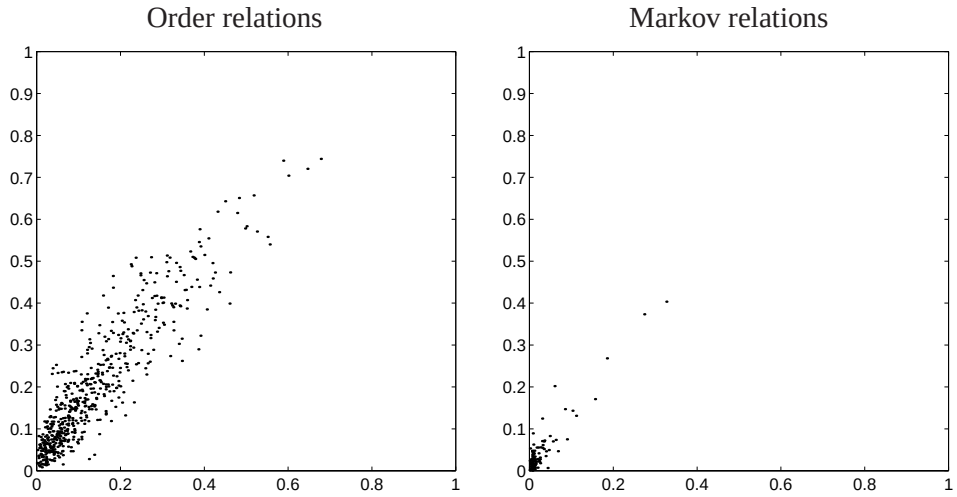


Figure 5.3: Comparison of confidence levels obtained in two datasets differing in the number of genes, on the multinomial experiment. Each relation is shown as a point, with the x-coordinate being its confidence in the the 250 genes data set and the y-coordinate the confidence in the 800 genes data set. The left figure shows order relation features, and the right figure shows Markov relation features.

using different thresholds. The comparison shows a definite correlation between the confidence estimates of features obtained at different discretization thresholds (graphs not shown). Obviously, this linear correlation gets weaker for larger threshold differences. We also note that order relations are much more robust to changes in the threshold than Markov relations.

A valid criticism of our discretization method is that it penalizes genes whose natural range of variation is small: since we use a fixed threshold, we would not detect changes in such genes. A possible way to avoid this problem is to *normalize* the expression of genes in the data. That is, we rescale the expression level of each gene, so that the relative expression level has the same mean and variance for all genes. We note that analysis methods that use *Pearson correlation* to compare genes, such as (Ben-Dor et al., 1999; Eisen et al., 1998b), implicitly perform such a normalization.[1] When we discretize a normalized dataset, we are essentially rescaling the discretization factor differently for each gene, depending on its variance in the data. We tried this approach with several discretization levels, and got results comparable to our original discretization method. The 20 top Markov relations highlighted by this method were a bit different, but interesting and biologically sensible in their own right. The order relations were again more robust to the change of methods and discretization thresholds. A possible reason is that order relations depend on the network structure in a global manner, and thus can remain intact even after many local changes to the structure. The

[1] An undesired effect of such a normalization is the amplification of measurement noise. If a gene has fixed expression levels across samples, we expect the variance in measured expression levels to be noise either in the experimental conditions or the measurements . When we normalize the expression levels of genes, we loose the distinction between such noise and true (i.e., significant) changes in expression levels. In the Spellman et al. dataset we can safely assume this effect will not be too grave, since we only focus on genes that display significant changes across experiments.

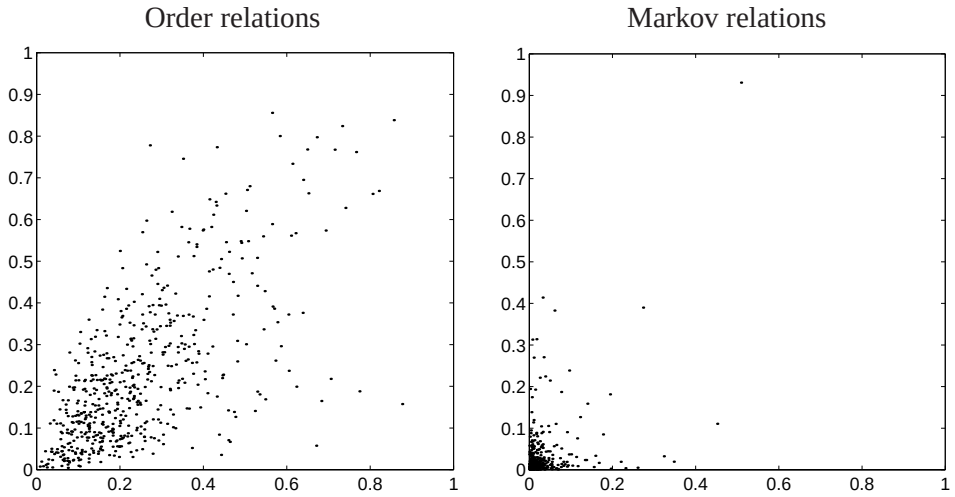


Figure 5.4: Comparison of of confidence levels between the multinomial experiment and the linear-Gaussian experiment. Each relation is shown as a point, with the x-coordinate being its confidence in the multinomial experiment, and the y-coordinate its confidence in the linear-Gaussian experiment. *left* - order relation features; *right* - Markov relation features.

Markov relation, being a local one, is more easily disrupted. Since the graphs learned are extremely sparse, each discretization method "highlights" different signals in the data, which are reflected in the Markov relations learned.

A similar picture arises when we compare the results of the multinomial experiment to those of the linear-Gaussian experiment (Figure 5.4). In this case there is virtually no correlation between the Markov relations found by the two methods, while the order relations show some correlation. This supports our assumption that the two methods highlight different types of connections between genes.

Finally, we consider the effect of the choice of prior on the learned features. It is important to ensure that the learned features are not simply artifacts of the chosen prior. To test this, we repeated the multinomial experiment with different values of K, the effective sample size, and compared the learned confidence levels to those learned with the default value used for K, which was 5. This was done using the 250 gene data set and discretization level of 0.5. The results of these comparisons are shown in Figure 5.5. As can be seen, the confidence levels obtained with K value of 1 correlate very well with those obtained with the default K, while when setting K to 20 the correlation is weaker. This suggests that both 1 and 5 are low enough values compared to the data set size of 76, making the prior's affect on the results weak. An effective sample size of 20 is high enough to make the prior's effect noticeable. Another aspect of the prior is the prior network used. In all the experiments reported here we used the empty network with uniform distribution parameters as the prior network, encoding zero prior knowledge about the domain. As our prior is non-informative, keeping down its effect is desired. It is expected that once we use more informative priors (by

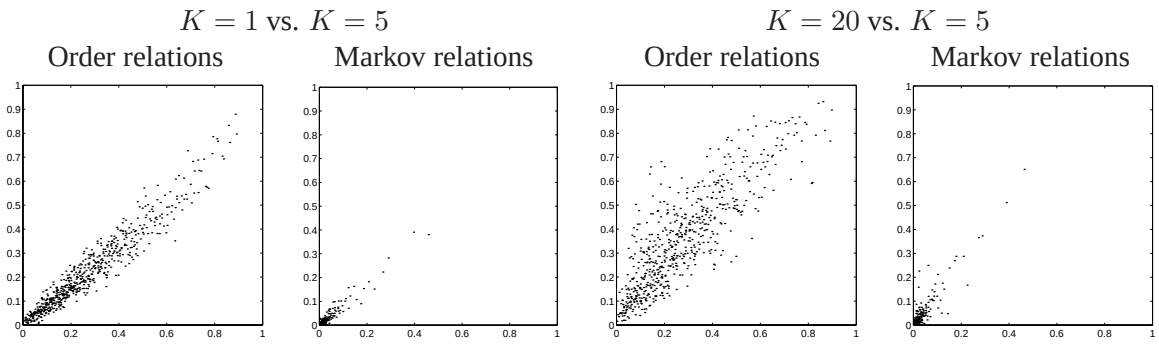


Figure 5.5: Comparison of confidence levels between runs using different parameter priors. The difference between priors is in the effective sample size, K. Each relation is shown as a point, with the x-coordinate being its confidence in a run with $K = 5$, and the y-coordinate its confidence in a run with $K = 1$ (left figures) or $K = 20$ (right figures). Both runs are on the 250 gene set, using discretization with threshold level 0.5.

incorporating biological knowledge, for example) and stronger effective sample sizes, the obtained results will be more biased towards our prior beliefs.

In summary, although many of the results we report below (especially order relations) are stable across the different experiments discussed in the previous paragraph, it is clear that our analysis is sensitive to the choice of local model, and in the case of the multinomial model, to the discretization method. It is probably less sensitive to the choice of prior, as long as the effective sample size is small compared to the data set size. In all the methods we tried, our analysis found interesting relationships in the data. Thus, one challenge is to find alternative methods that can recover all these relationships in one analysis. One option, which we develop in the next chapter, is to use a more biologically realistic type of CPD, that would circumvent the need for discretization on one hand, and allow realistic nonlinear dependency relations on the other.

5.2.2 Biological Analysis

We believe that the results of this analysis can be indicative of biological phenomena in the data. This is confirmed by our ability to predict sensible relations between genes of known function. We now examine several consequences that we have learned from the data. We consider, in turn, the order relations and Markov relations found by our analysis.

Order Relations

The most striking feature of the high confidence order relations, is the existence of *dominant genes*. Out of all 800 genes only few seem to dominate the order (i.e., appear before many genes). The intuition is that these genes are indicative of potential causal sources of the cell-cycle process. Let $C_o(X, Y)$ denote the confidence in X being ancestor of Y. We define the *dominance score* of X

Table 5.1: List of dominant genes in the ordering relations. Included are the top 10 dominant genes for each experiments.

Gene/ORF	Score in Experiment		Notes
	Multinomial	Gaussian	
MCD1	550	525	Mitotic Chromosome Determinant,null mutant is inviable
MSH6	292	508	Required for mismatch repair in mitosis and meiosis
CSI2	444	497	cell wall maintenance, chitin synthesis
CLN2	497	454	Role in cell cycle START, null mutant exhibits G1 arrest
YLR183C	551	448	Contains forkheaded associated domain, thus possibly nuclear
RFA2	456	423	Involved in nucleotide excision repair, null mutant is inviable
RSR1	352	395	GTP-binding protein of the RAS family involved in bud site selection
CDC45	-	394	Required for initiation of chromosomal replication, null mutant lethal
RAD53	60	383	Cell cycle control, checkpoint function, null mutant lethal
CDC5	209	353	Cell cycle control, required for exit from mitosis, null mutant lethal
POL30	376	321	Required for DNA replication and repair, null mutant is inviable
YOX1	400	291	Homeodomain protein
SRO4	463	239	Involved in cellular polarization during budding
CLN1	324	-	Role in cell cycle START, null mutant exhibits G1 arrest
YBR089W	298	-	

as $\sum_{Y,C_o(X,Y)>t} C_o(X,Y)^k$, using the constant k for rewarding high confidence features and the threshold t to discard low confidence ones. These dominant genes are robust to parameter selection for both t, k, the discretization cutoff of section 5.1.3 and the local probability model used. A list of the highest scoring dominating genes for both experiments appears in Table 5.1.

Inspection of the list of dominant genes reveals quite a few interesting features. Among them are genes directly involved in initiation of the cell-cycle and its control. For example, CLN1, CLN2, CDC5 and RAD53 whose functional relation has been established (Cvrckova and Nasmyth, 1993; Drebot et al., 1993). The genes MCD1, RFA2, CDC45, RAD53, CDC5 and POL30 were found to be essential (Guacci et al., 1997). These are clearly key genes in essential cell functions. Some of them are components of pre-replication complexes(CDC45,POL30). Others (like RFA2,POL30 and MSH6) are involved in DNA repair. It is known that DNA repair is associated with transcription initiation, and DNA areas which are more active in transcription, are also repaired more frequently (McGregor, 1999; Tornaletti and Hanawalt, 1999). Furthermore, a cell cycle control mechanism causes an abort when the DNA has been improperly replicated (Eisen et al., 1998a).

Most of the dominant genes encode nuclear proteins, and some of the unknown genes are also potentially nuclear: (e.g., YLR183C contains a forkhead-associated domain which is found almost entirely among nuclear proteins). A few non nuclear dominant genes are localized in the cytoplasm membrane (SRO4 and RSR1). These are involved in the budding and sporulation process which have an important role in the cell-cycle. RSR1 belongs to the RAS family of proteins, which are known as initiators of signal transduction cascades in the cell.

Table 5.2: List of top Markov relations, multinomial experiment.

Confidence	Gene 1	Gene 2	Notes
1.0	YKL163W-PIR3	YKL164C-PIR1	Close locality on chromosome
0.985	PRY2	YKR012C	Close locality on chromosome
0.985	MCD1	MSH6	Both bind to DNA during mitosis
0.98	PHO11	PHO12	Both nearly identical acid phosphatases
0.975	HHT1	HTB1	Both are Histones
0.97	HTB2	HTA1	Both are Histones
0.94	YNL057W	YNL058C	Close locality on chromosome
0.94	YHR143W	CTS1	Homolog to EGT2 cell wall control, both involved in Cytokinesis
0.92	YOR263C	YOR264W	Close locality on chromosome
0.91	YGR086	SIC1	Homolog to mammalian nuclear ran protein, both involved in nuclear function
0.9	FAR1	ASH1	Both part of a mating type switch, **expression uncorrelated**
0.89	CLN2	SVS1	Function of SVS1 unknown
0.88	YDR033W	NCE2	Homolog to transmembrame proteins suggest both involved in protein secretion
0.86	STE2	MFA2	A mating factor and receptor
0.85	HHF1	HHF2	Both are Histones
0.85	MET10	ECM17	Both are sulfite reductases
0.85	CDC9	RAD27	Both participate in Okazaki fragment processing

Markov Relations

We begin with an analysis of the Markov relations in the multinomial experiment. Inspection of the top Markov relations reveals that most are functionally related. A list of the top scoring relations can be found in Table 5.2. Among these, all involving two known genes make sense biologically. When one of the ORFs is unknown careful searches using Psi-Blast (Altschul et al., 1997), Pfam (Sonnhammer et al., 1998) and Protomap (Yona et al., 1998) can reveal firm homologies to proteins functionally related to the other gene in the pair. For example YHR143W, which is paired to the endochitinase CTS1, is related to EGT2 - a cell wall maintenance protein. Several of the unknown pairs are physically adjacent on the chromosome, and thus presumably regulated by the same mechanism (see Blumenthal, 1998), although special care should be taken for pairs whose chromosomal location overlap on complementary strands, since in these cases we might see an artifact resulting from cross-hybridization. Such an analysis raises the number of biologically sensible pairs to nineteen out of the twenty top relations.

There are some interesting Markov relations found that are beyond the limitations of clustering techniques. Among the high confidence Markov relations, one can find examples of conditional independence, i.e., a group of highly correlated genes whose correlation can be explained within our network structure. One such example involves the genes CLN2,RNR3,SVS1,SRO4 and RAD51.

Table 5.3: List of top Markov relations, Gaussian experiment. (The table skips over 5 additional pairs with close locality.)

Confidence	Gene 1	Gene 2	Notes
1.0	YOR263C	YOR264W	Close locality on chromosome
1.0	CDC46	YOR066W	YOR066W is totally unknown.
1.0	CDC45	SPH1	No suggestion for immediate link.
1.0	SHM2	GCV2	SHM2 interconverts glycine, GCV2 is regulated by glycine
1.0	MET3	ECM17	MET3 required to convert sulfate to sulfide, ECM17 sulfite reductase
1.0	YJL194W-CDC6	YJL195C	Close locality on chromosome
1.0	YGR151C	YGR152C	Close locality on chromosome
1.0	YGR151C	YGR152C-RSR1	Close locality on chromosome
1.0	STE2	MFA2	A mating factor and receptor
1.0	YDL037C	YDL039C	Both homologs to mucin proteins
1.0	YCL040W-GLK1	WCL042C	Close locality on chromosome
1.0	HTA1	HTA2	two physically linked histones
...			
0.99	HHF2	HHT2	both histones
0.99	YHR143W	CTS1	Homolog to EGT2 cell wall control, both involved in Cytokinesis
0.99	ARO9	DIP5	DIP5 transports glutamate which regulates ARO9
0.975	SRO4	YOL007C	Both proteins are involved in cell wall regulation at the plasma membrane.

Their expression is correlated, and in Spellman et al. (1998) they all appear in the same cluster. In our network CLN2 is with high confidence a parent of each of the other 4 genes, while no links are found between them (see Figure 5.6). This suits biological knowledge: CLN2 is a central and early cell cycle control, while there is no clear biological relationship between the others. Some of the other Markov relations are inter-cluster, pairing genes with low correlation in their expression. One such regulatory link is FAR1-ASH1: both proteins are known to participate in a mating type switch. The correlation of their expression patterns is low and Spellman et al. cluster them into different clusters. When looking further down the list for pairs whose Markov relation confidence is high relative to their correlation, interesting pairs surface. For example SAG1 and MF-ALPHA-1, a match between the factor that induces the mating process and an essential protein that participates in the mating process. Another match is LAC1 and YNL300W. LAC1 is a GPI transport protein and YNL300W is most likely modified by GPI (based on sequence homology).

The Markov relations from the Gaussian experiment are summarized in Table 5.3. Since the Gaussian model focuses on highly correlated genes, most of the high scoring genes are tightly correlated. When we checked the DNA sequence of pairs of physically adjacent genes at the top of Table 5.3, we found that there is significant overlap. This suggests that these correlations are spurious and due to *cross hybridization*. Thus, we ignore the relations with the highest score. However,

in spite of this technical problem, few of the pairs with a confidence of > 0.8 can be discarded as biologically false.

Some of the relations are robust and also appear in the multinomial experiment (e.g. STE2-MFA2, CST1-YHR143W). Most interesting are the genes linked through regulation. These include: SHM2 which converts glycine that regulates GCV2 and DIP5 which transports glutamate which regulates ARO9. Some pairs participate in the same metabolic process, such as: CTS1-YHR143 and SRO4-YOL007C all which participate in cell wall regulation. Other interesting high confidence (> 0.9) examples are: OLE1-FAA4 linked through fatty acid metabolism, STE2-AGA2 linked through the mating process and KIP3-MSB1, both playing a role in polarity establishment.

5.3 Discussion and Future Work

In this chapter we presented a new approach for analyzing gene expression data that builds on the theory and algorithms for learning Bayesian networks. We described how to apply these techniques to gene expression data. The approach builds on two techniques that were motivated by the challenges posed by this domain: a novel search algorithm (the "Sparse Candidate" algorithm) and an approach for estimating statistical confidence (Friedman et al., 1999). We applied our methods to real expression data of Spellman et al. (1998). Although, we did not use any prior knowledge, we managed to extract many biologically plausible conclusions from this analysis.

Our approach is quite different than the clustering approach used by (Alon et al., 1999; Ben-Dor et al., 1999; Eisen et al., 1998b; Michaels and et al., 1998; Spellman et al., 1998), in that it attempts to learn a much richer structure from the data. Our methods are capable of discovering causal relationships, interactions between genes other than positive correlation, and finer intra-cluster structure. The biological motivation of our approach is similar to work on inducing *genetic networks* from data (Akutsu et al., 1998; Chen et al., 1999a; Somogyi et al., 1996; Weaver et al., 1999). There are two key differences: First, the models we learn have probabilistic semantics. This better fits the stochastic nature of both the biological processes and noisy experiments. Second, our focus is on extracting features that are pronounced in the data, in contrast to current genetic network approaches that attempt to find a single model that explains the data.

The line of research presented here was extended in several works. In Pe'er et al. (2001), the notion of a feature was extended beyond pairwise relations, to include separators, hubs, and finally whole subnetworks. Additionally, the method is extended to account for learning from intervention experiments (i.e. knock-out or over-expression of a certain gene). In Pe'er et al. (2002) the model learning is focused on finding regulatory interactions by limiting the set of possible parents to proteins known to be involved in regulation, and by choosing an optimal small set of such regulators. Pe'er (2003) surveys and extends on both these works, and presents results on additional data sets.

A lot of progress can still be done in this line of work. For example, the use of prior biological knowledge through the prior network mechanism is yet to be explored. This prior network can also be built on other, new forms of data, such as DNA binding location data. Another direction for

improvement is developing the theory and algorithms for estimating confidence levels. Finally, a very promising direction involves the interaction between such learning methods and intervention experiments. Methods that, following learning from an initial data set, can suggest the most informative set of intervention experiments, can lead to a truly synergistic interaction between theory and experiment.

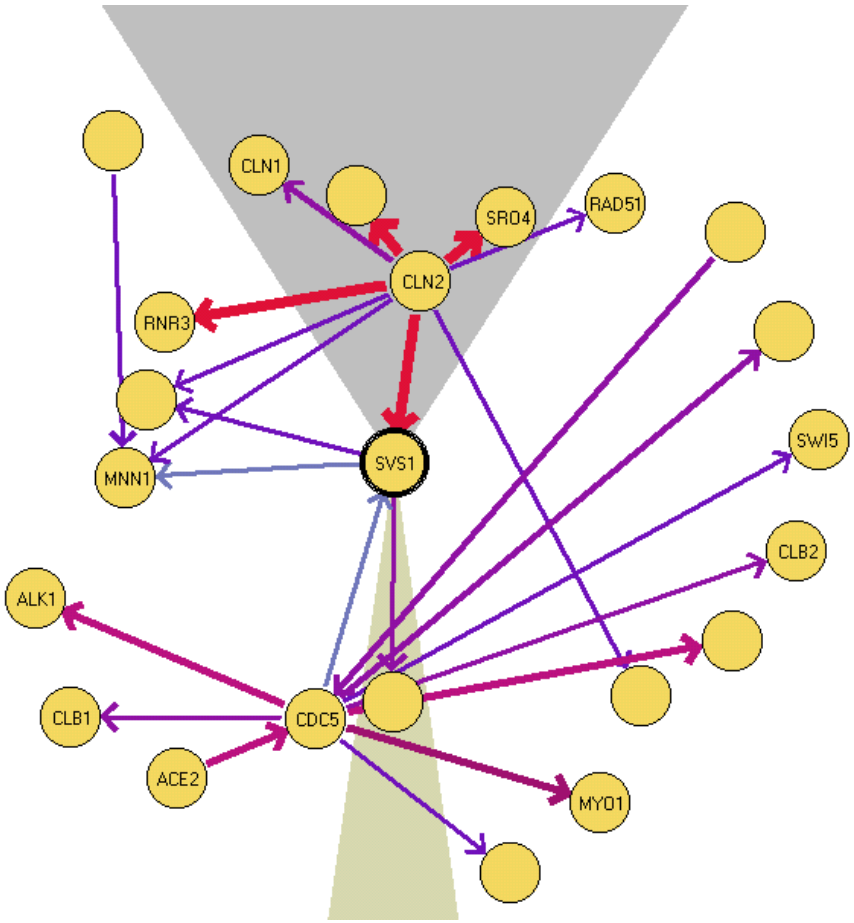


Figure 5.6: An example of the graphical display of Markov features. This graph shows a "local map" for the gene SVS1. The width (and color) of edges corresponds to the computed confidence level. An edge is directed if there is a sufficiently high confidence in the order between the genes connected by that edge. This local map shows that CLN2 separates SVS1 from several other genes. Although there is a strong connection between CLN2 to all these genes, there are no other edges connecting them. This indicates that, with high confidence, these genes are conditionally independent given the expression level of CLN2.

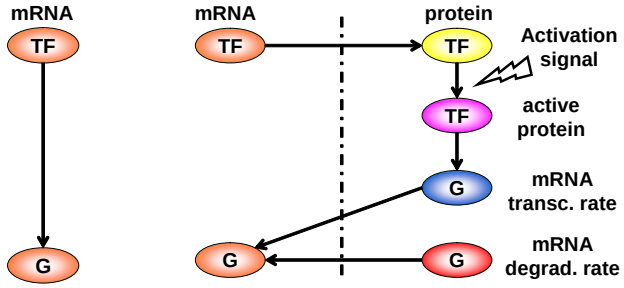
Chapter 6

Realistic Models of Regulatory Networks

In the previous chapter we have presented the first graphical model based approach to modeling and learning of gene regulation networks. In recent years many other methods have been developed to reconstruct such networks from high-throughput data, including genomic sequences, expression profiles and transcription factor location assays (Ong et al., 2002; Pe'er et al., 2001; Segal et al., 2002; Simon et al., 2001; Spellman et al., 1998; Tavazoie et al., 1999).However, these methods are based on coarse grained qualitative models, and cannot provide a realistic and quantitative view of regulatory systems. Recent studies (Guet et al., 2002; Kitano, 2002) indicate that network function depends on both qualitative and quantitative aspects of network organization. For example, Guet et al. (2002) show how differences in quantitative reaction rates have drastic effects on the function of circuits with identical qualitative properties such as connectivity and logic.

Furthermore, many of the network modeling methods, (including the one presented inChapter 5) attempt to learn regulatory connections by modeling dependencies between the mRNA levels of a transcription factor and its target gene (Figure 6.1(a)). Doing this they ignore a whole chain of regulation events, including translation of the regulator protein, activation of that protein, binding of that protein to the promoter region of the target gene and initiating transcription at a certain rate. The mRNA levels of the target gene are determined by this rate as well as the degradation rate of the mRNA molecules (Figure 6.1(b)). Most of these processes are not measured in typical high-throughput experiments, and are therefore hidden from us. In promoter activity experiments, the transcription rate can be deduced indirectly from the measurements, but the other processes are still hidden.

The problem with the simple model is that it fails in any case there is active regulation in any of the hidden stages. The most common case is the post-translational modification stage, where a regulator can be dynamically induced or inhibited by processes like phosphorilation. In such cases we might see no variability at all at the mRNA level of the regulator, and therefore no regulatory connection can be learned for it based on expression data alone. As we see inFigure 6.1(b), the closest quantities on the regulation path representing the two genes are the *active protein level* of the transcription factor, and the *mRNA transcription rate* of the target gene. While the latter is



(a) Simplified path (b) Realistic path

Figure 6.1: Two possible models of a regulatory connection between a transcription factor TF and a target gene G. (a) A simplified path (b) A more realistic path; The dashed line separates between the observed (left) and hidden (right) processes of regulation in typical microarray experiments.

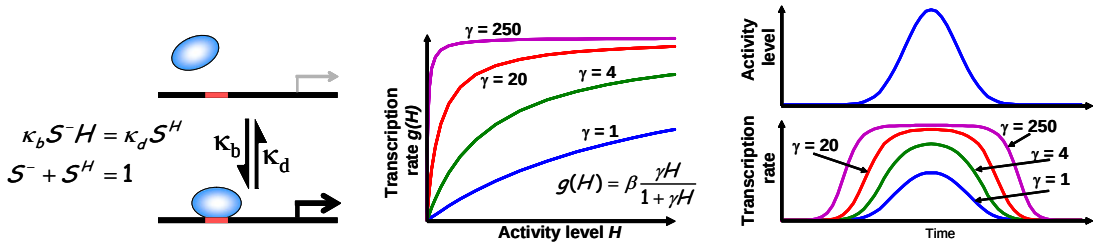
observed in some experiments, and can be derived in others, the former is essentially hidden from us.

In this chapter, we present a novel framework for the reconstruction of quantitative, realistic, fine-grained, dynamical models of gene regulatory networks. This method tries to answer the two weaknesses of other methods we pointed out. First, it models the connection between the hidden activity level of regulators, and the observed (or derived) mRNA transcription rates of target genes. Activity levels are modeled as unobserved variables, that indirectly encompass upstream regulatory events, without directly modeling these events. In particular, unlike previous work, we do not use the expression levels of regulators, and can thus identify the results of post-transcriptional events. Second, rather than using a simple qualitative interaction model, we build our regulation model from basic principles of the biochemical interactions.

The remainder of this chapter is organized as follows. In Section 6.1 we develop a kinematic model for the dependence of a target gene's transcription rate on its regulators' active protein levels. In Section 6.2 we incorporate this regulation model into a DBN-based model for transcription rates of complete regulation systems, composed of several target genes and regulators. We discuss methods for learning the parameters and structure for these models, and also methods to derive transcription rates from different expression measurement methods. In Section 6.3 we evaluate our framework on small systems, controlled by one or two regulators, both from synthetic and from real-life data. In Section 6.4 we apply the method to a large regulatory system in yeast. We show how we can recover both regulator activity profiles and kinetic parameters for networks of known architecture, as well as successfully learn a complex regulatory network *ab initio*.

6.1 Transcriptional Regulation Model

To develop a quantitative realistic probabilistic model of gene regulatory networks, we start by choosing a realistic model for regulator-target dependencies. First, we derive a kinematic model



(a) Two promoter states (b) Regulation as function of H (c) Temporal behavior

Figure 6.2: A kinematic model of transcription regulation by a single activator. (a) An active regulator protein, H, may bind to and disassociate from a target gene's promoter, with rate constants K_b and K_d, respectively. In a population of cells, fractions S^- and S^H of cells have free and bound promoters, respectively and satisfy the steady-state reaction equations. The bound gene is transcribed with rate $g(H) = \beta S^H$. (b) The regulation function, describing the transcription rate as a function of the active regulator concentration H, is in the Michaelis-Menten form. The transcription rate is a non-linear function of the activity level of H that depends on $\gamma = K_b/K_d$; In particular, at high levels of H, the transcription rate saturates. (c) Temporal behaviour of a single activator and the transcription rates of genes it regulates with different kinematic parameters.

of how the transcription rate of a single gene depends on its regulators. We then consider how to model the behaviour of multiple genes over time, and how to learn the model parameters and the unobserved activity levels of regulators from actual measurements, including transcriptional rates.

Our regulation model is based on a *regulation function* that describes the *transcription rate* of a target gene (number of RNA molecules transcribed per unit of time per cell) as a function of the *concentration of active regulator(s)* (number of proteins in active form in nucleus per cell). We start with the simplest example of a gene regulated by a single activator (Figure 6.2(a)). In this case, the regulation function takes the familiar, non-linear Michaelis-Menten form:

$$g(H : \beta, \gamma) = \beta \frac{\gamma H}{1 + \gamma H} \tag{6.1}$$

where H denotes the concentration of active regulator protein, β is the maximum transcription rate the gene can achieve, and γ is K_b/K_d the ratio of association and disassociation constants. Figure 6.2(b-c) show the behavior of the regulation function for different values of the affinity parameter γ. We note two things: First, the regulation function is non-linear, displaying a saturation effect. Second, the form of the function and the rate of saturation depends strongly on the affinity parameter γ. Figure 6.2(c) shows how target genes with different parameters can display a cascaded activation, such as measured in some known systems (Kalir et al., 2001). For the case of a single repressor, the regulation function takes the form:

$$g(H : \beta, \gamma) = \beta \frac{1}{1 + \gamma H} \tag{6.2}$$

We now develop regulation functions from basic principles of the biochemical reactions involved. We first consider the state equations of those reactions, and then solve the equilibrium equations to obtain the steady state distribution of promoter states in the population. By assigning mean activity levels to each promoter state, we will obtain the resulting mean transcription rate.

6.1.1 Modeling Binding/Disassociation Events with state equations

We begin by considering the biochemical reactions involved in the binding and disassociation of transcription factors from binding sites at the promoter region of the target gene. In the case of a single regulator, the simplest model assumes a single binding site in the target gene's promoter. Figure 6.2(a) illustrates this reaction. A biochemical equation describing this reaction is:

$$S + A \underset{K_d^A}{\overset{K_b^A}{\rightleftharpoons}} SA \tag{6.3}$$

where S is the concentration of free binding sites, A is the concentration of free regulator molecules, and SA is the concentration of sites bound to a regulator. K_b and K_d are the binding and disassociation constants, respectively, of the reaction.

In case there are two regulators, A and B, we must make some modeling choices. Perhaps the simplest choice is modeling a single binding site per factor, where there is neither competition in binding, nor synergism. In this case we consider the two binding reactions *independent*, and the state equations are:

$$\begin{aligned} S + A &\underset{K_d^A}{\overset{K_b^A}{\rightleftharpoons}} SA \\ S + B &\underset{K_d^B}{\overset{K_b^B}{\rightleftharpoons}} SB \end{aligned} \tag{6.4}$$

Here K_b^A and K_d^A are the kinetic constants for regulator A, and K_b^B and K_d^B are the constants related to regulator B. A more realistic picture takes into account *synergism*, meaning difference in binding energies due to the current binding state of the promoter. For example, if regulator A is currently bound, it might facilitate the binding of regulator B to the promoter. If the difference in energies is large, it is sometimes said that A "recruits" B. In this case, the possible transitions in

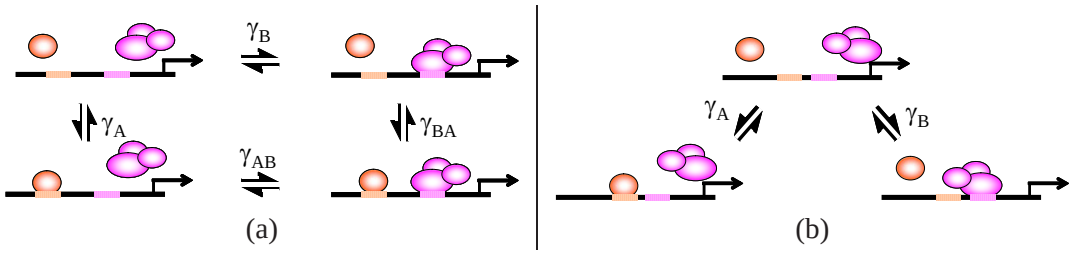


Figure 6.3: Possible promoter states and allowed transitions for a two-regulator promoter. (a) synergistic binding scenario (b) competitive binding scenario. For each reaction, $\gamma_X = K_b^X / K_d^X$.

the promoter state form a cycle, as shown in Figure 6.3(a), and the associated reactions are:

$$\begin{aligned} S + A &\underset{K_d^A}{\overset{K_b^A}{\rightleftharpoons}} SA \\ S + B &\underset{K_d^B}{\overset{K_d^B}{\rightleftharpoons}} SB \\ SA + B &\underset{K_d^{AB}}{\overset{K_b^{AB}}{\rightleftharpoons}} SAB \\ A + SB &\underset{K_d^{BA}}{\overset{K_b^{BA}}{\rightleftharpoons}} SAB \end{aligned} \tag{6.5}$$

Each one of the four possible reactions has its corresponding binding and disassociation constants. Here K_b^{AB} denotes the binding rate constant of B to a promoter where A is already bound. This model generalizes over the independent binding scenario, which is obtained if $\gamma_{AB} = \gamma_B$ and $\gamma_{BA} = \gamma_A$.

Yet a different scenario is one where A and B *compete* on the same site, or that their sites are too close to allow concurrent binding of the two factors, as depicted in Figure 6.3(b). In this case we can describe the reactions as follows:

$$A + SB \underset{K_b^B}{\overset{K_d^B}{\rightleftharpoons}} S + A + B \underset{K_d^A}{\overset{K_b^A}{\rightleftharpoons}} SA + B$$

In all the reactions described above, the rate constants are believed to depend strongly on the nucleotide sequence of the binding site: a high affinity sequence can result in a large difference in free energy between the bound and unbound states, resulting in a high ratio between K_b and K_d.

6.1.2 Computing the Equilibrium Distribution of Promoter States

To obtain transcription rates from the biochemical equations, we must first clarify what we assume biologically and what phenomena we neglect.

All the experimental methods we have described measure mRNA quantities (or transcription rates) in a large population of cells. This means that, when formulating a model, we can only model the mean behavior over such a population, rather than single-cell behavior. We therefore compute the *mean transcription rate* for a target gene in a cell population.

We assume that the change in concentration of a regulator is much slower than the kinetics of reactions described, and that at each time point the system is nearly at an equilibrium. Thus, we model the reactions described in the previous section in steady-state. A further assumption we make is that the number of active regulator molecules in each cell is much larger than the number of its target sites, thus neglecting any possible competition between different target genes on the same regulator.

To compute the distribution over promoter states, we first translate the state equations into a set of concentration equations, which have two types: conservation equations, and mass-action type equations.

Conservation equations simply sum the quantities of a component (e.g. a site, or a regulator molecule) in different states, making sure they equal the total amount of that component.

For the two regulator case we described earlier, the conservation equations are:

$$\begin{aligned} S_{tot} &= S + SA + SB + SAB \\ A_{tot} &= A + SA + SAB \\ B_{tot} &= B + SB + SAB \end{aligned} \tag{6.6}$$

where the quantities in the equations denote number of entities in the cell population (i.e., SA means number of sites in the population bound to A regulator only, and so on).

The mass action equations describe the equilibrium state of each reaction, under the assumption of *detailed balance*. Detailed balance means that between any two states which are connected by a reaction (e.g. S and SA), the flux in both directions of the reaction is equal. This means that the number of associations occurring per second in the population equals the number of disassociations occurring per second in this reaction. We note that unless the reaction graph has a cycle (as in Figure 6.3(a)), detailed balance is a direct result of equilibrium. However, we will assume it holds even when the reaction graph allows other forms of equilibrium.

The mass action equations for the independent $S + A \rightleftharpoons SA$ and $S + B \rightleftharpoons SB$ reactions are:

$$\begin{aligned} K_b^A S \cdot A &= K_d^A SA \\ K_b^B S \cdot B &= K_d^B SB \end{aligned}$$

For the other cases (synergistic, competitive) the equations take a similar form, with different rate constants.

Depending on whether no regulator, A, B, or both are bound to the promoter, we distinguish four

possible binding site configurations. We denote the fraction of sites in the population at each configuration by $S^{-,-}$, $S^{A,-}$, $S^{xb-,B}$, and $S^{A,B}$(so, for example, $S^{A,B} = SAB/S_{tot}$). We can express each of these fractions by combining the conservation equations with the mass-action equations. In the case of independent binding the resulting distribution is:

$$\begin{aligned} S^{-,-} &= 1/Z & S^{xb-,B} &= \gamma_B B/Z \\ S^{A,-} &= \gamma_A A/Z & S^{A,B} &= \gamma_A A \gamma_B B/Z \end{aligned}$$

where $Z = (1 + \gamma_A A)(1 + \gamma_B B)$ is a normalizing constant, or a *partition function*: it is a sum of all the enumerators, or explicitly:

$$Z = 1 + \gamma_A A + \gamma_B B + \gamma_A A \gamma_B B$$

When we solve the equations for the synergistic case described in Eq. (6.5), we get a similar result, except for the double binded case, which now becomes:

$$S^{A,B} = \gamma_A A \gamma_{AB} B/Z = \gamma_B B \gamma_{BA} A/Z$$

and the partition function Z is changed to maintain normalization. We note that we got a constraint on the relation between the different constants:

$$\gamma_A \gamma_{AB} = \gamma_B \gamma_{BA}$$

which is no surprise, since these four constants are all equilibrium ratios between concentrations in an interconnected system of reactions, and therefore they are not independent. Algebraically, we have 5 equations (4 mass action equations and one conservation equation) and 4 hidden variables, meaning the system is over-constrained, and the effective number of free parameters is down by one. If we repeat the same computation for three regulators, A, B and C, the reaction graph is cube-like, describing 12 reactions, each with its equilibrium constant. Now we have 13 equations (13 mass-action equations and one conservation equation) and 8 hidden variables. This means the number of free γ parameters is down from 12 to 7.

6.1.3 A Generic Regulation Function

We can now define a regulation function for two regulators as a generic weighted sum over all possible binding states. In the non-competitive case, there are four such states, and the regulation function looks like:

$$g(H_1, H_2 : \vec{\alpha}, \beta, \gamma_1, \gamma_2) = \beta(\alpha^{-,-} S^{-,-} + \alpha^{-,B} S^{xb-,B} + \alpha^{A,-} S^{A,-} + \alpha^{A,B} S^{A,B}) \qquad (6.7)$$

where H_1 and H_2 are the concentrations of the two regulators, β is the maximal transcription rate and $\vec{\alpha}$ is the vector of α parameters indicating the activity level of the different binding states. For example, for two non-cooperative activators we can set $\alpha^{A,-}, \alpha^{-,B}, \alpha^{A,B}$ to 1, and $\alpha^{-,-}$ to 0, reflecting that transcription occurs whenever at least one regulator is bound. For a more realistic model, where different promoter states may result in different rates, we can allow α to take any real value in the range $[0, 1]$.

The regulation function can be easily extended to any number of regulators, and to handle more complex scenarios, such as competitive or cooperative interactions between different regulators or a single regulator with two binding sites, each with a different effect on transcription. Let D be the number of factors controlling the promoter. Let $\mathbf{H} = H_1, \ldots, H_D$ denote the active protein concentrations of those D factors. A generic function for the case of independent non-competitive binding can be written as:

$$g(\mathbf{H}; \beta, \vec{\gamma}, \vec{\alpha}) = \beta \frac{\sum_{I \in \{0,1\}^D} \alpha_I \prod_{i=1}^{D} I_i \gamma_i H_i^{n_i}}{\prod_{i=1}^{D} (1 + \gamma_i H_i^{n_i})} \tag{6.8}$$

Here each vector $I \in \{0, 1\}^D$ denotes a promoter binding state: $I_i = 1$ means that H_i is bound, and $I_i = 0$ means H_i is not bound. The n_i coefficients are called Hill coefficients (Hill, 1913). In enzyme kinetics, they measure the cooperativity in binding of effector molecules to multiple attachment sites on a target molecule (e.g. oxygen molecules binding to Hemoglobin). In our model they can also partially absorb effects of multiple binding sites per regulator with possible synergism between them, though their use implies that the cooperation between the multiple regulator copies is required also for regulation. In the experiments we report below, we set those coefficients to 1.

If we want to model competition in binding, not all vectors in $\{0, 1\}^D$ necessarily represent a legitimate binding state. Let $C \subset \{0, 1\}^D$ denote the set of legitimate binding states, then the corrected regulation function is:

$$g(\mathbf{H}; \beta, \vec{\gamma}, \vec{\alpha}) = \beta \frac{\sum_{I \in C} \alpha_I \prod_{i=1}^{D} I_i \gamma_i H_i^{n_i}}{\sum_{I \in C} \prod_{i=1}^{D} I_i \gamma_i H_i^{n_i}} \tag{6.9}$$

This formalism can express both cooperativity and competition between regulators. Competition on binding is captured by the terms which are absent from the sums. Cooperativity is captured by the different magnitudes of the α activation levels for different binding combinations.

6.2 Temporal Regulation Modeling using Dynamic Bayesian Networks

To model a regulatory network, we need to consider not only multiple regulators, but also multiple target genes and their *temporal* behaviour. Since regulators typically regulate multiple targets in the same regulon (Lee et al., 2002; Shen-Orr et al., 2002), the same activity levels of a regulator H can be used in the regulation functions of all of its targets. However, the functions themselves are gene

specific. Consider a simple system of K genes that are regulated by the same regulator H where we measure transcription rates at T time points. Is it possible to reconstruct the values of H at different times, and the gene specific reaction constants? Since we have $K \times T$ observations, and we assume that these can be explained by T values of H and $2K$ parameters (different β and γ for each gene), we have an over-constrained problem when $K > 2$ and $T > 2$. Thus, such a reconstruction is feasible in principle.

We use the language of *dynamic Bayesian networks* (DBNs) which we reviewed in Section 2.6. By this we model the system as a stationary Markovian stochastic process over discrete time points. Our model combines a *regulation diagram* (e.g., Figure 6.4(a)) that summarizes the regulation topology between two types of attributes: the activity of regulators $H_1, \ldots, H_D$ and the transcription rates of target genes $R_1, \ldots, R_K$. The state of the system at time point t is described by random variables $H_1^{(t)}, H_2^{(t)}, \ldots$ and $R_1^{(t)}, R_2^{(t)}, \ldots$ that denote the values of all the system's attributes at time t.

The model describes relations between variables at the same time point and at consecutive time points. First, to represent the behaviour of the regulator activity attribute, we assume that $H_i^{(t+1)}$ depends on $H_i^{(t)}$. We model this dependence with the *persistence equation*:

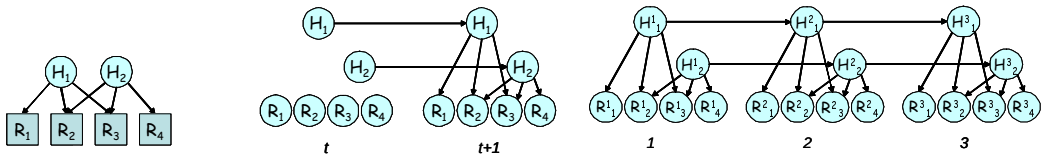
$$H_i^{(t+1)} = H_i^{(t)} + \epsilon_{h_i}^{(t+1)} \tag{6.10}$$

where $\epsilon_{h_i}^{(t+1)}$ is a normally distributed noise variable with zero mean and variance σ_i. By modeling the magnitude of change, our model prefers a smoother sequence of values H_i. Second, the transcription rate of each target gene depends on the instantaneous activity levels of the regulators that control it, as encoded by the regulation diagram. For example, if R_k depends on two regulators H_1 and H_3, then

$$R_k^{(t)} = g(H_1^{(t)}, H_3^{(t)} : \vec{\alpha}^k, \beta^k, \vec{\gamma}^k)\left(1 + \epsilon_{r_k}^{(t)}\right) \tag{6.11}$$

where $g(.)$ is the regulation function given by Eq. (6.9). The parameters $\vec{\alpha}^k$,β^k and $\vec{\gamma}^k$ are gene-specific (so for example, $\vec{\gamma}^k = [\gamma_1^k \ \ \gamma_3^k]$). The noise variable $\epsilon_{r_k}^{(t)}$ is again Gaussian with zero mean and variance σ_k. Note we use a multiplicative noise model: the noise level for R_k depends on its expected value given the regulator activity levels. As we reviewed in Section 1.2.3, and as can be observed in most gene expression data sets, the most dominant sources of noise in expression experiments are multiplicative. We note that Eq. (6.10) and Eq. (6.11) define the conditional probabilities $P(H_i^{(t+1)} \mid H_i^{(t)})$ and $P(R_k^{(t+1)} \mid H_1^{(t+1)}, \ldots, H_D^{(t+1)})$, respectively.

Figure 6.4(b) illustrates the DBN structure that corresponds to the transcriptional network of Figure 6.4(a). This structure represents the dependencies of variables at time $t+1$ on variables in the previous time step and the current time state. When we want to model the behavior of the system in the time range $1, \ldots, T$, we duplicate this structure $T-1$ times to get a Bayesian network with T copies of each variable. Figure 6.4(c) illustrates this for three time points.



(a) Regulation Diagram (b) DBN model (c) Unrolled network

Figure 6.4: Schematic representation of a DBN model for temporal gene regulation. (a) A regulation diagram with 2 regulators and 4 targets. (b) A 2-TBN representation of the DBN induced by this diagram. (c) An example of the unrolled Bayesian network model for three time points.

The DBN model for time range $1, \ldots, T$ defines a joint distribution over all the random variables in these T time points. The joint density of an assignment to all the variables is the product of the densities of the values of error variables $\epsilon_{h_i}^{(t)}$ and $\epsilon_{r_k}^{(t)}$ that achieve equality in Eq. (6.10) and Eq. (6.11):

$$\begin{aligned} & P(\mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(T)}, \mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(T)}) \\ &= \prod_{i=1}^{D} [P(H_i^{(1)}) \prod_{t=2}^{T} P(H_i^{(t)} \mid H_i^{(t-1)})] \prod_{k=1}^{K} [\prod_{t=1}^{T} P(R_k^{(t)} \mid \mathbf{H}^{(t)})] \\ &= \prod_{t=1}^{T} [\prod_{i=1}^{D} P(\epsilon_{h_i}^{(t)}) \prod_{k=1}^{K} P(\epsilon_{r_k}^{(t)})] \end{aligned} \tag{6.12}$$

6.2.1 Parameter Estimation

Once we define the DBN model, we can learn the kinetic parameters and the hidden activity levels of regulators from observations. We consider an observed set $\mathbf{E} = \mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(T)}$ of transcription rates of K genes in T time points and try to optimize for the most likely assignment of parameters and levels. Thus, assuming a fixed regulation diagram $\mathcal{G}$, we want to find parameters that maximize the likelihood

$$\ell(\mathbf{h}, \theta : \mathcal{G}, \mathbf{E}) = \log P(\mathbf{E}, \mathbf{h} \mid \theta, \mathcal{G})$$

where $\mathbf{h}$ are the values of the unobserved regulator activity levels at different times, and θ are the kinetic and variance parameters of the model. The right hand side can be decomposed according to Eq. (6.12). In order to make sure we learn sensible values for the model parameters, we follow the Bayesian approach and use priors over them. What priors are appropriate for each parameter type? First, we would like to ensure that all parameters end up positive. Also, we want the α activities to be bounded between $[0, 1]$. We therefore use a uniform prior over this range for the α parameters. For the γ affinity parameters we use a gamma prior of the following form:

$$P(\gamma) \propto \gamma^{a-1} e^{-b\gamma}$$

Incorporating the priors, we replace the maximum likelihood objective with the joint likelihood objective:

$$\log P(\mathbf{E}, \mathbf{h}, \theta \mid \mathcal{G}) = \log P(\mathbf{E}, \mathbf{h} \mid \theta, \mathcal{G}) + \log P(\theta \mid \mathcal{G})$$

To optimize this likelihood function w.r.t. $\mathbf{h}$ and θ, we perform piecewise optimization: In each step we optimize part of the $\{\mathbf{h}, \theta\}$ space, while keeping the rest of it fixed. The optimization is done using an analytical solution when possible, and gradient ascent otherwise, depending on which parameters are optimized. We call this approach the *Max-Max* algorithm.

A possible pitfall with the *Max-Max* approach is that we are optimizing over a high dimensional space (the time profiles of the hidden regulators can extend to a few hundred time points in promoter activity data). This is both costly in time and prone to have many local maxima. In addition, the parameters estimated in this approach are sensitive to the particular value of $\mathbf{h}$ that was reconstructed. A more cautious approach is to maximize the likelihood of the observed data, which integrates over all possible values of the regulator activity:

$$\max_{\theta} \log P(\mathbf{E}, \theta) = \max_{\theta} \log \int P(\mathbf{E}, \mathbf{h}, \theta) d\mathbf{h}$$

To optimize the likelihood, we use an *Expectation-Maximization* (EM) approach (Dempster et al., 1977). In EM we alternate between computing the posteriors $P(\mathbf{h} \mid \mathbf{E}, \theta)$ (E-step) and optimizing the parameters θ with respect to $E_{P(\mathbf{h}|\mathbf{E},\theta)} \log P(\mathbf{E}, \mathbf{h}, \theta)$ (M-step). It can be shown that each iteration of applying these two steps increases the value of $\log P(\mathbf{E}, \theta)$. The posterior probabilities over the hidden regulators of the E-step are needed to compute the *expected sufficient statistics* of each parameter in θ. These sufficient statistics are then used in the M-step optimization. At the end of the EM process we also have a posterior distribution over $\mathbf{h}$ from the last E-step. We can now take the means of those posteriors as our estimate of $\mathbf{h}$, and we can also put error bars on that estimate using the standard deviations from the posteriors.

To avoid over fitting of the model to the data, we match the model complexity to the amount of available data. When data is scarce, we fix some of the parameters in advance, whereas when the amount of data grows, we attempt to learn more parameters. In the current study, we preset the α parameters in the smaller studied systems to either 0 or 1, according to biological knowledge, keeping the number of free parameters low. In the larger systems, where we have more observations, we learn a single "leak" α parameter for each gene, to allow basal transcription levels in all "inactive" binding states. The number of parameters optimized per each of the target gene is between 3 and 5 in our experiments. This number is much lower than the number of observations (T for each gene).

6.2.2 Structure Learning

Besides using our model for estimating parameters and hidden variables, we also want to answer some model selection questions. These can be specific questions, such as "Which of the target genes is regulated by another repressor?", but also general structure selection tasks, like improving

an initial network structure, or even learning one from scratch. Since we formulated our model as a Bayesian network, we can perform model selection with the tools we described in Chapter 2 and Chapter 4.

In our framework we use the score based approach for structure selection, employing the BIC score (see Section 2.4.3). For the smaller systems, we can use either exhaustive or standard greedy search, depending on the number of variables involved. For the larger systems, this becomes prohibitively expensive, since each score evaluation requires full parameter optimization as described above. To overcome this obstacle we use the "Ideal Parent" method (described in Section 4.2) both for speeding up standard structure search, and for introducing new regulators into the model.

6.2.3 Transcription Rates

As we stressed earlier, we want to use mRNA transcription rates (as opposed to transcript levels, for example) in our regulation models. We now turn to the problem of extracting those rates from time series of two current experimental methods: promoter activity measurements, and microarray measurements.

Promoter Activity Data

In Section 1.2.2 we described the method of reporter plasmids in E. Coli to measure the promoter activity of different operons. In this experimental setting two quantities are measured in each well: The fluorescence level of GFP, which is indicative of total accumulated promoter activity for this operon, and the optical density, which is indicative of the cell population size.

Denoting by $G_k^{(t)}$ the GFP level for the promoter of operon k at time t, and by $O_k^{(t)}$ the optical density for the same plate, we are interested in the RNA transcription rate per cell. As we explained in Section 1.2.2, we should have $r_k^{(t)} = \frac{1}{O_k^{(t)}} \frac{d}{dt} G_k^{(t)}$. Unfortunately, measurement noise for both types of observations can lead to large fluctuations in numerical differentiation. This is particularly true for small OD values. To deal with this problem, we follow Ronen et al. (2002), and perform numerical smoothing of the measured GFP and OD levels before differentiating them. The resulting rates are then smoothed. The smoothing is done using a simple 20%-mean filter with a small window size. This procedure is good at removing high frequency oscillations in the gradient, but can suffer from lower frequency noise.

Microarray Data

Microarray techniques are the dominant method for parallel measurements of expression today, with many data sets available publicly, including time series data. The methods we describe here can be equally applied to data from oligonucleotide chips and from cDNA microarrays, given that we can extract transcript levels from them. For cDNA arrays, the measurements are of the form of log ratios between the expression levels of two conditions M_1 and M_0. For time series data, M_0 is

usually a common control. In this case, we can reconstruct the expression level up to a gene specific multiplicative constant that involves the control level of the gene and probe-specific issues such as hybridization efficiency. In oligonucleotide chips, the measurements usually signify raw expression levels (see Section 1.2.1).

To recover transcription rates, we consider how the mRNA expression level depends on both transcription and degradation, and use a simple gene-specific mRNA decay model:

$$\frac{d}{dt}e_k^{(t)} = r_k^{(t)} - \delta_k e_k^{(t)} \tag{6.13}$$

where $e_k^{(t)}$ is the expression level of gene k at time t and δ_k is the mRNA decay rate of gene k. We assume that mRNA decay rates may be gene-specific, but remain constant in time. Given the decay rate δ_k, and the expression measurements, we recover $r_k^{(t)}$ (up to a gene-specific multiplicative factor) by solving the differential equation Eq. (6.13). Such actual decay rates have been measured experimentally under specific conditions by several recent genome-wide studies (Holstege et al., 1998; Wang et al., 2002).

We note that the same model can be used to estimate transcription rates in steady state data sets (such as measurements under different conditions). In this case mRNA levels are assumed to be stable, *i.e.* $\frac{d}{dt}e_k^{(t)} = 0$, and so we get $r_k^{(t)} = \delta_k e_k^{(t)}$ from our decay model, meaning the rates are directly proportional to the measured transcript levels. Again, we assume the mRNA decay rates remain unchanged in the different samples.

6.3 Results on Small-Scale Systems

To understand the power and limitations of our methods, we started by evaluation on data sets from small-scale systems. In these experiments we try to assess how well our framework performs on the basic tasks of parameter estimation, test set prediction, and simple model selection. We first test these tasks on synthetic data sets, and then apply the method to two real data sets: a promoter activity data set from E. Coli, and a cDNA microarray data set from yeast.

We generated several time series for two regulators H_1 and H_2 from distinct smooth positive functions, including single and multiple modes. We then generated rates for 5 genes, using H_1, H_2 as either repressors or activators, depending on the experiment, and several distinct values of β and γ, with additional white noise. The number of genes was so chosen as to have a sample of different connectivity schemes and different parameterizations, yet still be able to analyze the results visually. The length of each time series was chosen to be 100 points, to resemble real promoter activity data sets (see Section 6.3.3). To simulate promoter activity measurements, O_k profiles were simulated to resemble actual profiles from real data. $G_k^{(t)}$ was calculated using $r_k^{(t)}$ and $O_k^{(t)}$, and finally "observed" series GFP and OD were generated by adding noise to $G_k^{(t)}$ and $O_k^{(t)}$, respectively. From these data sets, two types of datasets were generated: one containing the actual *real* rates r, and the other with the observations of GFP and OD, to which we applied the pre-processing steps of

estimating rates, as described above, to get *smoothed* rates.

6.3.1 Parameter Learning and Hidden Regulator Recovery

First, we tested recovery of the kinetic parameters β and γ, and the hidden regulator activity profile, $\mathbf{h}$. The α activation levels were preset to 0 or 1 in these experiments. We compared our method to the one suggested by Ronen et al. (2002). Their method is based on singular value decomposition (SVD), and is directly applicable only to systems with a single repressor. We therefore compared performance on such a system.

What accuracy should we expect to see when learning the model parameters? When we examine the regulation function behaviour for different parameter ranges (Figure 6.2(b)), we see that for large values of γ the function quickly saturates. Thus, in the high γ range the rate profile is less sensitive to changes in γ's value, and so we might expect the accuracy of parameter reconstruction to be lower than in small values of γ. Note that the baseline rate parameter, β, does not have a similar effect, since it only controls the scale of r.

To test this effect, we sampled γ parameters from different ranges. We then applied the different learning methods, and examined the learned parameters. Two phenomena were observed for all methods. First, the error in the learned parameters (both γ and β) correlates with the magnitude of γ as predicted. Second, as one might expect, the parameter estimates from actual rates were more accurate than estimates from rates that were reconstructed by smoothing the derivatives of simulated measurements. The large errors, in the latter case, mostly reflect noise introduced into the data by the smoothing process. The comparison shows that our generative models usually have smaller errors, an effect that increases with the level of γ.

We also examined the quality of regulator profile recreation in those experiments by looking at the correlation between learned and real $\mathbf{h}$ profiles. All methods, when applied to real rates, achieve almost perfect correlation (0.98 and higher). This shows that estimation of $\mathbf{h}$ is robust to errors in estimation of the kinetic parameters. However, when learning from smoothed rates, large values of γ exert an effect through the deformations of those rates. Generally, we expect such problems to diminish as the number of modeled target genes grows, since the major pre-processing noise is not correlated between the different genes, and therefore its effect will be smoothed out.

6.3.2 Identifying an Additional Regulator

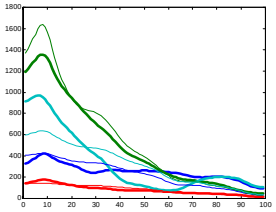
A key question we can ask in such small-scale, single regulator systems, is whether some of the genes in the regulation module are affected by an additional, non-cooperative regulator. We would like to be able to identify those genes, as well as reproduce the dynamics of the second regulator. This is a specific model selection task, where the solution is constrained to a subset of all possible models. To solve it, we consider different network architectures. For each one, we train parameters and score it using the BIC score. We then select the architecture with the highest score (our tested system is small enough that we can score all the allowed structures).

Table 6.1: Structure learning results for 6 different regulator profiles, using either *Max-Max* (MM) or *EM* for parameter learning. Each column represents a different choice of genes regulated by H_2 in the true network. The figures given for each combination are in the format correct/contained/missed, where *correct* means the true structure received the highest score, *contained* means the true connections are contained in the learned ones, and *missed* means other connections were learned. The cases where the exact complement set of connections was learned are counted in parentheses.

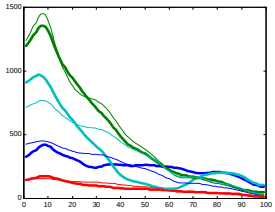
	$\{\}$	$\{3\}$	$\{2, 3\}$
MM	0/0/6	5/1/0	3(+3)/0/0
EM	5/0/1	0/6/0	1(+1)/4/0

	Set 1	Set 2	Set 3	Set 4
MM	> 1	0.15	0.15	0.24
EM	0.50	0.16	0.15	0.23
SVD	> 1	0.16	0.17	0.24

(a)



(b) SVD



(c) MM

Figure 6.5: Test set prediction on E. Coli promoter activity data. (a) Average test set prediction error on 4 promoter activity experiments of Ronen et al. (2002). (b) and (c) true rates (solid lines) vs. predicted rates (thin lines) for 4 genes in test set 4 for two of the methods.

In Table 6.1 we report qualitative structure learning results on simulated data sets from 6 different regulator time profiles with 3 different network architectures. We observe that the correct set of genes is almost always contained in the learned set, with some tendency to learn excessive connections. We note, however, that in almost all the cases where spurious connections were learned, their learned γ parameters were about an order of magnitude lower than those learned for the true connections, implying a much weaker affect of H_2 on those genes. Note that when the true set was not chosen to be connected to H_2, its complement set was (i.e. genes $(1, 4, 5)$, when the true set is $(2, 3)$), which allows the model exactly the same freedom to fit the data. This is an example where the system does not have a unique solution, or is not *identifiable*. We discuss the identifiability problem in Section 6.4.

6.3.3 Example I: Test Data Prediction in E. Coli SOS System

An important test for the validity of the learned models is their generalization ability, or predictive power over new data sets. Basically, given enough flexibility, a model can fit exactly any training set, but still not be able to perform predictions on new data. To perform this test, we used promoter activity time series measurements of 8 operons from the SOS system of *E. Coli*, from an experiment where a UV shock caused the activation of these genes by indirect cleavage of a repressor gene *lexA* (Ronen et al., 2002). The time series are 100 time points long, with 3 minute intervals between

measurements. We checked 4 experiments, where each experiment contains two repeats of the same protocol. We used one repeat for training and the other as a test set. Figure 6.5(a) reports the mean fractional error (MFE) averaged over all genes in each test data set, where MFE is defined as the mean of $|r^{(t)} - r^{(t)}_{pred}|/r^{(t)}$ over the whole time series. We can see that with the exception of one data set, all test sets are predicted within 15 to 25 percent average error. Figure 6.5(b-c) show test set predictions in set number 4, having 24% average error, for 4 representative genes. We can see that the MFE estimate might be over-pessimistic as it is mostly affected by errors at low values of $r(t)$. Overall, the results suggest that the simple regulation model of Eq. (6.2) can capture the behavior of the real dynamics of a gene circuit.

6.3.4 Example II: Yox1-Mcm1 Two Regulator System

Recent work (Pramila et al., 2002) shows that M phase-expressed genes in yeast can be distinguished into two sub-sets. A major set which is activated by Mcm1 and is expressed earlier in M phase, and a minor set which is activated by Mcm1 and repressed by Yox1, with delayed expression in late M phase. To evaluate the dynamics of this system, we built a model of this network, based on known Mcm1 and Yox1 targets (Pramila et al., 2002; Simon et al., 2001). We then used the cell-cycle mRNA expression data of Spellman et al. (1998) (see Section 5.2) and experimentally derived decay rates (Wang et al., 2002), to estimate transcription rates for these genes. We applied our parameter learning methods on each of the time series and learned activity profiles for the two regulators. As seen in Figure 6.6, the reconstructed activity level of Yox1 peaks earlier than that of Mcm1, consistent with its documented repressive role, and explaining the subsequent shift in the peak transcription levels of its target genes compared to that of Mcm1-exclusive targets. Surprisingly, Yox1's reconstructed activity peak appears relatively early in the cell cycle, before M phase. This novel finding was also obtained on a separate time series (data not shown) and is corroborated by Yox1's expression profile (Figure 6.6(b)). Note, that the regulator expression profiles themselves are not used in the reconstruction. This allows us to recover the hidden activity levels of regulators that are themselves not transcriptionaly regulated. For example, we accurately reconstruct the activity profile of Mcm1, which is not transcriptionaly regulated, with a clear peak at M phase.

6.4 Modeling Larger Systems: A Non-Linear Dimensionality Reduction

Up to now we have concentrated on small systems with one or two regulators. The questions there concentrated on recovery of unknown parameters and activity levels, as well as on specific hypothesis regarding the regulation diagram.

A more ambitious goal is to apply this framework to a large system, comprised of several regulators and many target genes. At its extreme, we would have liked to apply the method to the whole

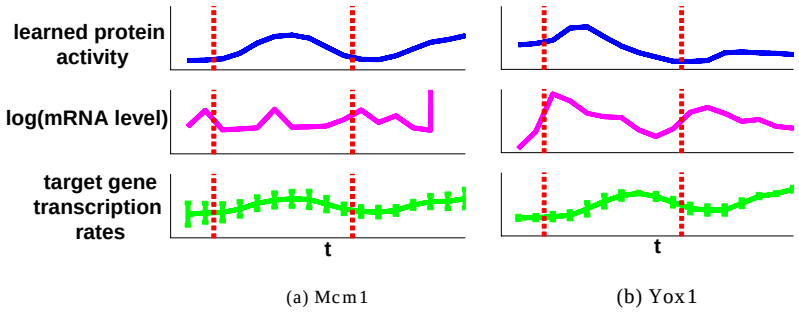


Figure 6.6: Regulator reconstruction in an activator-repressor system. (a) The learned activity for Mcm1 (top), vs. its mRNA log expression levels (middle), and its target genes transcription rates (bottom). Vertical lines denote cell cycle start points (end of M/G1 transition). (b) Same for the repressor Yox1.

transcriptome of the cell. As the number of regulators is an order of magnitude smaller than the total number of genes, learning a model which describes the transcriptome using the dynamics of the regulators and static affinity parameters would provide a realistic, biology-based dimensionality reduction. In a sense, we can leverage the model large scale structure for learning small scale details an quantitative aspects: At one end, learning more accurate activity profiles and affinity parameters due to the large number of observations. At the other end, correcting the regulatory network structure by using the expression data fit to the model.

We note that in many cases, the size and variability of a regulation system make the inference problem highly constrained, and so a unique solution might be imposed. This is in contrast to smaller systems where, as we saw in Section 6.3.2, there can be multiple solutions with the same expressive power. Liao et al. (2003) analyze this identifiability problem for a similar model with linear interactions, and give formal conditions for the system to be identifiable up to arbitrary scaling. These conditions require linear independence within the learned regulator profiles, and within the regulator connectivity vectors (corresponding to our γ parameters). When the number of target genes is much larger than the number of regulators and these targets have a variety of connectivity patterns, these conditions are more easily met. In our non-linear interaction model such an algebraic analysis does not apply, but we can use it as an approximation. As a matter of fact, the non-linearity of our model makes the problem even more constrained, since each regulation function (Eq. (6.9)) contains products of all orders of the hidden quantities, making it harder to decompose in different ways. For this reason, our system can be identifiable even in cases where in the linear model it is not.

Modeling the whole cell transcriptome presents two problems: First, under most experimental conditions, the majority of the genes are either silent or active at a low basal level, rendering them useless for learning a model. Second, the size of the system presents technical difficulties for learning. We therefore choose a large subsystem, which is characterized by its activity in a specific biological process. In here, we choose the yeast cell cycle system.

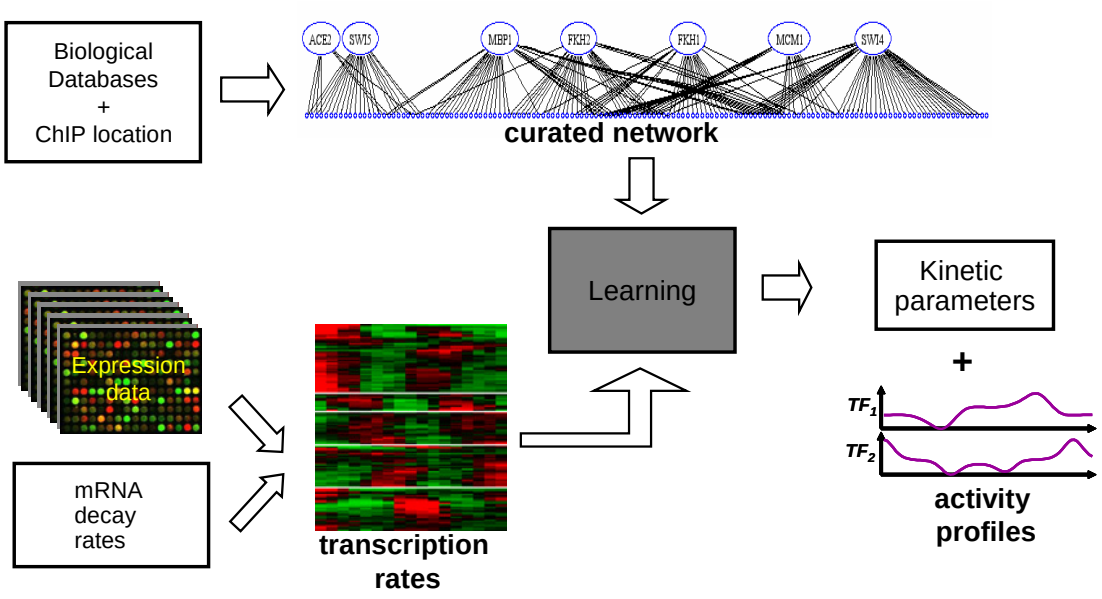


Figure 6.7: Flow of suggested method (see text).

We first study a curated model of the complex regulatory network of the entire cell cycle, and show that we can accurately identify activity levels of regulators based solely on our realistic modeling framework and the expression levels of their targets. We then employ the Ideal Parent structure learning algorithm to learn a regulatory network *ab initio*, based solely on expression data, and show the accuracy of both the resulting network topology and the reconstructed regulators' activity profiles. Finally, we combine the curated network and the structure learning algorithm, to try and improve the known network.

6.4.1 Example III: yeast cell cycle system

To test our framework on the cell cycle regulatory system of yeast, we assembled and curated a regulatory network by hand. We first picked 9 known cell cycle regulators (Simon et al., 2001). Two of these regulators are not DNA binding, and were therefore left out. We then looked for evidence of regulatory connections between the remaining 7 regulators and genes whose expression behaviour shows cell-cycle regulation (Spellman et al., 1998). As the source of data for regulatory connections, we used biological interaction databases (Costanzo et al., 2001) and data from DNA binding location experiments (Lee et al., 2002). In this experimental method, the binding of a transcription factor is measured in parallel for thousands of intergenic regions. When the method assigns high confidence to the factor's binding to a target gene's promoter region, we assumed this factor regulates that gene. The list was curated for suspicious intergenic probes. From this list we selected all genes regulated by one or two of the 7 factors. The resulting network contains 141 genes with various regulatory connections to the cell cycle regulators. Following the literature, all regulators

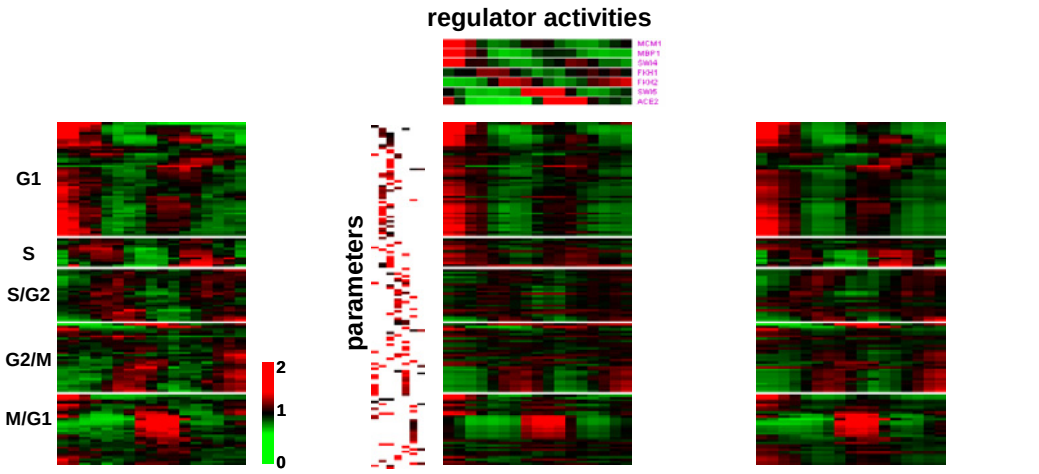


Figure 6.8: Predicted rates by curated and *ab initio* networks. (a) Input transcription rates for 141 genes, grouped by their peak expression cell-cycle phase. (b) Predicted rates in the curated model after model learning. The learned regulator activity profiles are shown at the top. The learned γ affinity parameters are shown on the left (darker color means stronger affinity). (c) Predicted rates after *ab initio* model learning (see text).

were modeled as non-cooperative activators, except for Mcm1 and Fkh1/2 which were modeled as cooperative activators (Boros et al., 2003; Simon et al., 2001). Using the alpha synchronization expression time series (Spellman et al., 1998) and experimentally derived decay rates (Wang et al., 2002), we estimated the transcription rates for the 141 genes. We then learned regulator activity profiles and kinematic parameters for this complex network (using *Max-Max* optimization). The flow of this experiment is shown in Figure 6.7.

One important aspect of model learning is the input data fit to the model. By plugging the learned activity profiles and kinetic parameters into the regulation functions, we can get the transcription rates that the model *predicts*, ignoring the noise component. The discrepancy between these rates and the input rates measure how well the model describes the data. The higher this discrepancy is, the higher the signal in the data which is attributed to noise. The predicted rates we learned for the 141 genes are shown in Figure 6.8, as well as the learned activity profiles and affinity parameters. We see we get a pretty good reconstruction of the 2397 data points, using only $7 \times 17 = 119$ regulator activity points and 466 parameters. For all seven transcription factors, the algorithm learned cyclic activity levels. But does their behavior correspond to what we expect biologically? Figure 6.9 shows the learned activity profiles for the 7 modeled regulators, against their mRNA expression levels and their target genes behaviour. We first note that the regulator profiles are consistent with their known activity based on molecular or genetic studies. For

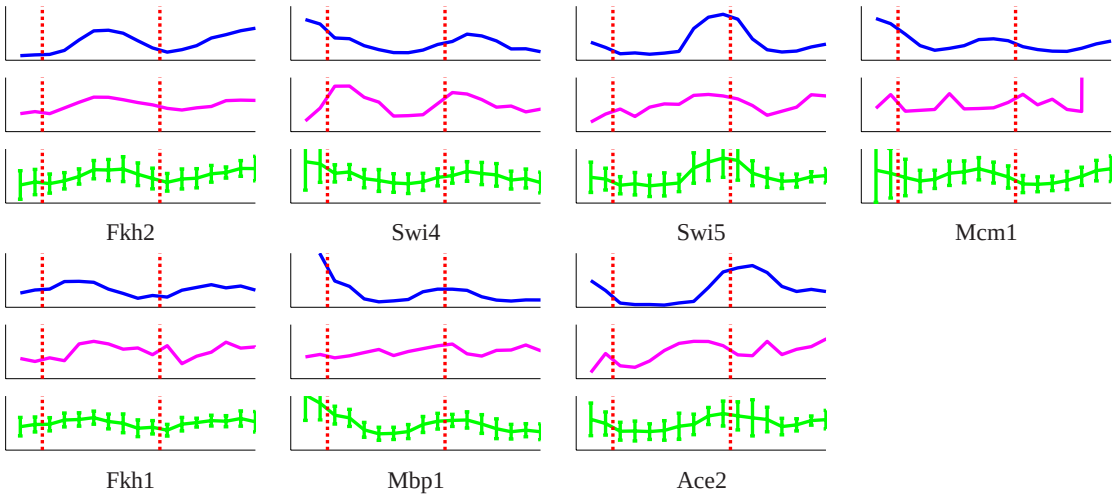


Figure 6.9: Regulator activity profiles learned from the curated network diagram. Axis meanings are as in Figure 6.6.

example, Swi5's activity peaks at late M/G1 and early G1, consistent with its previously reported activity (McBride et al., 1999); Mbp1 and Swi4's activity levels peak at mid to late G1 consistent with their role in G1/S gene expression (Baetz and Andrews, 1999); and Fkh1 and Fkh2 peak at late S/G2 and G2/M respectively, consistent with their reported effects in genetic studies (Hollenhorst et al., 2000). Thus, in many cases (*e.g.*, Swi5 and Swi4 or Fkh1 and Fkh2), the reconstructed activity levels distinguish between relatively subtle but important differences in true biological activities, the establishment of which has often required a large number of experiments. In some cases (*e.g.* Fkh2 or Swi5), our reconstructed activity profiles closely resemble the regulator's expression profile. But more importantly, since our reconstruction does not use the expression levels of the regulators, we are able to accurately reconstruct their activity levels even if they are not regulated transcriptionaly (*e.g.*, Mcm1), or if their expression and activity profiles are shifted (*e.g.*, Ace2), highlighting the power of our approach.

The full power of our framework lies in its ability to learn not only accurate activity profiles and kinematic parameters, but also the full network architecture *ab initio*. We therefore ran the Ideal Parent structure learning algorithm on a naïve network with the same 141 target genes all wired to a single activator. We allowed the algorithm to add more regulators and change regulatory connections until convergence. In this experiment we used simple correlation as the similarity measure between profiles. Addition of a new regulator was applied only if no other modification was accepted. To add a new regulator, we apply the CLUST algorithm (Ben-Dor et al., 1999) to find clusters of ideal regulator profiles that are highly correlated (above 0.8), and may correspond to a new regulator of the genes for which these ideal profiles were generated. We evaluate each proposed new regulator by introducing it into the network, and then apply gradient ascent to find the best parameter values and regulators activity profiles for the modified network. We then choose the new regulator that leads to the biggest score improvement and add it and its target links to the current network. When

no such regulator offers a positive improvement, no action is taken.

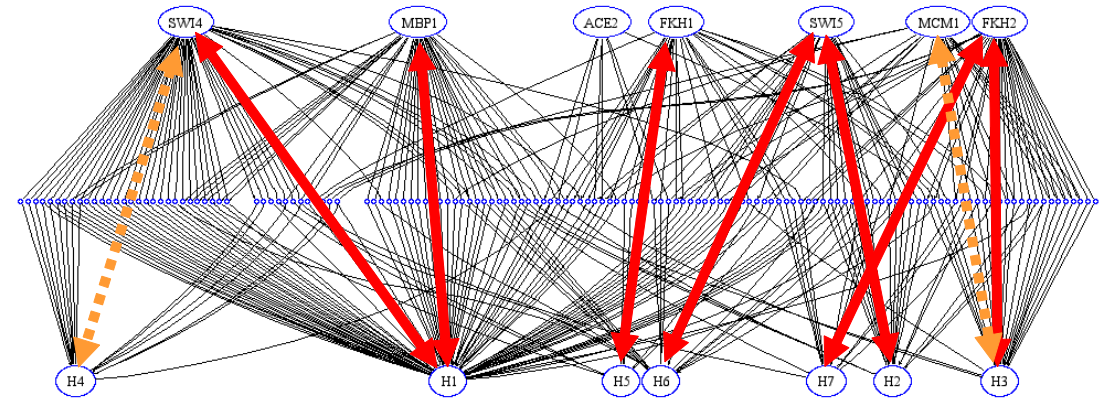
The resulting networks in several runs had between 6 and 9 regulators, where the first 6 or 7 were very similar between runs. Figure 6.8(c) shows predicted rates for one of those runs. As we can see, the data fitting of the ab initio model is generally better than that of the curated model, especially for genes peaking in S or M/G1 phase. The BIC score of the ab initio model was correspondingly better, even though the number of parameters (461) is essentially the same.

To evaluate the quality of our *ab initio* reconstructed network and identify the reconstructed regulators, we compared the topology of the learned network to that of the curated one (Figure 6.10(a,b)), and the learned activity profiles to those learned on the curated network (Figure 6.10(c)). In some cases, such as inferred regulator $H2$ and the known regulator Swi5, the correspondence in both targets and activity levels is striking (Figure 6.10(d) and (b,c), second row). In others, a single inferred regulator corresponds to two separate factors with similar activity patterns (*e.g.*, regulator $H1$ and the G1/S factors Mbp1 and Swi4). Overall, since in the known network some targets are regulated by more than one factor and some factors have similar profiles, by combining both tests we can roughly identify most of our inferred profiles (regulators 1, 2, 3, 5, 6 and 7) with known regulatory activities (MBF/SBF, Swi5, the Fkh2/Mcm1 complex, Fkh1, Swi5/Ace2 activity, and Fkh1/Fkh2 activity). Thus, these tests indicate that the inferred regulators have both targets and activity levels very similar to those in the known curated network, and highlight the success of our approach in learning both correct structure and parameters in the most stringent challenge.

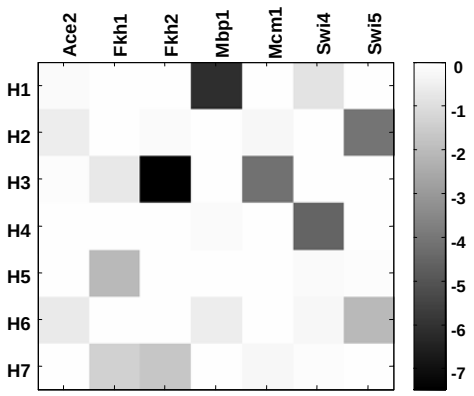
Finally, despite their impressive correspondence, both the *ab initio* learned network and the curated model are likely only approximations of the true biological systems. Thus, we combined our curated network and our structure learning approach, and used the curated network as a starting point for the structure learning algorithm, trying to improve the known structure. Indeed, this yielded a dramatic improvement in score (610 bits), by introducing changes in the connections for about 35 genes (despite not adding new regulators), primarily changing genes from SBF to MBF regulation and from Fkh1 to Fkh2. These modifications suggest novel hypotheses, potentially extending our partial biological knowledge.

6.5 Discussion

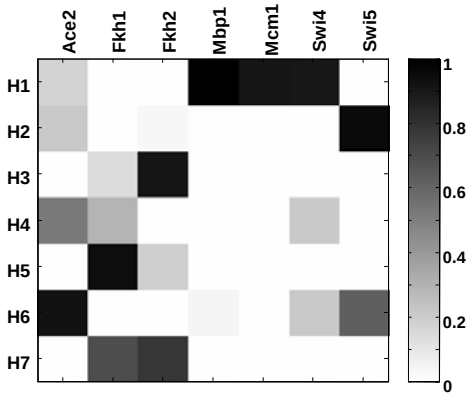
In this chapter, we examined the question of learning the dynamics of transcription networks, in terms of the temporal behaviour of regulators, as well as the kinetic parameters governing their effect on their targets. Our method provides a principled approach to handle a wide range of transcriptional network architectures and regulation functions. Unlike previous methods based on probabilistic models (Friedman et al., 2000; Kim et al., 2003; Ong et al., 2002; Pe'er et al., 2001), we addressed the fact that the relevant quantities - transcription rates and regulator activity levels - are usually not measured. Our DBN-based model to transcription rates and regulator activity levels allows us to handle these biologically relevant quantities despite the indirect measurement of the former and the lack of measurements of the latter. This is done by preprocessing steps to extract transcription



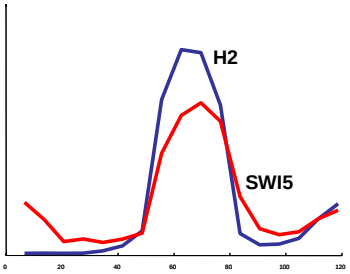
(a) Network structure



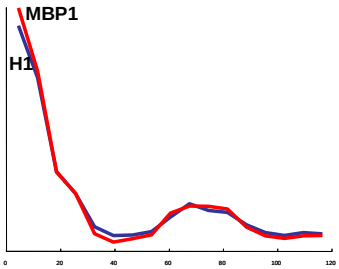
(b) Target intersection



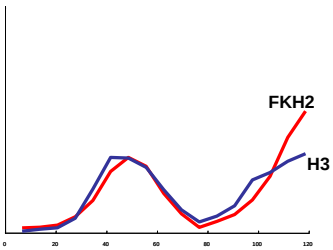
(c) Profile correlation



(d)



(e)



(f)

Figure 6.10: Comparison between the curated and *ab initio* learned models. (a) An integrated representation of the two networks; Target genes are at the middle row, regulators for the curated network are at the top, and regulators for the *ab initio* network are at the bottom. The solid red arrows show regulator pairs with significant target overlap and correlated activity. The dotted orange arrows show pairs with significant target overlap, but weakly correlated activity. (b) Log p-value of target intersection groups between known and *ab initio* regulators. (c) Positive correlations between learned activity profiles of known and *ab initio* regulators. (d-f) Three examples of highly correlated learned profile pairs of known (red) and *ab initio* (blue) regulators.

rates, and by the use of hidden variables to account for unobserved regulator activity levels. Several recent works (Battogtokh et al., 2002; Liao et al., 2003; Ronen et al., 2002) use a fixed regulation diagram to reconstruct unobserved regulator activity profiles and parameters. This work is the first to combine a network structure learning algorithm in this context. Our algorithm is based on the notion of "ideal" regulators, and we demonstrated its power on the cell cycle regulatory network.

The framework we present here allows us to handle noise inherent to the biology and the measurements in a principled way, and to learn both the parameters and the structure of the regulation network. However, our model still abstracts away some of the explicit processes that generate the actual observed expression data. A more explicit modeling of these will provide a more principled treatment of different sources of noise in the data. Furthermore, our model does not handle directly any of the upstream events that affect regulator activity. In fact, the current model is an open loop system, such that the regulation of regulator activity is itself viewed as exogenous to the system. By developing a richer modeling language we may capture more complex reaction models, model the upstream regulation of activity levels, and learn systems that involve feedback mechanisms and signaling. Finally, such extensions open the possibility of incorporating additional types of data, such as binding sites models, transcription factor binding data or protein-protein interaction data. These could serve not only as additional sources for initialization or validation of models, but also as a primary source of observations for model learning, thus widening the molecular scope covered by our framework.

Chapter 7

Discussion

7.1 Summary

In this dissertation we have developed several algorithms and model types in the Bayesian network framework, towards modeling gene regulation networks. We then presented two approaches for the analysis of such network from gene expression and other modes of data. We first introduced a non-parametric representation for local dependencies in Bayesian networks, based on Gaussian process priors, and showed its flexibility and power in detecting non-linear functional connections. We then presented two novel methods for speeding up network structure search: The Sparse Candidate algorithm achieves speedup by iteratively reducing the structure space through the selection of candidate parents. The Ideal Parent approach replaces exact and expensive scoring of candidate parents with similarity measurements to a hypothetical "Ideal" parent profile, and also uses these profiles for introducing new hidden variable into the network structure.

In the first application to regulatory network modeling, ensembles of learned Bayesian network models were used to detect statistically strong structural features, like direct connections between genes. In the second one, a realistic model of transcription regulation was formulated, and the hidden levels of regulator active proteins, as well as the kinetic parameters of regulation, were learned from gene expression and DNA binding location data. This framework was combined with the Ideal Parent method to search for network structures ab-initio, as well as improve currently known structures.

The two approaches we present for regulatory network analysis are quite different. Each of them employs different modeling assumptions, and each captures different features. The first method assumes that regulatory connections can be inferred from expression levels of the regulator and target genes. This assumption holds in many cases, in particular in steady state situations, when there is no advantage to the fast response of regulator protein activity modulation, and most of the regulation is done at the transcriptional level. Even in dynamic scenarios, such as during the cell cycle, some of the regulators are transcriptionaly regulated. By learning ensembles of networks from perturbed data sets, our method was able to capture many known regulatory connections. Besides

the yeast cell cycle system reported here, this method was successfully applied to other data sets, including a compendium of yeast deletion strains (Pe'er, 2003).

The second method we presented uses more realistic assumptions on the nature of regulatory pathways and on the relevant interacting sizes. It assumes nothing on the mechanisms regulating the regulators themselves, and models the regulators as hidden quantities. By that the method lends itself to any scenario of regulation, be it a steady state or a dynamic process, regardless of whether the regulators themselves are transcriptionaly regulated. Moreover, this method produces quantitative predictions, some of which can be experimentally verified, and not just qualitative wiring predictions. We demonstrated the method on the dynamic DNA damage pathway in E. Coli and on the yeast cell cycle system.

7.2 Related Work

Ever since the introduction of microarray measurement methods, a host of analysis approaches have been applied to them, each one with a different objective in mind. In the introduction we mentioned briefly some of the earlier approaches, including clustering methods and early network reconstruction methods.

Clustering applications to gene expression data mostly aim at grouping together genes with similar expression profiles, with the hope that they play a role in similar functional modules. Both the clustering algorithm and the similarity measure used have a big influence on the outcome of the clustering process. Moreover, the objective which the clustering algorithm tries to optimize often cannot be defined without reference to the algorithm itself (as in hierarchical clustering). The most common approach was first introduced in Eisen et al. (1998b), and uses hierarchical clustering with a Pearson correlation similarity metric. Using this similarity measure implicitly assumes a linear dependence between the genes. As we have shown in Chapter 6, in a realistic model of regulation there are non-linear dependencies between regulator and target genes, which in turn induce non-linear connections between the targets. In this sense the use of linear correlation is a simplification, just like the use of linear Gaussian connections in our Bayesian network framework of Chapter 5.

Close relatives of clustering methods are the decomposition, or dimensionality reduction methods. In these methods one attempts to express the data using a small number of components and parameters. Alter et al. (2000) first suggested the use of singular value decomposition (SVD) in this context, decomposing the gene expression data matrix to a linear combination of "eigengenes" and "eigenarrays". One can then try to assign biological meaning to the different components. For example, an eigengene might correspond to a specific regulator, but it can also represent a combination of several regulatory activities. While this approach is elegant and simple, it again lacks a biological basis. One useful application of such decompositions is in "pealing" off data components which originate in noise or experimental artifacts. A more recent work by Liao et al. (2003) introduces a method of recovering the regulatory network structure using biologically based decomposition. Their approach is similar to the one we present in Chapter 6, however it is limited

to an all-activator, full cooperativity scenario and is based on approximations which require long time intervals between measurements. Both our work and that of Liao et al. differ from the SVD decomposition in the sparseness of connections, which makes them interpretable also as network structure learning approaches.

Following the publication of our first Bayesian network based expression analysis method (Chapter 5), many followed the use of probabilistic graphical modeling for this problem. For example,Hartemink et al. (2001) use a model similar to that of Chapter 5, but additionally allow hidden variables to account for unobserved quantities like specific protein levels, and annotate the network edges for interaction sign (repression or activation). In a later work (Hartemink et al., 2002) they use DNA binding location data for setting a prior over regulatory connections in such models. Imoto et al. (2003) suggested the use of non-parametric regression for modeling local interactions within Bayesian network models. The use of the simpler, deterministic models (like Boolean networks) have seen a decrease in recent years. Linear interaction model methods have also acknowledged recently the sparseness of regulatory networks, introducing different ways to limit the number of non-zero interaction parameters (de Hoon et al., 2003). This renders their parameter optimization procedures closer to structure learning methods we employ here. As pointed out by Murphy and Mian (1999), both linear and non-linear models which optimize some error function while learning parameters are actually equivalent to Bayesian network models such as Gaussian networks, even though they are not presented explicitly as probabilistic models.

Several researchers have applied dynamic Bayesian networks to model time series expression data, acknowledging the fact that static Bayesian networks are incapable of modeling feedback cycles, and do not utilize the time dependence between samples in those data sets. Kim et al. (2004) extend the non-parametric model of Imoto et al. (2003) to the dynamic case. Ong et al. (2002) use hidden variables in this context to account for unknown operon structure in their bacterial data. Two recent works are closely related to our dynamical modeling approach: Perrin et al. (2003) and Rangel et al. (2004) both use dynamic models with hidden "regulation" states controlling observed transcription levels. However, the hidden state does not correspond to protein activity levels, but rather to mRNA levels and their derivatives (Perrin et al., 2003) or to an undefined "regulatory" state (Rangel et al., 2004). Moreover, both works end up using a linear regulation model between the hidden and observed variables, and a simple additive noise model. In this sense, they are not as close to real biology as our realistic models, and so interpreting their results is harder. Their modeling choices of course render these models much simpler to solve, borrowing from standard techniques for linear dynamical systems. It would be interesting to compare the performance of our methods against such simplified approaches.

Our realistic approach was motivated by a different path of research, which studies regulatory mechanisms using fine detailed realistic kinetic models, borrowing from a long tradition in enzyme kinetics modeling. McAdams and Arkin (1997) give a detailed model of a single gene's expression pathway, accounting for different sources of stochasticity. Ronen et al. (2002) learn the kinetic parameters for a single repressor regulation module in E. Coli using algebraic methods.

Battogtokh et al. (2002) face a similar task using an ensemble method to estimate error bars over the predicted parameters. These works base their interaction models on the biochemical processes, resulting in non-linear regulatory connections. They do not try to learn the network's structure, however, and use an architecture known in advance from biological literature.

A related question to ours is that of transcription factor binding motif discovery. Since the introduction of microarray methods, researchers used expression data sets to tackle this problem. The first implementations used clustering of gene expression profiles, and then look for motifs within each cluster (Tavazoie et al., 1999; Zhang, 1999). Bussemaker et al. (2001) circumvent the need for clustering by using a model in which the log expression level results from a linear sum of its upstream motifs. Pilpel et al. (2001) look for combinations of known motifs in promoters of genes with a correlated expression pattern, obtaining new knowledge about combinatorial regulation schemes.

In recent years, the combination of different data sources has become popular, leading to works that integrate between questions of sequence motif finding and regulatory connection discovery. For example, Tanay and Shamir (2004) explain transcription levels through a combination of regulator affinities and regulator doses (activity levels) as we do in Chapter 6, using a sequence motif discovery procedure to initialize the regulator levels and affinities. Their approach is somewhat more simplified, as they model functions over discretized levels of activity and affinity, and as they use the probabilistic score of a binding site as an approximation to its affinity level. Segal et al. (2002, 2003b) combine sequence motif models and expression profiles into a unified probabilistic graphical model. They also employ DNA location data either as noisy sensors of regulator-target connections (Segal et al., 2002) or as external validation for unknown regulator identities (Segal et al., 2003b). In Segal et al. (2003a), expression and sequence data are combined to characterize expression modules in a different type of model. Unlike the more realistic models, these types of models capture phenomena in the data without directly modeling the mechanisms that generate it. This might be considered a strength, as they can make useful predictions without employing too strong assumptions on the modeled phenomena. However it is harder to make quantitative predictions in such approaches. As we discuss in the final section, we hope to extend our realistic modeling framework to account for different data sources as well.

7.3 Future Directions

We believe the analysis methods introduced in this thesis hold great promise for elucidating the structure, parameters and eventually the dynamics of gene regulation networks. There are, however, limitations to these methods which should be acknowledged, some of which point at promising future research directions.

7.3.1 Closing the Loop

Our realistic regulation model currently does not account for the sources of regulator activity. In particular, the connection between a regulator's transcription rate at time t and its activity level at time $t+1$ is ignored. Though in many cases, as we pointed out, this connection does not carry any information (e.g. when all the regulator's control happens post-translationally), in some systems (particularly in bacterial ones) this connection is sound and can be modeled in a principled way. For example, when we expect no dynamic regulation at the post translational or degradation levels, we can model the dependence of a gene's protein levels on its mRNA levels using a simple protein production rule:

$$\frac{d}{dt}p_k^{(t)} = \lambda_k e_k^{(t)} - \delta_k p_k^{(t)} \tag{7.1}$$

where p_k and e_k are the protein and mRNA levels of gene k at time t, respectively, λ_k is the gene's protein translation rate and δ_k is the protein degradation rate.

We can benefit a lot by modeling these connections whenever possible. First, they remove artificial independencies present in the current model, and constrain the hidden protein activity levels more correctly during the learning process. Moreover, by closing the loop between mRNA and protein variables we can use the model to predict future dynamics in a better way. We can then study the model's dynamics, through simulation or analysis, getting an additional source for verifying its learned parameters. For example, if the biological system contains feedback with some known dynamic behavior (like cyclic oscillations, stabilizing effects or auto-catalytic behaviour), we expect to obtain this behaviour when running a simulation of the model's dynamics. This simulation is not possible in our current open loop model, as the only dependence between time points in it comes in the form of persistence edges.

7.3.2 Incorporating Different Data Sources

In our first approach we only use expression data as input. In the more realistic approach we also use binding location data for an initial guess of the regulatory network structure. As today we face a multitude of different data sources, from which data sets are accumulating every day, our models can benefit greatly by incorporating different modes of data in a principled manner, much like the heterogeneous-source methods we mentioned in the previous section do. For example, though our use of location data is straightforward, it cannot be justified in the Bayesian framework. A more principled approach would be to treat this data as some noisy sensor for gene regulation, as done in Segal et al. (2002). By this we would treat location data and expression data on equal footing: both are types of observations which are explained by our generative model. One benefit of such an approach is that it obviates the question of how to weigh different types of data when learning a model. The relevant term in the model score is simply the likelihood of all observed data given the model, without any arbitrary weighting factors.

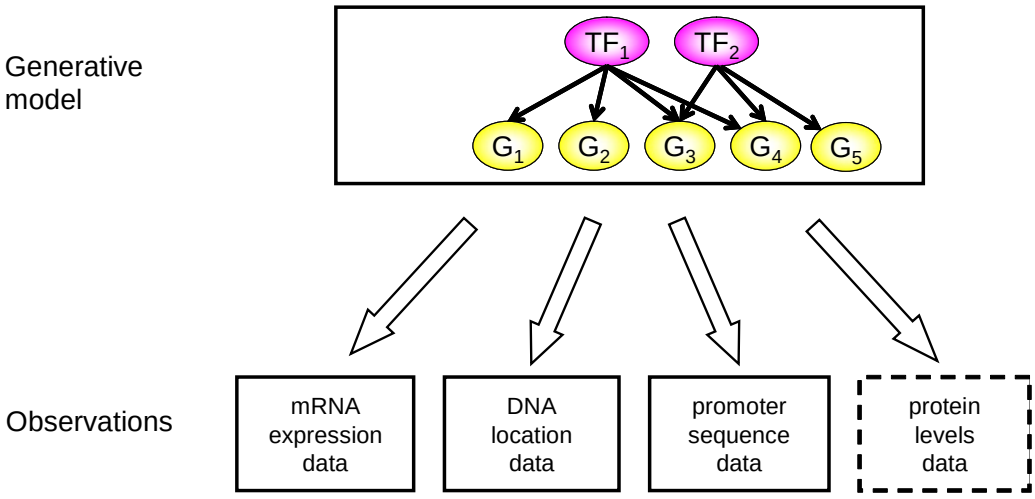


Figure 7.1: A possible approach for incorporating different data sources. A generative model, consisting of regulation network topology and kinetic parameters, provides a description of how observations of different types were generated. Expression, location and sequence data are today commonly available. Protein data will become more common in the future.

Other modes of data can also be incorporated this way. One prevalent source is promoter sequence, or motif occurrence data. Other modes which may be more common in the future are protein quantity measurements or protein activity measurements. Figure 7.1 illustrates the idea of such a scheme. Of course, working out the details of combining these different data sources with our realistic quantitative approach into a single coherent generative model is a challenging task.

7.3.3 Putting Things into Use

The major bulk of this thesis describes methods, approaches and algorithms by which one can gain better understanding of regulatory networks. Far too often in the field of computational biology researchers stop at the phase of demonstrating the power of a certain approach, without doing the extra step of putting their methods into real use, thus producing some new knowledge on biology. This step usually requires wet lab experimentation and can therefore be very time consuming. This is, however, the only way we can translate our efforts to statements such as "The novel factor X represses the group of genes A in the beginning of process P" with confidence. Our major goal now should be getting the methods we describe here to this stage. This requires application of the methods to specific systems (such as the ones we describe in the experiments of Chapter 6), generating testable predictions and collaborating with experimental labs to verify those predictions. Doing this we can make our computational research in the field of biology fulfill its true aim.

Bibliography

Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95). 1995.

Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96). 1996.

Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98). 1998.

Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99). 1999.

Advances in Neural Information Processing Systems 13. MIT Press, Cambridge, Mass., 2001.

Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01). 2001.

H. Akaike. A new look at the statistical idetification model. *IEEE Transactions on Automatic Control*, 19: 716–723, 1974.

S. Akutsu, T. Kuhara, O. Maruyama, and S. Miyano. Indentification of gene regulatory networks by strategic gene disruptions and gene over-expressions. In *SODA*. ACM-SIAM, 1998.

A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, Jr. Hudson, J., L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, L. M. Staudt, and et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403 (6769):503–11, 2000.

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. *PNAS*, 96(12):6745–50, 1999.

O. Alter, P. O. Brown, and D. Botstein. Singular value decomposition for genome-wide expression data processing and modeling. *Proc Natl Acad Sci U S A*, 97(18):10101–10106, Aug 2000.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nuc. Acids Res.*, 25: 3389–3402, 1997.

H. Attias. Inferring parameters and structure of latent variable models by variational bayes. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* uai (1999), pages 21–30.

K. Baetz and B. Andrews. Regulation of cell cycle transcription factor swi4 through auto-inhibition of dna binding. *Mol Cell Biol*, 19(10):6729–41, 1999.

D. Battogtokh, D. K. Asch, M. E. Case, and J. Arnold. An ensemble method for identifying regulatory circuits with special reference to the qa gene cluster of *neurospora crassa*. *Proc Natl Acad Sci U S A*, 99 (26):16904–9, 2002.

I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proc. 2'nd European Conf. on AI and Medicine*. Springer-Verlag, Berlin, 1989.

A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. *Journal of Computational Biology*, 7:559–584, 2000.

A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *J. Comp. Bio.*, 6(3-4):281–97, 1999.

A. Bhattacharjee, W. G. Richards, J. Staunton, C. Li, S. Monti, P. Vasa, C. Ladd, J. Beheshti, R. Bueno, M. Gillette, M. Loda, G. Weber, E. J. Mark, E. S. Lander, W. Wong, B. E. Johnson, T. R. Golub, D. J. Sugarbaker, and M. Meyerson. Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses. *Proc Natl Acad Sci U S A*, 98(24):13790–13795, Nov 2001.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`.

T. Blumenthal. Gene clusters and polycistronic transcription in eukaryotes. *Bioessays*, pages 480–487, 1998.

Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993. URL `citeseer.ist.psu.edu/bodlaender93tourist.html`.

Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Privara and Peter Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295*, pages 19–36, Berlin, 1997. Springer-Verlag. URL `citeseer.ist.psu.edu/bodlaender98treewidth.html`.

J. Boros, F. L. Lim, Z. Darieva, A. Pic-Taylor, R. Harman, B. A. Morgan, and A. D. Sharrocks. Molecular determinants of the cell-cycle regulated Mcm1p-Fkh2p transcription factor complex. *Nucleic Acids Res*, 31(9):2279–2288, May 2003.

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)* uai (1996).

X. Boyen, N. Friedman, and D. Koller. Learning the structure of complex dynamic systems. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* uai (1999).

X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)* uai (1998), pages 33–42.

H.J. Bussemaker, H. Li, and E.D. Siggia. Regulatory element detection using correlation with expression. *Nature Genetics*, 27:167–71, 2001.

P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In Fayyad U., Piatesky-Shapiro G., Smyth P., and Uthuruasmy R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI Press, Menlo Park, CA, 1995.

T. Chen, V. Filkov, and S. Skiena. Identifying gene regulatory networks from experimental data. In *RECOMB 1999*, 1999a.

T. Chen, H. L. He, and G. M. Church. Modeling gene expression with differential equations. *Pac Symp Biocomput*, pages 29–40, 1999b.

J. Cheng, D.A. Bell, and W. Liu. An algorithm for bayesian belief network construction from data. In *Proceedings of AI & STAT*, pages 83–90, Ft. Lauderdale, Florida, 1997.

J. Cheng, G. Grainer, J. Kelly, D.A. Bell, and W. Lius. Learning bayesian networks from data: An information-theory based approach, 2001. URL `citeseer.ist.psu.edu/628344.html`.

D. M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *UAI '95* uai (1995), pages 87–98.

D. M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H.-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*. Springer Verlag, 1996.

D. M. Chickering and D. Heckerman. Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning*, 29:181–212, 1997.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462–467, 1968.

G. Cooper and C. Yoo. Causal discovery from a mixture of experimental and observational data. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* uai (1999), pages 116–125.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

G.F. Cooper and C. Glymour. *Computation, Causation, and Discovery*. MIT Press, 1999.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Mass., 1990.

M.C. Costanzo, M.E. Crawford, J.E. Hirschman, J.E. Kranz, P. Olsen, L.S. Robertson, M.S. Skrzypek, B.R. Braun, K.L. Hopkins, P. Kondu, C. Lengieza, J.E. Lew-Smith, M. Tillberg, and J.I. Garrels. Ypd, pombepd, and wormpd: model organism volumes of the bioknowledge library, an integrated resource for protein information. *Nuc. Acids Res.*, 29:75–9, 2001.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

L. Csato and M. Opper. Sparse on-line gaussian processes. *Neural Comput*, 14(3):641–668, Mar 2002.

F. Cvrckova and K. Nasmyth. Yeast G1 cyclins CLN1 and CLN2 and a GAP-like protein have a role in bud formation. *EMBO. J*, 12:5277–5286, 1993.

P. Dagum and M. Luby. An optimal approximation algorithm for Baysian inference. *Artificial Intelligence*, 93(1–2):1–27, 1997.

M. J. de Hoon, S. Imoto, K. Kobayashi, N. Ogasawara, and S. Miyano. Inferring gene regulatory networks from time-ordered gene expression. *Pac Symp Biocomput*, pages 17–28, 2003.

R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)* uai (1996).

M. H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–39, 1977.

J. DeRisi, V. Iyer, and P. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 282:699–705, 1997.

P. D'Haeseleer, X. Wen, S. Fuhrman, and R. Somogyi. Linear modeling of mRNA expression levels during CNS development and injury. *Pac Symp Biocomput*, pages 41–52, 1999.

A. Doucet, N. de Freitas, and N. Gordon (eds). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proc. Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI '00)*, pages 176–183, 2000a. URL citeseer.nj.nec.com/doucet00raoblackwellised.html.

A. Doucet, S. J. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statist. Comp*, 10:197–208, 2000b.

M. A. Drebot, G. C. Johnston, J. D. Friesen, and R. A. Singer. An impaired RNA polymerase II activity in *saccharomyces cerevisiae* causes cell-cycle inhibition at START. *Mol. Gen. Genet.*, 241:327–334, 1993.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

R. Durrett. *Probablity Theory and Examples*. Wadsworth and Brooks, Cole, California, 1991.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, London, 1993.

M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95(25):14863–8, 1998a.

M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95:14863–14868, 1998b.

G. Elidan and N. Friedman. The information bottleneck EM algorithm. In *Proc. Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI '03)*. 2003.

G. Elidan, N. Lotner, N. Friedman, and D. Koller. Discovering hidden variables: A structure-based approach. In *Advances in Neural Information Processing Systems 13* nip (2001).

K. J. Ezawa and T. Schuermann. Fraud/uncollectable debt detection using a Bayesian network based learning system: A rare binary outcome with mixed data structures. In *UAI '95* uai (1995).

N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *ML '97*, pages 125–133. 1997.

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)* uai (1996).

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 421–460. Kluwer, Dordrecht, Netherlands, 1998.

N. Friedman, M. Goldszmidt, and A. Wyner. Data analysis with Bayesian networks: A bootstrap approach. In *Proc. UAI*, pages 206–215, 1999.

N. Friedman and D. Koller. Being bayesian about network structure. In *Proc. Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI '00)*. 2000.

N. Friedman and D. Koller. Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–126, 2003.

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *J. Comp. Bio.*, 7:601–620, 2000.

N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)* uai (1998), pages 139–147.

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 program in the response of yeast cells to environmental changes. *Mol. Bio. Cell*, 11: 4241–4257, 2000.

D. Geiger and D. Heckerman. Learning gaussian networks. In *UAI '94*, pages 235–243. 1994.

L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI '99*. 1999.

M. N. Gibbs and D. J. C. MacKay. Efficient implementation of gaussian processes. 1997.

M. N. Gibbs and D. J. C. MacKay. Variational Gaussian process classifiers. Unpublished manuscripts, available at http://wol.ra.phy.cam.ac.uk/mackay, 1997.

W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov Chain Monte Carlo Methods in Practice*. CRC Press, 1996.

V. Guacci, D. Koshland, and A. Strunnikov. A direct link between sister chromatid cohesion and chromosome condensation revealed through the analysis of MCD1 in *s. cerevisiae*. *Cell*, 91(1):47–57, October 1997.

C. C. Guet, M. B. Elowitz, W. Hsing, and S. Leibler. Combinatorial synthesis of genetic networks. *Science*, 296(5572):1466–70, 2002.

F. Harary and E. M. Palmer. *Graphical Enumeration*. Academic Press, NY, 1973.

A. Hartemink, D. Gifford, T. S. Jaakkola, and R. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. In *Pac. Symp. Biocomp. 6*, 2001.

A. J. Hartemink, D. K. Gifford, T. S. Jaakkola, and R. A. Young. Combining location and expression data for principled discovery of genetic regulatory network models. *Pac Symp Biocomput*, pages 437–449, 2002.

P. A. Haynes and J.R. 3rd Yates. Proteome profiling-pitfalls and progress. *Yeast*, 17(2):81–87, Jun 2000.

D. Heckerman and D. Geiger. Likelihoods and parameter priors for bayesian networks. Technical report, 1995. Technical Report MSR-TR-95-54, Microsoft Research.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995a.

D. Heckerman, A. Mamdani, and M. P. Wellman. Real-world applications of Bayesian networks. *Communications of the ACM*, 38, 1995b.

D. Heckerman, C. Meek, and G. Cooper. A Bayesian approach to causal discovery. In *Computation, Causation, and Discovery* Cooper and Glymour (1999), pages 141–166.

A.V Hill. The combinations of haemoglobin with oxygen and with carbon monoxide. *Biochem. J.*, 7:471–480, 1913.

R. Hofmann and V. Tresp. Discovering structure in continuous variables using bayesian networks. In *Advances in Neural Information Processing Systems 9*. MIT Press, Cambridge, Mass., 1996.

P. C. Hollenhorst, M. E. Bose, M. R. Mielke, and C. A. Fox. Forkhead genes in transcriptional silencing, cell morphology and the cell cycle. overlapping and distinct functions for fkh1 and fkh2 in saccharomyces cerevisiae. *Genetics*, 154(4):1533–48, 2000.

F. C. Holstege, E. G. Jennings, J. J. Wyrick, T. I. Lee, C. J. Hengartner, M. R. Green, T. R. Golub, E. S. Lander, and R. A. Young. Dissecting the regulatory circuitry of a eukaryotic genome. *Cell*, 95(5):717–28, 1998.

S. Imoto, S. Kim, T. Goto, S. Miyano, S. Aburatani, K. Tashiro, and S. Kuhara. Bayesian network and nonparametric heteroscedastic regression for nonlinear modeling of genetic network. *J Bioinform Comput Biol*, 1(2):231–252, Jul 2003.

R. A. Irizarry, B. M. Bolstad, F. Collin, L. M. Cope, B. Hobbs, and T. P. Speed. Summaries of Affymetrix GeneChip probe level data. *Nucleic Acids Res*, 31(4):e15, Feb 2003.

F. V. Jensen. *An introduction to Bayesian Networks*. University College London Press, London, 1996.

F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 5(4):269–282, 1990.

T. Jochims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *ICML*, 1997.

M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational approximations methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

S. Kalir, J. McClure, K. Pabbaraju, C. Southward, M. Ronen, S. Leibler, MG. Surette, and U. Alon. Ordering genes in a flagella pathway by analysis of expression kinetics from living bacteria. *Science*, 292:2080–2083, 2001.

S. A. Kauffmann. *The Origins of Order*. Oxford University Press, New York, 1993.

S. Kim, S. Imoto, and S. Miyano. Dynamic Bayesian network and nonparametric regression for nonlinear modeling of gene networks from time series gene expression data. *Biosystems*, 75(1-3):57–65, Jul 2004.

S. Y. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic bayesian networks. *Brief Bioinform*, 4(3):228–35, 2003.

H. Kitano. Computational systems biology. *Nature*, 420(6912):206–10, 2002.

D. Koller, U. Lerner, and D. Angelov. A general algorithm for approxiamte inference and its application to hybrud bayes nets. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* uai (1999).

S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17:31–57, 1989.

N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems 15*. MIT Press, Cambridge, Mass., 2003.

TI Lee, NJ Rinaldi, F Robert, DT Odom, Z Bar-Joseph, GK Gerber, NM Hannett, CT Harbison, CM Thompson, I Simon, J Zeitlinger, EG Jennings, HL Murray, DB Gordon, B Ren, JJ Wyrick, JB Tagne, TL Volkert, E Fraenkel, DK Gifford, and RA Young. Transcriptional regulatory networks in *saccharomyces cerevisiae*. *Science*, 298:799–804, 2002.

U. Lerner, E. Segal, and D. Koller. Exact inference in networks with discrete children of continuous parents. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)* uai (2001).

U. N. Lerner. *Hybrid Bayesian Networks for Reasoning about Complex Systems*. PhD thesis, Dept. of Computer Science, Stanford University, 2002.

C. Li and W. H. Wong. Model-based analysis of oligonucleotide arrays: expression index computation and outlier detection. *Proc Natl Acad Sci U S A*, 98(1):31–36, Jan 2001.

S. Liang, S. Fuhrman, and R. Somogyi. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. *Pac Symp Biocomput*, pages 18–29, 1998.

J. C. Liao, R. Boscolo, L. M. Tran, C. Sabatti, and V. P. Roychowdhury. Network component analysis: reconstruction of regulatory signals in biological systems. *Proc Natl Acad Sci U S A*, 100(26):15522–7, 2003.

D. J. Lockhart, H. Dong, M. C. Byrne, M. T. Follettie, M. V. Gallo, M. S. Chee, M. Mittmann, C. Want, M. Kobayashi, H. Horton, and E. L. Brown. DNA expression monitoring by hybridization of high density oligonucleotide arrays. *Nature Biotechnology*, 14:1675–1680, 1996.

D. J. C. MacKay. Introduction to Gaussian processes. Unpublished manuscripts, available at http://wol.ra.phy.cam.ac.uk/mackay, 1998.

D. Madigan and E.E. Raftery. Model selection and accounting for model uncertainty in graphical models using Occam's window. *Journal Americal Statistical Association*, 89:1535–1546, 1994.

D. Madigan and J. York. Bayesian graphical models for discrete data. *International statistical Review*, 63: 215–232, 1995.

J. Martin and K. VanLehn. Discrete factor analysis: Learning hidden variables in Bayesian networks. Technical report, Department of Computer Science, University of Pittsburgh, 1995.

H. H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proc Natl Acad Sci U S A*, 94(3): 814–9, 1997.

H. J. McBride, Y. Yu, and D. J. Stillman. Distinct regions of the swi5 and ace2 transcription factors are required for specific gene activation. *J Biol Chem*, 274(30):21029–36, 1999.

P. McCullagh and J.A. Nelder. *Generalized Linear Models*. Chapman & Hall, London, 1989.

W. G. McGregor. DNA repair, DNA replication, and UV mutagenesis. *J. Investig. Dermatol. Symp. Proc.*, 4: 1–5, 1999.

G. S. Michaels and et al. Cluster analysis and data visualization for large scale gene expression data. In *Pac. Symp. Biocomputing*, pages 42–53. 1998.

Thomas P. Minka. Pathologies of orthodox statistics. Available from `http://www.stat.cmu.edu/~minka/papers/pathologies.html`, 1998.

E. Mjolsness, D.H.Sharp, and J. Reinitz. A connectionist model of development. *J. Theor. Biol.*, 152:429–453, 1991.

A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1997.

K. Murphy and S. Mian. Modeling gene expression data using dynamic bayesian networks. Technical report, 1999. Available at http://www.cs.berkeley.edu/ murphyk/publ.html.

K. Murphy and Y. Weiss. Loopy belief propagation for approximate inference: An empirical study. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* uai (1999).

K. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in dbns. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)* uai (2001), pages 378–385.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

I.M. Ong, J.D. Glasner, and D. Page. Modelling regulatory pathways in *e. coli* from time series expression profiles. *Bioinformatics*, 18(Suppl 1):S241–248, 2002.

E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge Univ. Press, 2000.

J. Pearl and T. S. Verma. A theory of inferred causation. In *KR '91*, pages 441–452. 1991.

D. Pe'er. *From Gene Expression to Molecular Pathways*. PhD thesis, School. of Computer Science and Engineering, Hebrew University, 2003.

D. Pe'er, A. Regev, G. Elidan, and N. Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17(Suppl 1):S215–24, 2001.

D Pe'er, A Regev, and A Tanay. Minreg: Inferring an active regulator set. *Bioinformatics*, 18 Suppl 1: S258–S267, 2002.

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 Suppl 2:II138–II148, 2003.

Y. Pilpel, P. Sudarsanam, and G.M. Church. Identifying regulatory networks by combinatorial analysis of promoter elements. *Nature Genetics*, 29:153–9, 2001.

T. Pramila, S. Miles, D. GuhaThakurta, D. Jemiolo, and L. L Breeden. Conserved homeodomain proteins interact with MADS box protein Mcm1 to restrict ECB-dependent transcription to the M/G1 phase of the cell cycle. *Genes Dev*, 16(23):3034–45, 2002.

C. Rangel, J. Angus, Z. Ghahramani, M. Lioumi, E. Sotheran, A. Gaiba, D. L. Wild, and F. Falciani. Modeling T-cell activation using gene expression profiling and state-space models. *Bioinformatics*, 20(9):1361–1372, Jun 2004.

C. E. Rasmussen. *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*. PhD thesis, 1996.

J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, River Edge, NJ, 1989.

M. Ronen, R. Rosenberg, B. I. Shraiman, and U. Alon. Assigning numbers to the arrows: Parameterizing a gene regulation network by using accurate expression kinetics. *PNAS*, 99(16):10555–10560, 2002. URL http://www.pnas.org/cgi/content/abstract/99/16/10555.

F. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:832–837, 1956.

M. Sahami. Learning limited dependence bayesian classifiers. pages 335–338, 1996.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

E. Segal, Y. Barash, I. Simon, N. Friedman, and D. Koller. From promoter sequence to expression: A probabilistic framekwork. In *RECOMB'02*. 2002.

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. *Nat Genet*, 34(2):166–176, Jun 2003a.

E. Segal, R. Yelensky, and D. Koller. Genome-wide discovery of transcriptional modules from DNA sequence and gene expression. *Bioinformatics*, 19 Suppl 1:i273–i282, 2003b.

SS Shen-Orr, R Milo, S Mangan, and U Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31:64–8, 2002.

I. Simon, J. Barnett, N. Hannett, C.T. Harbison, N.J. Rinaldi, T.L. Volkert, J.J. Wyrick, J. Zeitlinger, D.K. Gifford, T.S. Jaakkola, and R.A. Young. Serial regulation of transcriptional regulators in the yeast cell cycle. *Cell*, 106:697–708, 2001.

A. Smola and P. Bartlett. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems 13* nip (2001).

R. Somogyi, S. Fuhrman, M. Askenazi, and A. Wuensche. The gene expression matrix: Towards the extraction of genetic network architectures. In *WCNA96*, 1996.

E. L. Sonnhammer, S.R. Eddy, E. Birney, A. Bateman, and R. Durbin. Pfam: multiple sequence alignments and hmm-profiles of protein domains. *Nuc. Acids Res.*, 26:320–322, 1998.

P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Mol. Biol. Cell*, 9(12):3273–97, 1998.

D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20:579–605, 1990.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. Springer Verlag, 1993.

P. Spirtes, C. Meek, and T. Richardson. An algorithm for causal inference in the presence of latent variables and selection bias. In *Computation, Causation, and Discovery* Cooper and Glymour (1999), pages 211–252.

A. Tanay and R. Shamir. Multilevel modeling and inference of transcription regulation. *J Comput Biol*, 11 (2-3):357–375, 2004.

S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church. Systematic determination of genetic network architecture. *Nat Genet*, 22(3):281–5, 1999.

B. Thiesson, C. Meek, D. M. Chickering, and D. Heckerman. Learning mixtures of Bayesian networks. In *Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)* uai (1998).

S. Tornaletti and P. C. Hanawalt. Effect of DNA lesions on transcription elongation. *Biochimie*, 81:139–146, 1999.

V. Tresp. Mixtures of gaussian processes. In *Advances in Neural Information Processing Systems 13* nip (2001).

G. Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, 1990.

G. Wahba. Multivariate model building with additive, interaction, and tensor product thin plate splines. pages 491–504, 1991.

G. Wahba. An introduction to model building with reproducing kernel hilbert spaces. Technical Report TR 1020, Univ. of Wisconsin, 2000.

Y. Wang, C. L. Liu, J. D Storey, R. J. Tibshirani, D. Herschlag, and P. O. Brown. Precision and functional specificity in mRNA decay. *Proc Natl Acad Sci U S A*, 99(9):5860–5, 2002.

D. Weaver, C. Workman, and G. Stormo. Modeling regulatory networks with weight matrices. In *Pac. Symp. Biocomputing*, pages 112–123, 1999.

X. Wen, S. Furhmann, G. S. Micheals, D. B. Carr, S. Smith, J. L. Barker, and R. Somogyi. Large-scale temporal gene expression mapping of central nervous system development. *PNAS*, 95:334–339, 1998.

J. Wilkinson. *The Algebric Eigenvalue Problem*. Claderon Press, Oxford, 1965.

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, Mass., 1996.

C. K. I. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13* nip (2001).

L. Xu and M.I. Jordan. On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, 8:129–151, 1996.

G. Yona, N. Linial, and M. Linial. Protomap - automated classification of all protein sequences: a hierarchy of protein families, and local maps of the protein space. *Proteins: Structure, Function, and Genetics*, 37: 360–378, 1998.

M. Q. Zhang. Promoter analysis of co-regulated genes in the yeast genome. *Comput Chem*, 23(3-4):233–250, Jun 1999.

N.L. Zhang. Hierarchical latent class models for cluster analysis. pages 230–237, 2002.