Small Gene Networks: Finding Optimal Models

Shivani Puri

September 2004

Masters of Science

School Of Informatics

University of Edinburgh

Abstract

Genetic networks help in identifying interactions between genes, and provide information about the function role of individual genes in the cellular system. In this thesis, we have employed a Bayesian framework to learn network structures from microarray data, and implemented an algorithm that uses dynamic programming to find the optimal gene network model for a small number of genes.

To test its performance, we applied the method to two distinctive synthetic datasets. ROC graphs were used to evaluate the effects of noise and a small number of samples, features that are known to be characteristic of gene expression datasets. Results showed significant improvements when the numbers of samples in a dataset were increased. The effect of adding noise to the data gave unexpected results and requires further analysis. The method was finally applied to a real microarray dataset, and led to biologically plausible results.

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Stuart Aitken for his skillful guidance and help throughout the project. I need to thank Sascha Ott for answering my many questions, and for providing me with the script to generate data for testing my implementation. I would like to thank EPSRC, who has funded my thesis and Masters degree.

Finally, I would like to thank all my friends in family who have been extremely supportive, and kept me motivated.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Shivani Puri)

Table of Contents

1 Introduction

All of the genetic information in any living creature is stored in DNA (deoxyribonucleic acid). The double helix structured molecule is functionally divided into information units called *genes* that contain instructions to make proteins, which play a vital role in all cellular processes. The Human Genome Project, initiated in 1990, aimed to sequence the entire human genome and catalogue all human genes. The draft human genome was published in 2000, and genomes of several other organisms, for example, E.coli, mouse, chicken, and yeast have also been completed. Due to the availability of this vast amount of sequencing information, the main focus of genomic research is switching from sequencing to using the genome sequences in order to understand how genomes function [Brazma & Vilo 2000].

To retrieve the information encoded in the gene, cells use the process of *gene expression*. According to the central 'dogma of molecular biology', this occurs in two steps. Genes are *transcribed* into messenger RNA (mRNA), which is then *translated* into proteins. A gene is *expressed* in a cell if its corresponding proteins are present in the cell. "Functioning of different cells or an organism as a whole, to a large extent is governed by the 'selective' expression of genes" [Fuente et al., 2002]. Although all cells in an organism contain more or less the same DNA, i.e. the same genetic material, we observe specialized behaviour from different cell types. For example, the function of a liver cell differs from that of a skin cell. This occurs due to differences in gene expression, that is, whether or not a product a gene codes for is produced, and how much is produced. For a cell to function properly, the proteins being synthesized must be controlled, the amount being produced changing as per the cell's needs. Genetic regulation ensures this. Levels of gene products do change during development, cell differentiation, the onset of disease and in response to the environment [Hunter 1993].

Gene expression levels are regulated in humans by control mechanisms at several levels in the steps between transcription and protein synthesis. Regulation at the transcriptional level is most important since it determines the amount of mRNA to be made. The process is controlled by regulatory proteins (transcription factors) that bind to *cis*-regulatory elements (promoters or enhancers) in special regions of the gene.

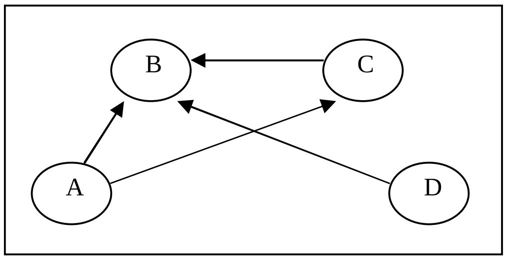
Since transcription factors are also products of transcription and translation, we reason that genes interact with and influence each other (induce or repress) by controlling each other’s expression levels. To gain a good level of understanding of how cells work, we need to therefore discover how specific genes are regulated, study gene – gene interactions and attempt to reconstruct an accurate model of the gene interaction network.

This is a difficult task mainly because of the sheer number of interactions involved. Traditional biology approaches are unable to search through every possibility to identify the genes involved in a particular process. The number of experiments necessary to build a gene network model is a lot more than is possible in ‘wet’ labs [Dutilh & Hogeweg 1999]. Due to this, research focus has been on studying single genes, proteins or single reactions, looking at direct correlations between the genes and phenotypes to determine gene function. A full breadth of important roles for well-known, highly characterized genes have been discovered in this manner, but a need for methods that could provide a wider experimental perspective on how genes interact was also recognized [Kim et al., 2000].

With the advent of high-throughput microarray technology, this has been made possible, and our ability to explore gene interactions has increased dramatically. Gene expression data is now available on a large (genome-wide) scale since it is possible to measure expression levels of all genes of a given organism, at a number of time points, or under various conditions. “The datasets provide snapshots of the molecular state of cell populations at the transcript level and are rich in information about gene networks. It seems logical that these data are the best to uncover gene networks, and indeed this strategy is presently the most widely adopted” [Brazhnik et al., 2002]. Researchers have proposed numerous computational methods for inferring gene regulatory networks from this huge amount of data. In this thesis, we concentrate on one particular method, and evaluate its performance on synthetic and real biological data.

1.1 Gene Networks

Gene networks are models that display causal relationships between gene activities, usually at the mRNA level, and are commonly represented as directed graphs (Figure1a). A directed graph N is defined as a tuple $\langle V, E \rangle$ with V being a set of vertices and E a set of edges. A directed edge is a tuple $\langle i, j \rangle$ of vertices, where i is the head (child) and j is the tail (parent) of the edge. The vertices/nodes correspond to genes and the edges denote interactions between the connected genes. More information about genes and their interactions can be shown, for instance, by labelling the edges with sign +, or – to indicate whether a gene i is activated or inhibited by gene j.



	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	1	0	0
D	0	1	0	0

(a) **(b)**

Figure 1: (a) Graph representation of a gene network. (b) Matrix representation of a gene network. 1 represents an interaction, while 0 represents no interaction.

Gene networks can also be represented as matrices. Each column and row of the matrix represents one gene, while matrix elements represent relationships. The matrices can be qualitative, where positive interactions (activation) are denoted by 1, negative interactions (inhibition) with –1, and 0 for the case of no interactions between genes [Brazhnik et al., 2002]. In some cases, the elements only take on two values; 1 represents an interaction (positive or negative), and 0 represents no interaction (Figure 1b). Quantitative representations indicate the sign as well as the strength of the gene interaction by using real values in the matrix.

Both of the above representations show which genes are interacting with the others in the network. How changes in expression of the 'originating gene' affect the 'target gene' in the network is still not explained. For this, some kind of function at each node in the network is used. The type of function depends on the network model. Bayesian networks, for example, use conditional probability distributions to explain gene interactions, while Boolean networks make use of Boolean logical rules.

Inferring the biological gene network from a large amount of experimental data, as obtained from microarrays is a difficult problem. There are a number of factors that complicate the modelling of gene networks.

1.1.1 Microarray Datasets

Microarray analysis provides a global picture of gene expression for the genome by revealing which genes are expressed at a particular stage of the cell cycle or development cycle of an organism, or genes that respond to a given environment signal to the same extent. A large amount of experimental data is available at the mRNA level as a result. From a biological point of view, it would be preferable to use protein levels rather than mRNA levels to describe gene expression since proteins are the ultimate products of a gene, not mRNA. However, we know that transcription is the first step in gene regulation, so information from transcript levels will be useful for understanding gene networks. Furthermore, correlation studies between mRNA and protein abundance in a cell, such as done by Greenbaum et al. in 2003, imply that high levels of mRNA in a cell is likely to correspond to a large presence of the respective protein, and vice versa. Since it is much easier to measure mRNA levels, and on a much larger scale than measuring protein levels directly, gene expression data is currently being modelled using mRNA abundance level data.

A microarray is a glass slide, onto which single-stranded DNA molecules are attached at fixed locations, called *spots*. Such a microarray may consist of thousands of spots, each representing genes, fragments of genes, or ESTs (Expressed Sequence Tags). The central principle of the microarray-technique is *hybridization*, the selective binding of complementary single-stranded nucleic acid sequences. There are several ways of using microarray technology. A popular method is to compare the mRNA levels from two different samples, for example a healthy and a tumour affected cell. mRNA from both samples are extracted and labelled with two different fluorescent labels, red for healthy sample and green for the tumour sample. They are then mixed and washed over the surface of the microarray, and allowed to hybridize to their complementary sequences in the spots [Kerr et al., 2001].

To quantify the hybridization level of each spot in the array, a laser is used to excite the fluorescent tags, and a photo detector measures the amount of light with the label specific wavelength that is emitted. If the mRNA from the tumour cell is in abundance, the spot will be green, if the healthy cell's mRNA is in abundance, the spot will be red. If mRNA from both cells binds equally, the spot will be yellow, while if neither binds, there will be no fluorescence and the spot will be black (Figure 2). Thus, by using this technique we are able to determine the relative levels of mRNA associated with a huge number of genes in a single experiment.

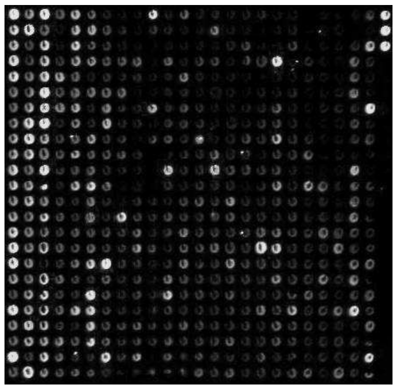


Figure 2: A sample image obtained from a microarray experiment. The intensity and colour of each spot encode information on a specific gene from the tested sample. Figure taken from Gene-Chips [http://www.gene-chips.com/sample1.html]

Measuring expression levels of genes in an organism under various conditions ultimately helps in building 'gene expression profiles' that characterize the dynamic functioning of each gene in an organism's genome. The expression data can be represented using a 'gene expression matrix' with rows representing genes, columns representing samples, and each matrix cell containing a number characterizing the expression level of the particular gene in the particular sample. However, when analysing this gene expression matrix, we need to keep in mind that at each stage in the microarray process: sample extraction, fluorescence labeling, hybridization, and image processing of the two fluorescent light signals, a number of errors are introduced. Software is used to minimise the effect of the errors at each level but the resulting dataset is still very noisy, and may contain distorted gene expression levels.

Furthermore, modelling of gene networks from microarray datasets is presented with another problem, namely the curse of dimensionality. The number of samples/measurements (usually dozens) made in microarray experiments is far less than the number of genes (tens of thousands) whose expression levels have been measured. This basically means that there is not enough data to infer the complete underlying gene biological network. Ideally, we would need to collect as many measurements/samples as the number of genes.

In a recent study, Husmeier [2003] tried to quantify experimentally exactly how much can be learned from the data in this unfavourable situation. Gene expression data was simulated from a realistic biological network, and interactions inferred using a *Bayesian model*, which are described in detail in Chapter 2. The simulation results were presented as receiver operator characteristics (ROC) curves, and demonstrated that increasing the training set i.e. the number of samples in the dataset resulted in an increase in the number of true interactions, and a decrease in the number of spurious interactions being inferred.

1.1.2 Properties of Biological Networks

Biological systems exhibit certain characteristics that need to be taken into account when one tries to reconstruct a genetic network model. Some of them are outlined below.

Stochastic Nature

Genes in some pathways evolve rapidly, while others do not; some processes are highly sensitive to mutations and external stimuli, while others vary little despite significant pressures; some individuals with mutations are affected with disease while others with the same mutation are healthy. Some of the above phenomena can be explained away by the complex and stochastic nature of gene network and biological systems in general.

A stochastic process is one governed by a random process, and in a biological context this means that the system is subject to fluctuations. With respect to gene networks, “Stochasticity allows significant variations in the sequence of activation and inactivation of genes” explains Szallasi [1999]. In such a system, a given gene-expression state can generate more than one successive gene-expression states, and therefore, different cells of the same population may follow a different gene-expression path from one state of gene-expression to another.

Robustness

Biological systems are dynamic, with gene interactions taking place at different points in space and time. The network is continuously evolving under effects of mutation, environmental effects and other perturbations. Robustness ensures that the gene network is able to cope with changes, adapt to them and restore stability. Some of the mechanisms that allow this are the existence of duplication in the genome (redundancy), feedback control (positive and negative) and degeneracy – the ability of different elements in the network to perform the same function or yield the same output. Gene network models should try and incorporate these features, trying to ensure that sensitivity to small variations in network parameters are reduced.

Modularity

Modularity is a highly characteristic feature of biological networks according to Alon [2003]. He describes modules as “set of nodes that have strong interactions and a common function. A module has defined input nodes and output nodes that control the interactions with the rest of the network. A module also has internal nodes that do not significantly interact with nodes outside the module”. Modular networks can change with time and adapt to new conditions, while a non-modular system, in which every component is optimally linked to every other component, is effectively frozen and cannot evolve to meet new optimisation conditions. What this essentially implies is that the connections within a gene network are sparse – genes are unlikely to be linked with every other gene in the network. Sub networks consisting of small number of densely connected genes work on a local level to propagate changes through the network to yield a global response. In view of this, gene networks might be approached by first modelling parts of the network; fully derive complete connectivity within, and then look at interactions between the different sub networks.

1.2 Gene Network Models

There is absolutely no doubt that biological networks exhibit behaviour that is very complex. Existing methods try to somehow abstract this complexity, at least at some levels, by introducing assumptions to simplify their models. It can be argued that even if we are unable to model all the parameters or variables involved in a realistic gene network, we could observe the average overall behaviour of our model, and predict, for instance how a network of genes would react to an external stimuli.

This section aims to provide an overview of several ‘popular’ gene network models. Advantages and disadvantages of each model are discussed with respect to how well it contends with the noisy nature and dimensionality problem of microarray datasets, and the extent to which it is able to reflect the complexities of the underlying biological network.

1.2.1 Weight Matrices

A weight matrix model considers the interactions between all combinations of genes, by using a weight matrix, W, where each row of the matrix represents all the inputs for one gene. The effect of gene i on gene j is simply the expression level of i multiplied by its influence on j, the weight W_{ij}. The total influence exerted on a specific gene j can then be calculated by summing the influences of all genes in the system, and then normalizing to produce an expression value between 0 and 1.

The model is advantageous in the sense that each gene can have multiple inputs, both positive and negative, with varying strengths. It allows us to study many different kinds of interactions, as found in real biological systems. Weaver et al. [1999] for example, investigated the effects of various environmental variables on the network model, and observed various alterations in gene expression patterns, one of which was the classic transition of periodic gene expression to a stable gene expression pattern. They state many advantages of the model, but also discuss several limitations. “These models make assumptions about the behaviour of regulatory systems that are known to be untrue. For example, the assumption that all genetic interactions can be treated as independent events is contradicted by known transcriptional regulators that have different activities depending on their protein partners”.

1.2.2 Boolean Networks

A Boolean network considers each node representing a gene as a binary variable, which can either be expressed or repressed (node has state 1 or 0, respectively). The dynamics of the network are determined by a list of n Boolean functions which each receive input from k specified nodes. Every node has its own specific function, which can determine its next state from the current states of all the input nodes. A sequence of states connected by transitions forms a trajectory of the system, with all initial states eventually reaching a steady state (point attractor) or a state cycle (dynamic attractor) [D'Haeseleer et al., 1999a].

"Boolean networks allow large gene networks to be analysed in an efficient way, by making *strong simplifying assumptions* on the structure and dynamics of a genetic regulatory system" says De Jong [2002]. While nodes in a Boolean model take binary values (genes are considered either 'on' or 'off'), which are updated synchronously, quantities of gene expressions in real networks are not binary and change continuously with time [Akutsu et al., 1999]. Furthermore, for computational reasons, most Boolean model networks are designed such that all genes can be controlled by k, a fixed maximum number of other genes in the network. This does not truly reflect the complexities of gene networks, where there is a great degree of variation in the number of genes controlling a specific gene. Some genes are known to have many regulatory inputs while others have but a few interactions.

Despite their simplicity, Boolean network models have been used extensively in the past as conceptual tools for investigating the principles of a gene network – its structure, organisation and dynamics. Studies on attractor states and trajectories, for instance, have confirmed biological network characteristics such as stability and robustness. Constraints in the number of inputs and outputs per gene, input and output sharing among genes evolved within a gene family or pathway, and restrictions on rule types (thresholding, no "exclusive or" rules etc) have also been discovered through simulations of Boolean nets [D'Haeseleer et al. 1999b]. Ultimately, these models serve as good starting points for investigation of gene networks. For a more realistic approach however, better methods are required.

1.2.3 Differential Equations

Differential equation methods have been used widely to model gene networks [Chen et al., 1999b; De Hoon et al., 2002; 2003]. Gene interactions and regulation are based on rate equations, which express the rate of production of a component as a function of the concentration of other components in the system [De Jong 2002]. The models allow gene regulation to be described in great detail to the level of individual reaction steps. Hartemink et al. [2001] remark, "While such low-level dynamics are critical to a complete understanding of regulatory networks, they require detailed specifications of both the relationship between the interacting agents as well as the parameters of the biochemical reaction, like reaction rates, diffusion constants, etc.".

Finding appropriate parameter values that fit the data is very difficult, and is the model's greatest drawback. The approach is therefore restricted to very small systems. Chen et al. [1999b] proposed a differential equation model for gene expression, and constructed their model using temporal expression data. Both transcription and translation processes by kinetic equations with feedback loops from translation products to transcription were modelled, and the parameters solved using **linear algebra**. The model was specific to time expression data, and was unable to work on mRNA expression level data alone. It required knowledge of protein levels at different time steps also. Linearity in the model allowed parameters to be found efficiently, but was unable to model important non-linear gene interactions. To tackle this weakness, non-linear rate equations have also been used to model gene regulation. Although the models are more realistic, they are computationally very expensive, and require a larger number of parameters to be approximated than linear models [Szallasi 2001]. Another approach worth mentioning is of models that incorporate stochasticity into their methods. As we know, gene regulatory interactions are best described as stochastic processes. Models that use stochastic differential equations are thus better than those that make assumptions of concentrations of substances varying continuously and deterministically [De Jong 2002]. Again however, these stochastic models are too computationally expensive in terms of approximating parameters and fitting them to data.

Problems regarding the measurements of the numerous kinetic parameters can be to some extent solved with the growing availability of gene expression data obtained through microarray technology. However, one would need to solve the dimensionality problem (See 1.1.1) since for reliable estimation, we would need the number of measurements or data points to exceed the number of parameters. Two methods have tried to solve the problem. One method clusters genes with similar expression profiles, so as to reduce the number of parameters in the model, while the other tries to increase the number of measurements by interpolating data points [D'Haeseleer et al., 1999a]. Both approaches make strong assumptions that can lead to mistaken conclusions. It is thus wise to use differential equation methods once we have enough knowledge, and appropriate data.

1.2.4 Clustering

Clustering works by partitioning genes and/or samples into groups by applying some kind of similarity measure on gene expression data. Euclidean distance, linear correlation, rank correlation and mutual information are most popular. Clusters are formed such that there is high correlation among elements within a cluster, and low correlation between elements from different clusters. Szallasi [2001] explains that a high correlation between two genes i.e. both exhibiting similar expression profiles most probably signifies that they are part of the same regulatory process and are functionally related.

According to D'Haeseleer et al. [1999b], clustering is potentially useful in at least three areas. Functions of unknown genes may be inferred by studying genes with known function in the same cluster. Secondly, instead of grouping genes, we could search for clusters of microarray samples/experiments that are highly correlated over a subset of genes. This would help in classification of different cell types. For instance, Golub et al. [1999] performed clustering on gene expression data obtained from human leukaemia patients. They were able to discover new tumour classes, and predict the cancer type of incoming leukaemia patients from the clusters found. Finally, by combining clustering methods with sequence analysis, studies such as that of Brazma et al [1998] have been able to determine common regulatory factors for a set of co-regulated (highly – correlated) genes.

Despite their many advantages, clustering techniques are limited. A gene is allocated to a single cluster that is associated with performing one biological function. This is inappropriate since we know that a gene can perform multiple functions, and is controlled by several genes through a variety of regulatory elements. Thus, methods should allow genes to belong to more than one cluster. Moreover, similarity is quantified using a 'global' correlation measure, which may cause relations that only exist over a subset of the data to remain unidentified.

Most importantly, clustering can only go far as identifying which genes are co-regulated. It does not lead to a fine resolution of the interaction processes. I like to regard clustering as a useful pre-processing step for inferring gene regulatory networks. Once we are able to determine a group of genes that most probably, share a biological function, we can perform further analysis to elucidate finer structure and relations between the genes within. We can hope to answer questions such as, is the effect of one gene on another direct, or mediated by other genes? Which genes mediate the interactions within a cluster of genes or between clusters? What is the nature of the interaction between genes? [Pe'er et al., 2001]

1.2.5 Bayesian Networks

A Bayesian network is a probabilistic graph model, which describes the multivariate probability distribution for a set of variables [Pearl 1988; Heckerman 1998]. When applied to our problem of genetic networks, the expression level of each gene is treated as a random variable, and regulatory interactions as probabilistic dependencies. The joint distribution over the set of all genes then reflects the distribution of cell 'states' and how these affect expression levels [Pe'er et al., 2001]. Bayesian learning techniques try to estimate and understand the network structure that best describes this distribution with respect to the data. They are able to capture complex relationships between genes by extracting information about their (conditional) dependencies and independencies encoded in the high-dimensional microarray datasets.

The work of Friedman et al. [2000] was the first to use Bayesian networks for analysing gene expression data. They discovered that the models took advantage of the modularity characteristic in biological networks (See 1.1.2) i.e. they were particularly useful for describing processes composed of locally interacting components. Their method demonstrated that genes could be modelled as discrete or continuous variables but was limited to detecting linear interactions between the genes. Imoto et al. [2002, 2003a] extended the above work, and constructed a genetic network from expression data by using a nonparametric regression strategy in conjunction with a Bayesian network framework. Their method was successful in capturing even nonlinear relationships between the genes. Another extension by Hartemink et al. [2001] incorporated hidden variables in the Bayesian framework to capture unobserved factors in the data, and was able to describe gene interactions at varying levels of refinement.

There are a number of reasons why Bayesian networks are becoming increasingly popular as methods to infer genetic networks. First of all, the models are used to capture causal relationships within the data, leading us to make conclusions that are biologically meaningful. Secondly, the statistical framework of Bayesian learning is designed for domains with a large number of variables [Dejori 2002], and is therefore ideal for modelling interactions of the huge number of genes found in any biological network. The probabilistic nature of Bayesian models are able to deal with the stochastic aspects of gene expression and noisy measurements in a natural way [De Jong 2002], and allow the confidence in the inferred network structures to be estimated objectively [Husmeier 2003]. Since they discover dependencies among all variables, the models are even able to handle incomplete datasets.

Another advantage of the Bayesian approach, as mentioned by Heckerman [1998] in his tutorial on Bayesian Networks, is the ability to combine prior knowledge with the information extracted from data. Prior or domain knowledge is crucially important if one performs a real-world analysis, he says, in particular, when data is inadequate or expensive. Studies by Hartemink et al. [2002] and Segal et al. [2002] used binding site information as priors to improve their Bayesian models. Imoto et al. [2003b] went further and used a large range of biological knowledge, such as protein – protein interactions, protein – DNA interactions, binding site information, existing literature etc with their microarray data. They showed that their Bayesian network model was successful in extracting more information from the data and estimated the gene network more accurately.

As with all models, even Bayesian networks have limitations. For instance, circular dependencies present in biological networks cannot be modelled. De Jong [2002] comments “Although Bayesian networks are intuitive representations of genetic networks, they have the disadvantage of leaving dynamical aspects of gene regulation implicit”. He also states that the problem can be overcome by using dynamic Bayesian networks to model dynamic processes, such as feedback mechanisms.

1.3 Aim and Structure of This Report

This project aims to develop and evaluate an algorithm that infers the structure of a gene network for a relatively small number of genes using Bayesian Networks and dynamic programming.

Chapter two provides a detailed description of Bayesian Networks, and goes on to discuss the task of inferring the network from biological data. Scoring functions and search algorithms are reviewed.

Chapter three aims to provide the reader with an understanding of the optimal search algorithm. We present the basic concepts and then go on to discuss the implementation, providing rationale behind any choices that had to be made.

Chapter four is a detailed analysis of the algorithm. Receiver operating characteristic (ROC) graphs have been employed to assess the performance of the technique. Results from both synthetic and real datasets are shown and discussed. Effects of increasing sample size and noise in the data are also investigated.

Chapter five concludes with a brief summary of the project, its achievements and limitations and includes ideas for further work.

2 Bayesian Networks

A Bayesian Network B for a set of random variables $X = \{X_1,...,X_p\}$ is the pair $\langle N, \Theta \rangle$ that uniquely specifies the joint probability distribution P_B for X. The network structure N is a directed acyclic graph (DAG), consisting of a set of nodes corresponding to random variables $\{X_1,...,X_p\}$ and a set of directed edges that represent dependencies between the variables. The parameter Θ is the set of conditional probability distributions that describe the conditional probability $P(X_i \mid Pa_i)$ of a variable X_i given its parents Pa_i in the graph.

A conditional independency $i(X_i;Y \mid Z)$ expresses the fact that X_i is independent of Y given Z, where Y and Z denote sets of variables. The graph N encodes conditional independence assumptions (Markov independencies), which state that each variable X_i is independent of its non-descendants, given its parents in N. Thus, for each variable, we have $i(X_i; nondescendants(X_i) \mid (Pa_i))$ [De Jong 2002]. The joint distribution P_B of a network B that satisfies the above independence statements can be decomposed into product form as

$$P_B = P(X_1,...,X_n) = \prod_{i=1}^{n} P(X_i \mid Pa_i), \qquad (1)$$

where Pa_i is the set of parents of X_i in N [Pearl 1988].

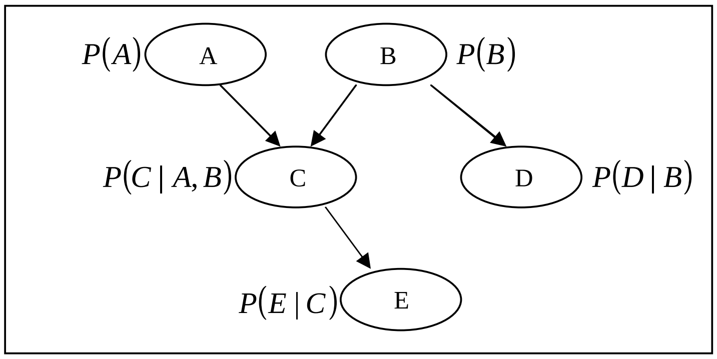


Figure 3: A direct acyclic graph: conditional probabilities for each variable are specified.

The above figure shows an example of a graph consisting of the set of variables $X = \{A,B,C,D,E\}$. The joint probability distribution, $P(A,B,C,D,E)$ can be calculated from the product of the conditional probability distributions for each variable, and is given by $P(A)\,P(B)\,P(C\,|\,A,B)\,P(D\,|\,B)\,P(E\,|\,C)$. Additional conditional independencies found in the graph are: $i(A;B,D)$, $i(B;A)$, $i(C;D\,|\,A,B)$, $i(D;A,C,E\,|\,B)$ and $i(E;A,B,D\,|\,C)$.

More than one direct acyclic graph (DAG) can imply the same set of independencies. For example, let us examine the three graph structures (a), (b), and (c) in Figure 4 below. The decomposed joint probability distributions respectively are:

$$\begin{aligned} p_a(A,B,C) &= p(A)p(C\,|\,A)p(B\,|\,C), \\ p_b(A,B,C) &= p(B)p(C\,|\,B)p(A\,|\,C), \\ p_c(A,B,C) &= p(C)p(A\,|\,C)p(B\,|\,C). \end{aligned}$$

The laws of conditional probability show that the three graphs represent the same probability distribution, and therefore denote the same set of independencies. The graphs are said to be equivalent to each other and belong to the same *equivalence class*.

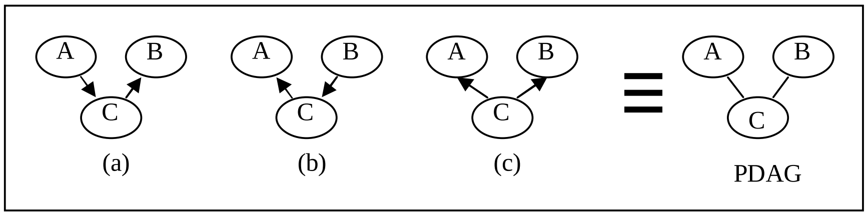


Figure 4: The three graphs (a), (b) and (c) belong to the same equivalence class, and can thus be uniquely represented by the PDAG structure.

Chickering [1995] shows that equivalent graphs have the same underlying undirected graph but can differ on the direction of some edges. Furthermore, his work demonstrated that an equivalence class of network structures can be uniquely represented by a partially directed graph (PDAG), where a directed edge $X \rightarrow Y$ compels all members of the equivalence class to contain the edge $X \rightarrow Y$, while an undirected edge $X - Y$ allows members to contain an edge in either one of directions $X \rightarrow Y$, and $Y \rightarrow X$.

The above notion of *equivalence* is important and especially relevant to our problem of learning structures from data. Graphs belonging to the same equivalence class cannot be distinguished from observing data alone. Additional criteria or knowledge is required [Buntine 1996]. Hence, in the absence of prior information, search strategy should be to find an equivalence class of networks that best matches the data rather than trying to find a single network.

2.1 Learning Bayesian Networks from Expression Data

The process of establishing relationships between genes on the basis of observed expression levels is referred to as 'reverse engineering' [Brazhnik et al., 2002]. This gene network inference task can be regarded as an unsupervised learning problem. We define our microarray expression dataset of *p* genes and *n* samples/experiments as a *training set* $D = \{x^1, x^2, ..., x^n\}$ of independent observations, where each data point x^i is an p-dimensional vector $x^i = \{x_1^i, x_2^i, ..., x_p^i\}$. We need to find a network $B = \langle N, \Theta \rangle$, or more precisely an equivalence class of networks that *best matches* this dataset D; both a biologically meaningful network structure and parameters (conditional probability distributions) need to be found.

2.1.1 Scoring Mechanism

Learning network structures from data can be termed as a *model selection* problem in the sense that each network corresponds to a distinct model and one is to be selected based on the data [Buntine 1996]. When searching for the best network over the space of possible networks, a criterion that measures the degree to which a network structure (equivalence class) fits the prior knowledge and data is required. A statistically motivated *scoring function* that assigns a score $S(B)$ to each network structure B is generally used to rank models based on their 'goodness of fit' to the data. Moreover, the difference between the scores for any two models leads to a direct significance measure for determining how strongly one should be preferred over the other Hartemink et al. [2001].

A number of scoring functions are used in research; all of them exhibiting two important characteristics – namely *decomposability* and *structural equivalence.*

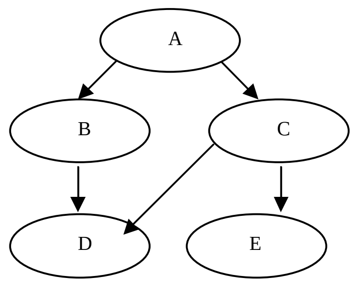
Decomposability

A scoring function can be decomposed in the presence of full data. When the dataset D is complete i.e. it contains neither missing or hidden values, the score for a network structure $S(B)$ can be expressed as a summation of terms that corresponds to individual nodes in the network. Just as the joint probability distribution of a Bayesian network is specified as the product of the conditional probability distributions of each of its variables (Equation 1), the scoring function factorises into terms related to the individual variable dependent only on its parents [Dejori 2002], allowing us to make efficient computations.

Given a network B consisting of a set of genes $G = \{g_1, ..., g_p\}$, the function $s : G \times 2^G \rightarrow \Re$ assigns a score to a gene $g \in G$ and a set of parent genes $A \subseteq G$. The score of the whole network is defined as

$$S(B) = \sum_{g \in G} s\left(g, Pa_g^B\right), \tag{2}$$

where Pa_g^B denotes the set of parent genes of gene g in the network B (Figure 5).



$$\begin{aligned} &S(B) \\ &= s(a, \{\ \}) + s(b, \{a\}) \\ &+ s(c, \{a\}) + s(d, \{b, c\}) \\ &+ s(e, \{c\}) \end{aligned}$$

Figure 5: Score of a network is calculated as the sum of scores for each variable in the network.

The contribution of each gene g to the total score thus depends only on its own value and the values of its parents in the dataset D.

Structural Equivalence

In the beginning of this chapter, *equivalence* among graphs, and the concept of an *equivalence class* of network structures were introduced. Structures belonging to the same equivalence class contain the same set of independencies, and thus have equal sample likelihoods. For this reason, two graphs that are structurally equivalent will be given the same score value by a scoring function. This is called *score equivalence*.

The following sections provide the details for three popular scoring mechanisms that have been used to infer gene networks from microarray datasets.

2.1.1.1 Bayesian Score (BDe)

One of the ways to learn a network from data D produced by microarray experiments would be to compute the posterior probability $P(B \mid D)$ i.e. the probability of the model B being correct given the observed data. The task would then be to find a model network $B^{\otimes}$ that maximises this probability, and parameters $\Theta^{\otimes}$ that maximise $P(\Theta \mid D, B^{\otimes})$. The Bayesian score is derived from Bayesian statistical methods and is proportional to the posterior probability.

According to Bayes rule, the posterior probability can be computed as:

$$P(B \mid D) = \frac{P(D \mid B)P(B)}{P(D)} \tag{3}$$

The term $P(B)$ is the prior probability of the network structure, while $P(D)$ is a normalisation constant that does not depend on the choice of model structure and can be ignored. The term $P(D \mid B)$ is the marginal likelihood for network structure B and represents the likelihood or probability of the data D given that the network structure is B and has parameters Θ [Friedman et al., 2000].

The Bayesian score is defined as the logarithm of the posterior probability $\log P(B \mid D)$ and can therefore be computed as $\log P(D \mid B) + P(B)$. To evaluate the marginal likelihood $P(D \mid B)$ we must consider all possible parameter assignments to B. Thus,

$$P(D \mid B) = \int P(D \mid B, \Theta) P(\Theta \mid B) d\Theta\,, \tag{4}$$

where $P(D \mid B, \Theta)$ is defined as the joint distribution of the variables in network (Equation 1), and $P(\Theta \mid B)$ is the prior density over parameter assignments to B [Friedman & Goldszmidt 1998]. The particular choice of priors $P(B)$ and $P(\Theta \mid B)$ determines the exact Bayesian score.

The work of Cooper & Herskovits [1992] introduced a set of BDe priors to be used in conjunction with the Bayesian score to evaluate the 'goodness' of a network. The BDe criterion evaluates a Bayesian network based on the multinomial distribution, and therefore needs data to be discretised. Additionally it makes assumptions of *complete data*, *parameter independence*, and *parameter modularity*. It requires the prior over parameters $P(\Theta \mid B)$ to have Dirichlet prior distributions and the structure prior $P(B)$ to be uniform. All of the above restraints cause the marginal likelihood term $P(D \mid B)$ to be rewritten. Heckerman [1998] explains the set of assumptions and shows that they justify the decomposition of the integral in Equation 4 making it analytically tractable.

The assumption of *parameter independency* states that the parameter values associated with a given variable is independent of the parameter values associated with another variable. This permits us to construct the priors for the parameters for each variable X_i separately. *Parameter modularity* implies that the distributions for parameters Θ depend only on the structure of the network that is local to variable X_i - namely X_i and its parents Pa_i. Finally, a *complete dataset* indicates that there are no hidden or missing values in the dataset. Friedman & Goldszmidt [1998] used the BDe criterion for model selection and demonstrated that it was suitable for inferring the true network.

2.1.1.2 Minimum Description Length (MDL)

The MDL scoring function is based on coding theory: given the data and a class of network models, select the model which achieves the shortest codelength for the data and the model. Codelength, also termed *description length* refers to the number of bits used in encoding.

In the context of learning Bayesian networks from expression data, we have networks B that describe a probability distribution P_B over the *n* samples of *p* genes appearing in dataset D. The MDL score for a network is defined as the *total description length*, which is the sum of the length of the encoded data, and the length of the description of the model network.

To describe the network model $B = \langle N, \Theta \rangle$, we need to encode both the graph N and its parameters Θ. The description of the DAG N depends on the number of parents each variable has in the network. If a variable X_i has k parents, then $\binom{p}{k}$ is the number of possible combinations of parents X_i could have from a set of p variables. The description length for the graph structure is therefore given by:

$$MDL_{graph}(N) = \sum_{i}^{p} \left(\log p + \log \binom{p}{\|Pa_i\|} \right), \tag{5}$$

where the first term encodes the number of parents $\|Pa_i\|$ while the second term encodes the index of the set of parents for the variable [Friedman & Goldszmidt 1998].

Next, to describe the network parameters Θ, we store the conditional probability distributions for each variable in tabular form. We assume that the dataset is discretised so that each variable X_i takes values from a finite domain. The encoding length of a variable's conditional probability table is given by:

$$MDL_{table}(X_i, Pa_i) = \frac{1}{2}(\|X_i\| - 1)(\|Pa_i\|)^k \log n, \tag{6}$$

where $(\|X_i\| - 1)(\|X_i\|)^k$ are the dimensions for each variable's table, with $\|X_i\|$ denoting the number of values taken by the variable. $1/2 \log n$ is the number of bits used to store each numeric parameter with n being the number of samples in the dataset. The above encoding term is also referred as the *penalty* term since it penalises the complexity of the network structure. For example, a simple network with fewer edges is preferred over a network containing more edges.

To encode the training data, the probability distribution P_B defined by the network B is used to build an encoding scheme that assigns shorter codelengths to instances that occur in the dataset with high probability. In effect, we encode the data D using network B by calculating the marginal likelihood $P(D \mid B)$. The representation length for encoding the marginal likelihood can be decomposed as a sum of terms that are local to each variable's conditional probability distribution. For a multinomial distribution, we get

$$MDL_{data}(D \mid B) = -\sum_{i=1}^{p} \sum_{j=1}^{r_i} \sum_{l=1}^{v_i} N_{ijl} \log \frac{N_{ijl}}{N_{ij}}. \quad (7)$$

The equation above works out the number of bits necessary to encode each value of X_i given that we know the value of the variable's parents. v_i is the set of values which variable X_i can take while r_i denotes the set of values that the parents of X_i can take on. The term N_{ijl} denotes the number of instances found in the dataset where variable X_i takes on value *l* and its parents take on value *j*.

Finally, the MDL score for network structure B is the total description length and is given by:

$$MDL(B) = MDL_{graph}(N) + \sum_{i}^{p} MDL_{table}(X_i, Pa_i) + MDL_{data}(D \mid B) \quad (8)$$

The optimal model is the one that minimises the score. The MDL approach seeks a class of network models that describe the data as accurately as possible, but also considers the complexity of the models as a penalising factor, striving to strike a balance between the two modeling aspects [Tabus & Astola 2001]. As the number of samples in the dataset increase, the criterion has been shown to converge to the BDe criterion described in section 2.1.1.1. Moreover, minimising MDL is equivalent to maximizing another scoring function known as the Bayesian information criterion (BIC) [Heckerman 1998].

2.1.1.3 BNRC

Both of the scoring mechanisms reviewed so far (Bayesian score with BDe priors and MDL) require data to be discretised and assume multinomial distributions. Friedman et al. [2000] reason that the number of discrete values allowed in the model, and the thresholds used in discretization of the continuous data are unknown parameters which have to be estimated from the data. Unsuitable parameters may therefore lead to wrong results. For this reason Imoto et al. [2002] propose a scoring mechanism called BNRC (Bayesian Network and Nonparametric Regression Criterion) that deals with continuous variables.

As with the BDe score in section 2.1.1.1, the BNRC criterion is also derived from Bayesian statistical methods. As before, the posterior probability of the network i.e. the probability of the model B being correct given the observed data can be written as:

$$P(B \mid D) = \frac{P(D \mid B)P(B)}{P(D)} = P(B)\int P(D \mid B, \Theta)P(\Theta \mid B)d\Theta \qquad (9)$$

Instead of using a multinomial distribution, the joint probability distribution $P(D \mid B, \Theta)$ of the variables (Equation 1) is captured using nonparametric regression models that are able to identify linear as well as non-linear dependencies between the variables. Laplace approximations are then used to compute the integration. The criterion is shown to be decomposable; the BNRC score for the network can be computed as the sum of the local scores of each variable in the network. Finally, the network that minimises the score is chosen to be the best model.

Imoto et al. [2002; 2003a; 2003b] apply their score on microarray expression data and obtain promising results. Furthermore they show that the score can incorporate a number of different types of biological knowledge into the prior probability $P(B)$ of the network. They conclude, "The balance between microarray information and biological knowledge is optimised by the proposed criterion". **Nariai et al. [2004]** also use the BNRC score criterion on expression data in combination with protein - protein interaction data, and attain accurate results that are comparable with earlier studies.

2.2 Finding the Optimal Model: Search Strategies

As defined earlier, learning structures from data involves finding a network or an equivalence class of networks that best fits the available data. To accomplish this, we utilise a score based search algorithm that identifies high scoring networks over the space of possible networks.

Finding the *optimal gene network* is a difficult task for two reasons. First, the number of samples contained in gene expression datasets is relatively small compared to the number of genes (Section 1.1.1). With a large number of samples, learning the ‘true’ network model is possible with high probability. The data that is currently available to us is not informative enough to determine a single optimal model. Searching will result in several different networks that equally fit the data reasonably well thereby introducing uncertainty into model selection [Buntine 1996]. From a Bayesian viewpoint, this means that the posterior probability over the search space is spread and not dominated by a single network model. Friedman et al. [2000] deal with this dimensionality problem by focusing on *features* such as pairwise relations that are common to high scoring networks instead of looking for a single network or a single equivalence class of networks. Pe’er et al. [2001] extend their work and consider additional features, like activation, inhibition and mediation relations between the variables.

Secondly, the number of possible Bayesian networks increases super exponentially with the number of variables in the network (Table 1). The problem is NP-hard, making computation intractable for networks containing a large number of variables. Ott et al. [2004] remark, “Even for a gene network of 9 genes (search space roughly $1.21x10^{15}$), a brute force approach would take years of computation time even on a supercomputer”.

Number of variables In network	Number of possible DAG structures
1	1
2	3
3	25
4	543
5	2.9×10^4
6	3.7×10^6
7	1.1×10^9
8	7.8×10^{11}
9	1.2×10^{15}
10	4.2×10^{18}
20	2.3×10^{72}

Table 1: The search space of possible Bayesian networks increases super exponentially with the number of variables in the network.

2.2.1 Heuristic Methods

To deal with the large number of gene variables found in microarray datasets, researchers have so far used local heuristic searches to learn Bayesian Networks from the data. Local searches work by making successive edge changes to the network, taking advantage of the decomposability property of scoring functions (Section 2.1.1) to evaluate the gain made by each change. Changes made at each step could be adding, removing or reversing the direction of a single edge. All changes ensure that the resulting network structure does not contain directed cycles. Popular heuristic search algorithms are greedy-hill climbing, greedy random search, simulated annealing, and Monte-Carlo methods.

The simplest heuristic method is the greedy hill-climbing search, and has been adopted by Pe'er et al. [2001] for learning gene networks. The method starts with an initial network structure B, which could be an empty graph, a random graph, or a domain specific prior network. Local search proceeds through the set of eligible changes B' that can be made to the graph, and keeps the change B'' for which the gain in score is largest. The algorithm is terminated when the structure cannot be improved further i.e. there is no structure with a better score (Figure 6).

```
Choose initial network B
FOR each change $B_i$ in B'
     Compute $S(B_i)$
END

$B'' = \text{argmax}_{B_i} S(B_i)$
IF $S(B'') > S(B)$ THEN
     B:= B''
ELSE return B
```

Figure 6: Pseudo-code for greedy hill-climbing search algorithm.

Although greedy hill-climbing search has been commonly used for learning network purposes, it encounters several problems. Firstly, local search might find a set of edge changes that have the same high score. Which edge change does the method pick to improve the structure? Secondly, the method can get stuck in local optima, meaning that there might be another network structure with a higher score that hasn't been discovered by the search algorithm.

To avoid the above problems, Imoto et al. [2003a] use greedy random search for inferring gene networks. They apply the greedy hill-climbing method until it hits a local maximum. Then, they randomly perturb the network structure by permuting the computational order of the genes and repeat the greedy search for a number of iterations. Hartemink et al. [2002] try to prevent local maxima by using a search algorithm called simulated annealing.

Simulated annealing starts with an initial network structure B and picks an edge change e from the set of eligible graph changes B' at random. The change e is accepted as an improvement to network B if it obtains a higher score. If it obtains a lower score, it is accepted with probability $p = \exp(S(e) - S(B)/T_0)$ where T_0 is a *temperature* parameter. We repeat the process, and lower the value of the temperature parameter gradually after a select number of iterations (Figure 7). Initially, when the temperature is high, a lot of edge changes are accepted. The method explores a lot of search space, and hence has a higher chance of finding the global optimal network. As the temperature decreases, few edge changes are accepted and a stable network is obtained.

```
Initialise $T_0$
Choose initial network B
WHILE $T_0$ < T
FOR x = 1 to N
  Pick random change $B_i$ from B’
  Compute S($B_i$)
  Change = S($B_i$)-S(B)
  IF Change>0 OR exp(Change/$T_0$)>random
  THEN
     B:= $B_i$
END FOR
Reduce $T_0$
END WHILE
```

Figure 7: Pseudo-code for the simulated annealing method.

The results from heuristic methods vary depending on the network used at initialisation and the order in which computations are made. Despite the improvements made by simulated annealing or random greedy hill-climbing searches, we cannot be confident about the accuracy of the approach.

2.2.2 Ott's Approach

Avoiding heuristic methods, Ott et al. [2004] propose an algorithm that uses *dynamic programming* to search through the super exponential space of possible networks, obtaining the optimal model in exponential time. The authors focus on real-valued continuous gene expression data and utilise the BNRC scoring criterion (Section 2.1.1.3) for learning the network structure. They apply their method on microarray datasets that studied the response of yeast to various stress conditions, and obtain biologically plausible results.

The method has a sound theoretical basis, and can be made to work with different scoring functions, and any kind of gene expression measurements. The method is feasible for studying only small gene networks of up to 30 genes, although techniques such as limiting the number of parents for each gene can increase the scalability of the approach. Biological information such as protein-protein interaction data, binding site information can also be incorporated into the method as prior information.

In this thesis, we implement the above method but concentrate on using discretised data, searching for networks using the MDL score (Section 2.1.1.1). We evaluate the algorithm's performance on both synthetic and real microarray datasets, and investigate the effects of increasing sample size, and noise in the dataset on the method's accuracy. The next chapter gives a description of how the algorithm works and discusses implementation.

3 Optimal Search Model

Inferring a gene network using a Bayesian framework involves finding a set of parent genes for each gene in the network such that the resulting network is acyclic and that the score of the network is minimal. The naïve approach for finding the best model is to search through the whole search space of possible networks, and return the network or equivalence class of networks that gives the minimal score. As explained in Section 2.2, this is computationally infeasible. Additionally, a lot of time would be wasted on investigating networks that relate very poorly with the given data. The optimal search algorithm by Ott et al. [2004] finds the optimal gene network model for a set of genes using a dynamic programming strategy.

Dynamic programming is applied to our gene network inference problem since it helps us to effectively prune the huge search space. Information gained from searching aids in making decisions about which subspace needs to be searched next, that is we are able to avoid or eliminate poor network models from being investigated. The idea is to find the subspace within the super-exponential search space that contains the optimal network. Once we know the subspace, we can exhaustively search and determine the network.

3.1 Formal Definition

The method uses the concept of π-linearity and ordering of the variables in the graph to define network subspaces. In a network B consisting of a set of genes $G = \{g_1, ..., g_p\}$ we can define an ordering of a set $A \subseteq G$ as a permutation $\pi : \{1, ..., |A|\} \rightarrow A$. We would like a network to have an ordering such that all edges in its acyclic graph are oriented in the direction of the ordering. In other words, the each gene g in the network should be positioned earlier on in the ordering than its parents. Such an ordering would result in a network that is π-linear.

Definition 1: π - *linearity*

Let $A \subseteq G$ and $\pi \in \prod^A$ where $\prod^A$ is the set of all permutations of A. Let $B \subseteq A \times A$ be a network. B is π-linear iff for all $(g, h) \in B$ $\pi^{-1}(g) < \pi^{-1}(h)$ holds.

Given a fixed ordering or permutation π of a network, we can define a subspace as the set of all networks that comply with the given ordering i.e. the set of all networks that are π-linear. Two functions, namely *F* and Q^A have been defined that search through the space of π-linear networks given a permutation π and calculate the score of each network.

Definition 2: *F*

$F : G \times 2^G \to \Re$, where $F(g, A) =_{def} \min_{B \subseteq A} s(g, B)$ for all $g \in G$ and $A \subseteq G$.

The function *F* returns the optimal choice of parents for a gene *g* when parents have to be selected for it from the subset *A*. It works by scoring each set of possible parents in *A* with gene *g* using some kind of scoring criterion, be it MDL, BNRC, BDe or indeed any score mechanism that can be decomposed (Section 2.1.1). The optimal parent set is the one that returns the minimal score.

Definition 3: *Q^A*

Let $A \subseteq G$. For all $\pi \in \prod^A$, $Q^A : \prod^A \to \Re$ can be defined as:

$$Q^A(\pi) =_{def} \sum_{g \in A} F\left(g, \{h \in A \mid \pi^{-1}(h) < \pi^{-1}(g)\}\right)$$

Given a permutation π on a set of genes *A*, function Q^A calculates the score of each π-linear network by summing the score of each gene *g*, and its optimal parents using the information obtained from the *F* function. The best π-linear network is the one that returns the minimal score. The authors Ott et al. [2004] argue that if they are able to find the best π-linear network for a given permutation π, then in order to find the optimal network, all that needs to be done is to find the optimal permutation π. This is defined by function *M*.

Definition 4: *M*

For all $A \subseteq G$, we define $M : 2^G \to \bigcup_{A \subseteq G} \prod^A$ as:

$$M(A) =_{def} \arg\min_{\pi \in \prod^A} Q^A(\pi)$$

The above function returns the optimal permutation for a set of genes *A* by selecting the permutation π that returns the minimal *Q* value.

3.2 The Algorithm

The goal of the optimal search model is to find $Q^G(M(G))$ i.e. we want to know the optimal permutation for all genes G in the network, and build the optimal network according to this permutation. Since both *Q* and *M* functions require values of *F*, we compute $F(g, A)$ for all *g* and all *A*, and also $M(A)$ for all *A*. In other words, for each gene *g* and each set *A*, we compute the optimal selection of parents for *g* from *A*, and for each set *A*, we compute the optimal permutation for π- linear networks on *A*.

Dynamic programming allows us to divide the task into stages and calculate the F, Q and M functions for a set of genes recursively.

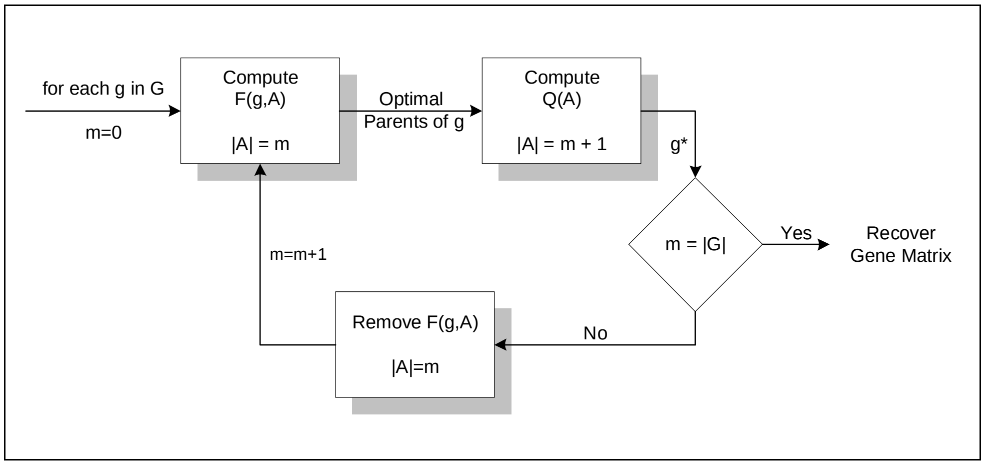


Figure 8: The Optimal Search Algorithm

The above figure shows how the algorithm functions. For each gene *g* in the set of genes *G*, we start by computing the F score, with *A* being the empty set ($|A| = 0$), i.e. we calculate the score for each gene assuming that it has no parents. This allows us to compute the scores for all possible π- linear networks containing only one gene, i.e. we compute $Q(A)$ for $|A| = 1$. The remaining F and Q scores are calculated recursively by increasing the cardinality of set *A*. At each stage, the *Q* score for a subset $A \subseteq G$ of cardinality $|A| = m$ will require the $F(g, A)$ score values of each gene *g* and all subsets *A* of cardinality $|A| = m - 1$. We stop recursion once *A* reaches the set of all genes *G*, i.e. when $|A| = |G|$.

The remainder of this section describes how we implement the algorithm to obtain an optimal gene network model. The program was written in C++, and the full code listing can be seen in Appendix B.

3.2.1 Input

The program receives as input a set of discrete valued data that has been generated from a network of a number of genes. For any given data, we need to specify three parameters, the number of genes in the network (*nvar*), the number of samples present in the dataset (*ns*), and the number of values that each variable in the network can have (*cs*). For binary data, this value would be 2, whereas for ternary data containing for instance the gene expression values -1, 0 and +1, the value would be 3.

Our program uses the Standard Template Library in C++, and uses sets as the basic data structures. We use the function *generate ( )* that takes as parameters an integer *k* and the number of variables *nvar*, and creates subsets of cardinality *k* from a set of *nvar* integers.

3.2.2 Computing F-scores

The function $F(g, A)$ that takes as input a gene *g* and a set of parents *A* of cardinality *m* is computed as follows:

$$F(g, A) = \min\{s(g, A), \min_{a \in A} F(g, A - \{a\})\} \tag{10}$$

It basically computes the score of gene *g*, and parents *A* and compares it to previously stored score values of gene *g* with the number of parents being one less than those contained in set *A*. The optimal parent set for gene *g* is the set with the minimal score. As mentioned before, the formula is recursive. *F* scores for gene g and set *A* of cardinality *m* need to lookup F scores for gene g and the set of parents *A* of cardinality *m* – 1. To store the score values and the set of optimal parents, a class named *Fset* was created. To speed lookups of the score values and optimal parents, each *Fset* was directly indexed by the pair (g, A) for which it was calculated.

The score criterion $s(g, A)$ for computing the *F* scores was MDL. We defined an *mdl()* function that took a gene *g*, the set *A*, and the given dataset as input and calculated the score using the equations defined in Section 2.1.1.2. As explained, the total description length consists of summing up the description length of the graph, the conditional probability table of the variable *g* given its parents in set *A*, and the length of encoding the data. Calculating MDL_{graph} (Equation 5) and MDL_{table} (Equation 6) is fairly easy, since we know the number of variables (*nvar*), the number of values the data can take (*cs*), and the number of parents contained in the set *A*. To calculate MDL_{data} is a little more complex. We record the frequencies of all possible value combinations of gene *g* with parents in set *A* occurring in the dataset, and use them for calculating the likelihood of *g* given its parents in set *A* (Equation 7). Taking logs of this likelihood gives us the description length for the data.

3.2.3 Computing Q-scores

The function $Q(A)$ takes as input a set *A* and returns g*, an element that is considered to be the last element in the permutation of all elements of set *A*.

$$g^* = \arg\min_{g\in A}\left(F\left(g, A-\{g\}\right)+Q^{A-\{g\}}\left(M\left(A-\{g\}\right)\right)\right) \tag{11}$$

Therefore, for a set *A* of cardinality *m*, the Q function involves looking up both *F* scores and *Q* scores for subsets of cardinality *m-1*. Another class named *Qset* was created to store the *Q* score values and g* element for a given set *A*. We also stored the optimal parents of the g* element by looking up $F\left(g^*, A-\{g^*\}\right)$.

3.2.4 Output: Gene Matrix

By incrementing the cardinality of set *A*, we eventually compute the *F* and *Q* scores of all $g \in G$ and all possible subsets $A \subseteq G$ in the network. We now have all the information we need to build the optimal gene network for the set of given genes. We can figure out the optimal permutation or ordering of the variables in the network by looking at the g* element stored in $Q(G)$. To find the next last element in the ordering, we decrease the cardinality of set G and lookup the g* element in $Q\left(G-\{g^*\}\right)$. We do this repeatedly until we reach the empty set. To find the optimal parents for each gene g in the ordering, we simply lookup the optimal parents associated with each g* element stored in the *Qset*. The optimal permutation and the optimal set of parents for each gene are used to compute an *nvar* x *nvar* gene matrix that specifies child – parent relations between the gene variables.

3.3 Small Example

For a clear understanding of how the algorithm works, we present a small example. Let's say for instance that we have a network of three genes, a, b, and c. To find the optimal model, we perform the following computations:

1. Compute $F(g, A)$ for all subsets A of cardinality 0:

$$F(a,\{\ \}) = s(a,\{\ \})$$
$$F(b,\{\ \}) = s(b,\{\ \})$$
$$F(c,\{\ \}) = s(c,\{\ \})$$

2. Generate all possible subsets A of cardinality 1: $\{a\},\{b\},\{c\}$
3. Compute $Q(A)$ for all subsets A of cardinality 1:

$$Q(a) = F(a,\{\ \}) \rightarrow g^* = a$$
$$Q(b) = F(b,\{\ \}) \rightarrow g^* = b$$
$$Q(c) = F(c,\{\ \}) \rightarrow g^* = c$$

4. Compute $F(g, A)$ for all subsets A of cardinality 1:

$$F(a,\{b\}) = \min\{s(a,\{b\}), \min F(a,\{\ \})\}$$
$$F(a,\{c\}) = \min\{s(a,\{c\}), \min F(a,\{\ \})\}$$
$$F(b,\{a\}) = \min\{s(b,\{a\}), \min F(b,\{\ \})\}$$
$$F(b,\{c\}) = \min\{s(b,\{c\}), \min F(b,\{\ \})\}$$
$$F(c,\{a\}) = \min\{s(c,\{a\}), \min F(c,\{\ \})\}$$
$$F(c,\{b\}) = \min\{s(c,\{b\}), \min F(c,\{\ \})\}$$

5. Generate all possible subsets A of cardinality 2: $\{ab\},\{ac\},\{bc\}$
6. Compute $Q(A)$ for all subsets A of cardinality 2:

$$Q(ab) = \min\{F(a,\{b\}) + Q(b), F(b,\{a\}) + Q(a)\} \rightarrow g^* = a \mid b$$
$$Q(ac) = \min\{F(a,\{c\}) + Q(c), F(c,\{a\}) + Q(a)\} \rightarrow g^* = a \mid c$$
$$Q(bc) = \min\{F(b,\{c\}) + Q(c), F(c,\{b\}) + Q(b)\} \rightarrow g^* = b \mid c$$

7. Compute $F(g, A)$ for all subsets A of cardinality 2:

   $$F(a,\{bc\}) = \min\{s(a,\{bc\}), \min\{F(a,\{b\}), F(a,\{c\})\}\}$$
   $$F(b,\{ac\}) = \min\{s(b,\{ac\}), \min\{F(b,\{a\}), F(b,\{c\})\}\}$$
   $$F(c,\{ab\}) = \min\{s(c,\{ab\}), \min\{F(c,\{a\}), F(c,\{b\})\}\}$$

8. Generate all possible subsets *A* of cardinality 3: $\{abc\}$
9. Compute $Q(A)$ for all subsets A of cardinality 3:

$$Q(ab) = \min\{F(a,\{bc\}) + Q(bc), F(b,\{ac\}) + Q(ac), F(c,\{ab\}) + Q(ab)\} \to g^* = a \mid b \mid c$$

By increasing cardinality of the subset *A*, we have been able to recursively calculate both the functions *F* and *Q*. No more computations needed to be carried out once the cardinality of set *A* reached 3, the number of genes in the network. It should also be noted that for computational reasons, we are able to remove the cached *F* score values once they have been used up in calculating the *Q* scores.

In order to build the gene network from the above calculated scores, we do the following:

1. Find $Q(abc)$ and lookup the corresponding g* and optimal parents value. If $F(a,\{bc\}) + Q(bc)$ is minimal, then $g^* = a$, and its optimal parents are from the set $\{bc\}$.
2. Find $Q(bc)$ and lookup the corresponding g* and optimal parents value. If $F(c,\{b\}) + Q(b)$ is minimal, then $g^* = c$, and its optimal parents are from the set $\{b\}$.
3. Finally lookup $Q(b)$. The corresponding $g^* = b$ and optimal parents are from the empty set $\{\ \}$.

The algorithm displays the optimal permutation as being $\langle b, c, a \rangle$ where *a* has parents from the set $\{bc\}$ and *c* has parents from the set $\{b\}$.

4 Results and Discussion

Most studies on inferring genetic networks assess the accuracy of their results on real gene expression data by comparing predicted regulatory interactions with those known from biological literature. Although we are able to estimate the number of true interactions, there is yet no reliable way of quantifying the number of false edges detected in the network. As Husmeier [2003] says, "It is impossible to decide without doing more experiments whether an algorithm has discovered a new, previously unknown interaction or whether it has flagged a spurious edge". For this reason, it has become increasingly necessary to test the viability of any genetic inference method on synthetic data as well as real data.

The aim of the results and discussion presented in this chapter is two-fold. First, we assess the effect of changing sample size and noise in the dataset using artificially constructed networks. Can the method cope with noisy data and low number of samples, features that are characteristic of microarray data? Secondly, we apply the algorithm to real data, selected from the study by Kim et al. [2000] and compare the results with those presented in the paper.

4.1 Synthetic Data

Two synthetic datasets have been used to see how much information from a known network can be recovered under the varying conditions (number of samples, noise). The first was a benchmark Bayesian network known as the 'Asia Network' consisting of 8 variables while the other was a constructed network of 11 variables.

4.1.1 Asia Network

This is a very small Bayesian network proposed by Lauritzen et al. [1988] to help diagnose patients arriving at a chest clinic (Figure 9). Each variable in the network corresponds to some condition of the patient. The network consists of 8 discrete variables with binary values (true, false) connected by 8 edges. The links between the nodes indicate how the relationships between the nodes are structured. The two top nodes, *A* and *S* are for predispositions which influence the likelihood of the diseases.

They link to nodes *T*, *L*, *E*, and *B* that represent internal conditions or failure states. They in turn link to the nodes for observables i.e. the symptoms *X* and *D*.

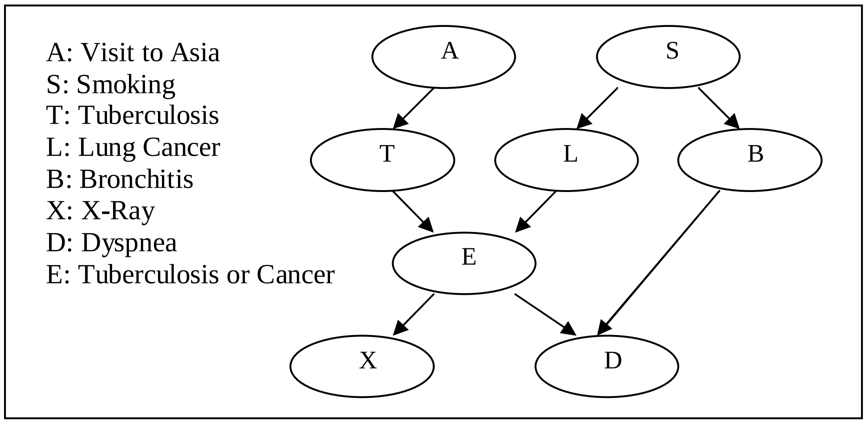


Figure 9: The Asia Network

The set of conditional probability distributions for the network are described in Netica [Norsys], the software that was used to generate the datasets. We chose to experiment with nine sample sized datasets, ranging from 10 to 1000 in the number of samples. For each sample size, the algorithm was applied to 5 different datasets, running a total of 45 experiments on this network.

4.1.2 11-Node Network

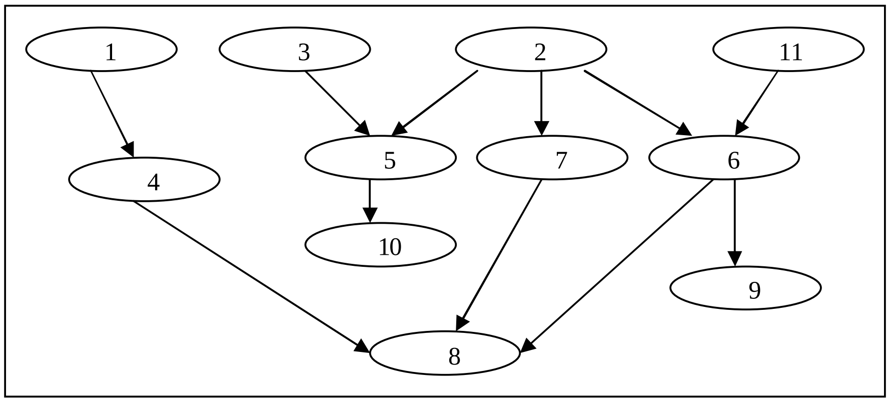


Figure 10: 11-node Network

The second set of experiments was performed on the above artificial network (Figure 10). Unlike the Asia network, this network is not generated from a set of probability distributions. To generate data, we specified a set of functions where the value of each variable depends on the values of its parents in the network. Variable nodes 1, 2, 3 and 11 are independent, while the rest of the variables have at most three parents. The functions describe linear as well as non-linear relationships among the variables. An ‘R’ script [Appendix C] provided by Sascha Ott used the set of functions to generate continuous valued data for the network. Since we were using the MDL criterion to score networks, we had to modify the script to output discrete data values. This was done by normalisation – variable values within one standard deviation of the mean were set to value 0. Values above and below this range were set to +1, and -1 respectively.

Two sets of experiments were performed on this network. First, we ran the algorithm on 10 different sized datasets containing between 10 and 1000 samples. Secondly, to try and simulate real microarray data, we introduced noise into the network. In order to solely study the affect of noise, the experiments were run on large datasets each containing 1000 samples. Seven different noise levels in the range of 0.01 and 0.5 were introduced into the data. We did not choose to experiment with levels greater than 0.5 since that would mean that the values were being generated randomly. To ensure validity, each experiment was repeated thrice, each time on a different dataset.

4.1.3 Performance Measures

Synthetic data allows us to quantify accuracy by looking at the similarities and differences between the inferred network and the known network. There are four possible outcomes when comparing the true network with the predicted model. If an edge exists in the known network and is recovered by the model, it is counted as a *true positive* **(TP)**; if it is not recovered it is considered as a *false negative* **(FN)**. When both the network and inferred model agree on the absence of an edge, we have a *true negative* **(TN)**, while if the model infers an edge that does not exist in the known network, we count the edge as *false positive* **(FP)**. Figure 11 encapsulates the above observations as a *confusion matrix*.

Predicted Model

		Edge	No Edge
Actual	Edge	True Positive	False Negative
Network	No Edge	False Positive	True Negative

Figure 11: Confusion matrix for gene network inference

We used *receiver operator characteristics* (ROC) graphs, the evaluation method suggested by Husmeier [2003] to assess the efficiency of our gene network inference method. It is as important to determine the number of true edges predicted as it is to quantify the number of false edges predicted by the model. ROC graphs enable us to judge the performance of a method on both these measures. The accuracy or *sensitivity* of a model is defined as the proportion of recovered true edges, also known as *true positive rate.*

$$\text{True Positive Rate} = \frac{TP}{(TP + FN)}$$

Similarly, the proportion of recovered spurious edges is known as *false positive rate* or *complementary specificity* of a model.

$$\text{False Positive Rate} = \frac{FP}{(TN + FP)}$$

To assess our network inference algorithm, we analysed its output i.e. the matrix denoting the predicted network. For each experiment, we plotted the model's true positive rate against its false positive rate to produce a single point located in two-dimensional ROC space (Figure 12).

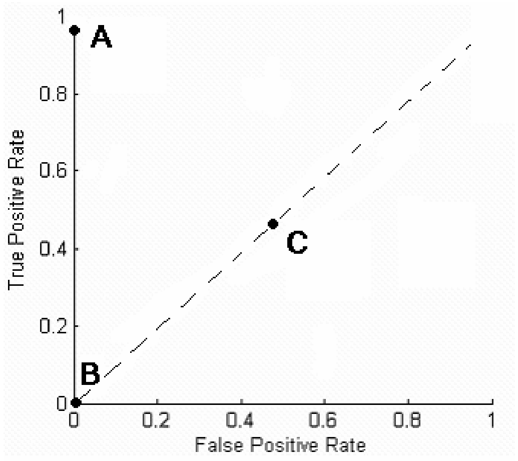


Figure 12: An ROC graph showing three different network predictors

In the ROC graph above, Point A at location (1, 0) produces the optimal network. Point B at location (0, 0) represents a method that is unable to infer any edges, while point C at (0.5, 0.5) represents a method that infers true edges randomly. In general, a model that produces a point located in the north-west region of the graph is said to be a good network predictor. The idea is to maximise the number of true edge predictions (high TP rate), while minimising the number of spurious edge predictions (low FP rate) at the same time.

In Chapter 2, we had mentioned that networks belonging to the same equivalence class cannot be distinguished by data alone. This means that our method could possibly infer a network structure that is not the true known network, but belongs to its equivalence class. For this reason, we decided to construct the PDAG (partial DAG) matrix for both the Asia and 11-node network. Conversion of a DAG to a PDAG was relatively easy and is explained by Chickering [1995] in his study on Equivalent Bayesian Network structures.

For each experiment performed on both sets of synthetic data, the predicted model's DAG, or network matrix was compared with the DAG and PDAG matrices of the actual network. For the 11-node network, we compared the predicted graph with an additional matrix that considered justifiable inferences as true edges. For instance, if we have a $A \rightarrow B \rightarrow C$ relationship in the network, a justifiable inference would be that A is the parent of C, i.e. we considered indirect relationships as being true. This new matrix was termed JDAG (justifiable DAG).

4.1.4 Results

This section presents the results of testing the performance of our method on synthetic data. The effect of sample size was studied using both networks, while the noise parameter was varied on the 11-node network. Complete set of results can be consulted from Appendix A.

4.1.4.1 Effect of Sample Size

Figure 13 shows the ROC graph of the algorithm's performance on the Asia network dataset. The graph shows the *average* true positive and false positive rates since we repeated the experiment on 5 different datasets for each sample size. The gene matrix of the predicted model was compared to the directed acyclic DAG matrix of the known network. As explained in the previous section, we had also compared the predicted graph with the partial DAG structure of the known network. The true positive and false positive rates for both the DAG and PDAG structures were almost identical. Hence, we only present the results of the DAG structure here. As can be seen from the graph below, there is a clear correlation between the number of samples in the dataset and the accuracy of the algorithm. As the number of samples increase, the models being inferred move closer towards the point location (1, 0) which denotes the optimal model. We observe a steady rise in the true positive rate as the size of the dataset increases, i.e. the proportion of true edges being recovered.

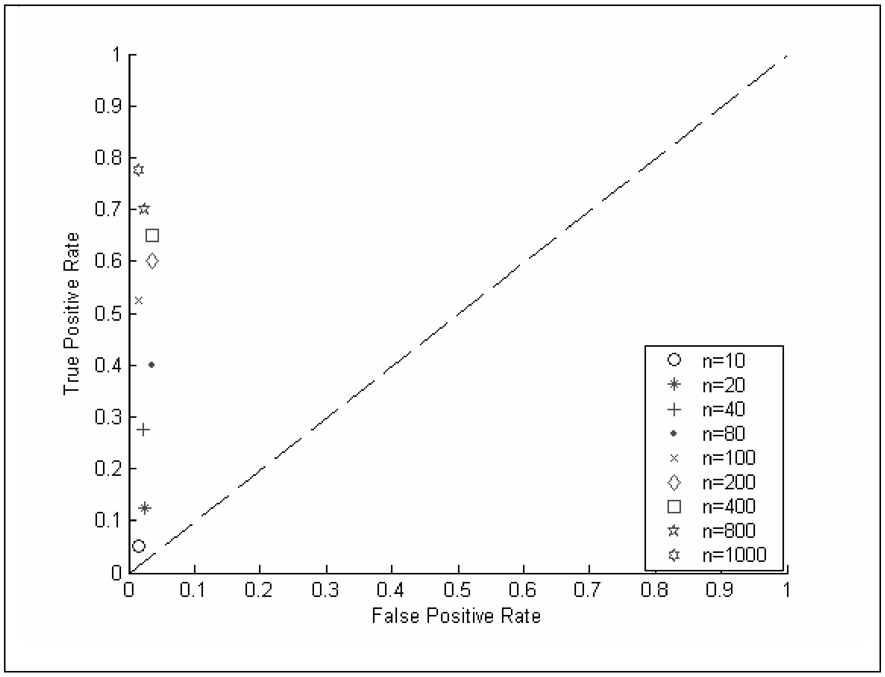


Figure 13: ROC graph for the Asia network, results being averaged over 5 runs. The graph shows the effect of varying sample size in the dataset. Predicted model was compared with the directed acyclic graph (DAG) of the known model.

In the above graph, we hardly notice a difference in the false positive rates. This could be explained by the fact that the underlying model of the Asia network is itself a Bayesian network. The optimal search algorithm is therefore highly unlikely to infer spurious edges since it also is designed under the same framework. For interest, we plotted the TP and FP rates for the five different datasets on which each sample size was tested. The results are displayed in Figure 14, with the ROC graph zoomed in. We can still observe the general pattern of larger sample sizes giving more accuracy.

For instance, the graph clearly demonstrates that a dataset of 10 samples is not at all sufficient for recovering the true network. At the same time, we notice a fair amount of variance in the performance of the large datasets. The true positive rate of a 400 sample large dataset for instance ranges between 0.5 – 0.9.

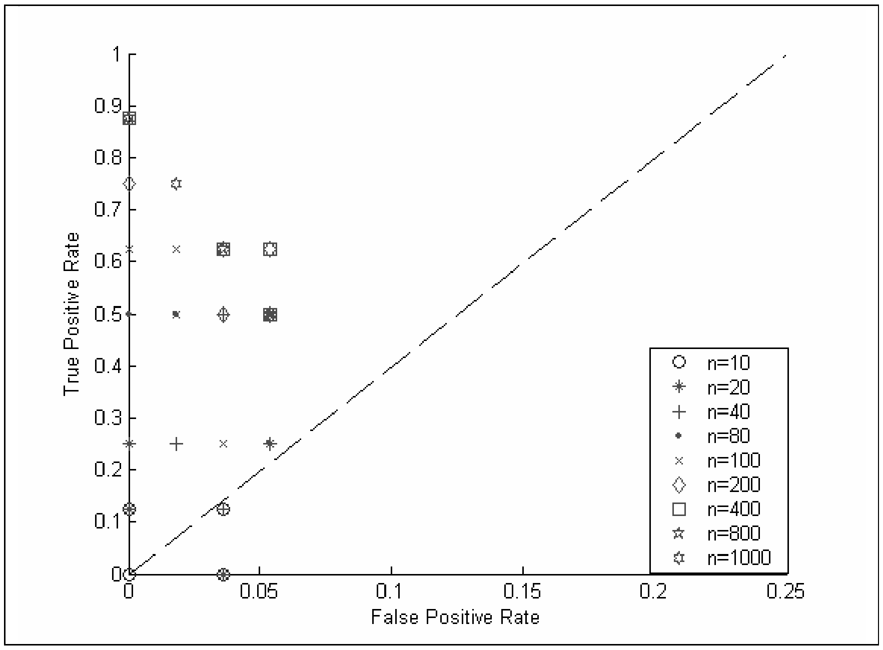


Figure 14: ROC graph for the Asia network. For each sample size tested, the algorithm was applied to five different datasets. The graph shows the variance in performance over the 5 datasets.

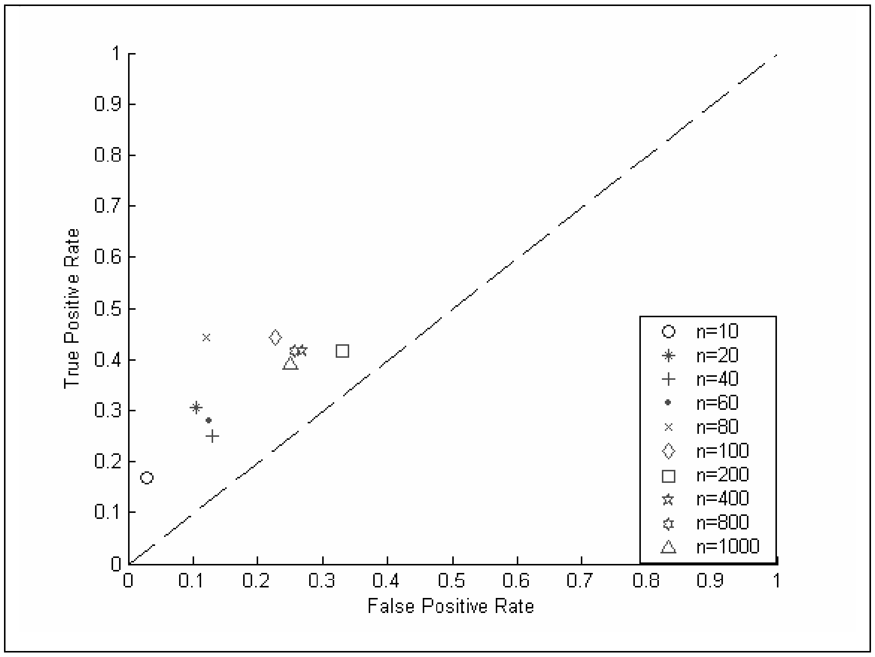


Figure 15: ROC graph for the 11-node network, results being averaged over 3 runs. The graph shows the effect of varying sample size in the dataset. Predicted model was compared with the PDAG of the known model.

Figure 15 and 16 show the effect of sample size on the performance of the 11-node network. The former is the ROC graph when comparing the predicted model to the PDAG structure of the known network, while the latter compares it to the justifiable directed acyclic graph (JDAG).

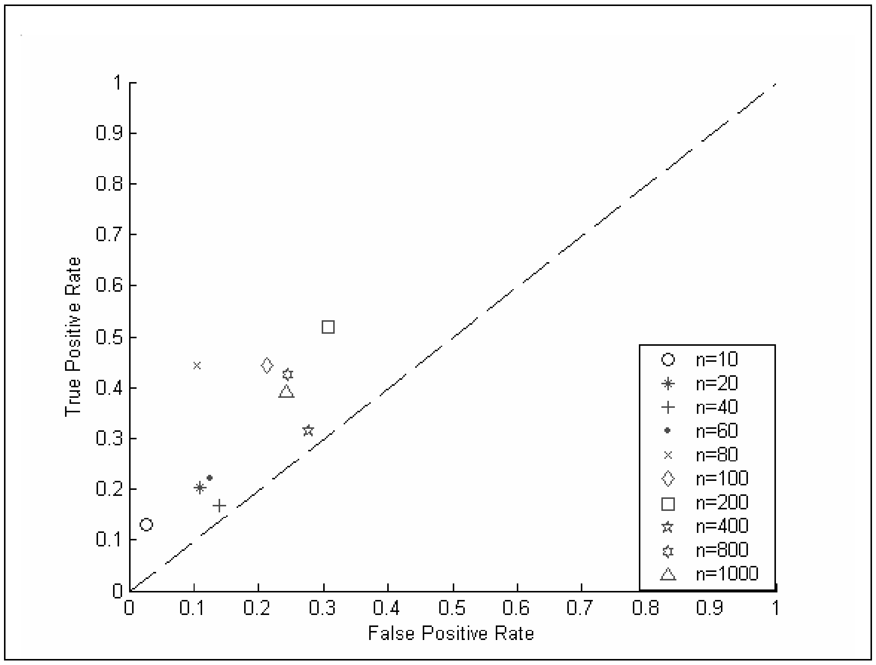


Figure 16: ROC graph for the 11-node network, results being averaged over 3 runs. The graph shows the effect of varying sample size in the dataset. Predicted model was compared with JDAG of the known model.

The performance of the method on the second synthetic dataset is not as good as the performance on the Asia network. In general, most of the datasets have both the true positive rates and false positive rates below 0.5. Datasets of size 10, 20, 40 and 60 are clustered together in the south western region of the graph and are poor performers, which is intuitive. Increasing the dataset size does improve performance in general but there is no clearly defined pattern. Furthermore, the optimal search method seems to be inferring a large amount of edges that are not present in the actual network. In fact, larger sized datasets induce a higher number of spurious edges into the network, causing the false positive rate to be high.

4.1.4.2 Effect of Noise in Data

The 11-node network was used to assess the effect of noise present in datasets containing 1000 samples. Figure 17 shows the average results plotted in ROC space for the datasets with varying levels of noise incorporated. The first thing we notice is that adding noise really only affects the proportion of true edges being recovered. The proportion of false edges being inferred is almost constant over all the datasets.

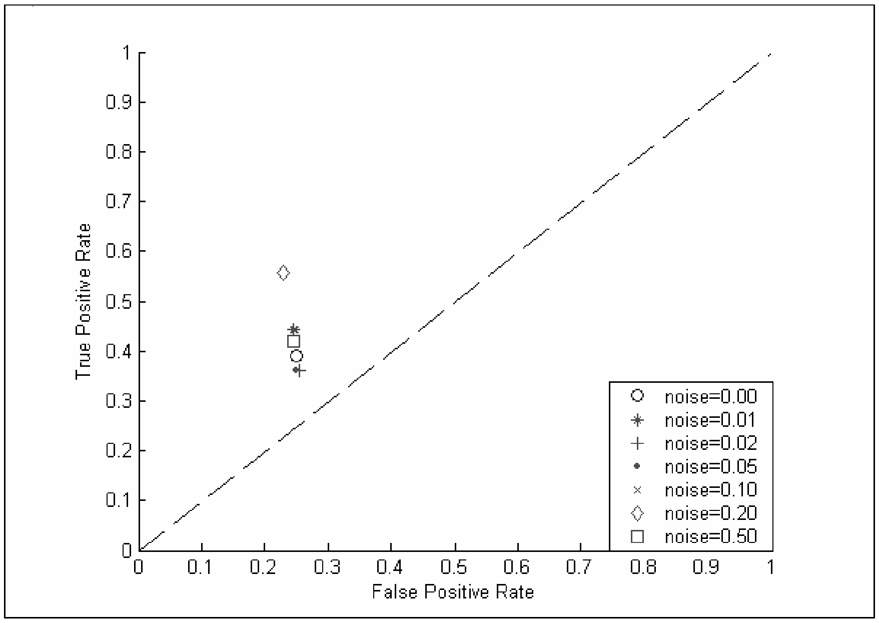


Figure 17: ROC graph for the 11-node network, results being averaged over 3 runs. The graph shows the effect of varying noise levels in the dataset.

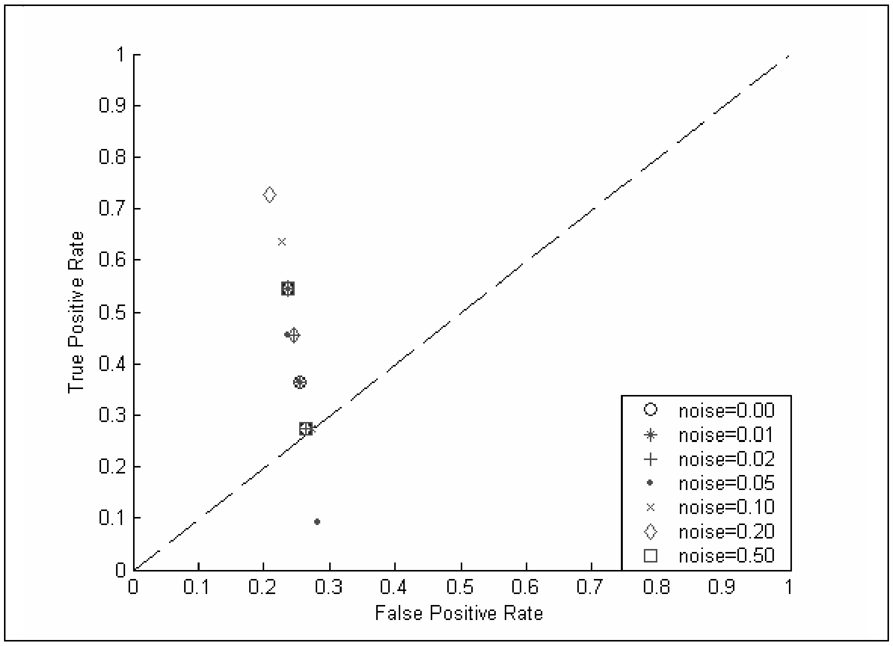


Figure 18: ROC graph for the 11-node network. Depicts the variance in performance of the algorithm on datasets containing varying noise parameters.

Secondly, we observe that the performance of the method actually *increases* when we amplify the noise in the dataset. In fact, the best results are provided by the dataset where the noise level is set to 0.2. This is somewhat puzzling. We plot the true positive and false positive rates of each of the individual datasets tested (See Figure 18) and observe the variance in performance. We now observe that noisy datasets do not always give better performance than datasets containing a lower level of noise. Most if not all of the datasets show a large variance in performance, and it might just be necessary to average the performance of the method over a larger number of datasets than used in this study.

4.2 Microarray Data

After several experiments on artificially constructed data, we ran the algorithm on a microarray dataset. Since our focus was on discretised data, we chose a dataset that had already been discretised by Kim et al. [2000] in their study on gene networks. This option was easier than obtaining continuous microarray data, selecting a subset of genes using clustering and then discretising the values ourselves using some arbitrary threshold.

Kim et al. [2000] used real microarray data from known gene response pathways of ionizing radiation and downstream targets of inactivating gene mutations and converted it into ternary expression data by thresholding the changes at the transcript level. They used a perceptron network to predict relations among the genes, providing a measure of confidence with their prediction. Several relations had been found that were known from previous biological knowledge. Our program was provided with the discretised dataset of 12 genes, and 30 samples. The resulting gene matrix obtained was highly sparse and inferred a total of only nine relationships, some of which were mentioned in the study.

4.3 Discussion

The results presented in the previous sections indicate that the optimal search algorithm implemented in this project is indeed capable of inferring a network from a given dataset. The algorithm was successful in reconstructing 7 out of 8 edges in the Asia network from a dataset of 1000 samples, and reasonably successful at reconstructing the network of the 11-node network. When applied to real microarray data, the method constructed a highly sparse gene matrix, but inferred some biologically meaningful gene relations.

Effect of Sample Size

Increasing the sample size of data generated from the Asia network showed very good results. The method clearly demonstrated that large datasets increase the proportion of true edges being inferred from the data. The results from experiments run on the 11-node network were not as profound but still indicated that a large sample size would result in higher accuracy. The findings are not surprising. In small datasets, the number of edges being inferred in total is low since the dependencies encoded in the data are statistically too insignificant to be picked up. In a large dataset however, we have enough data to estimate the likelihood of a particular variable having another variable as a parent in the network.

Several important observations were noted from the above experiments on sample size. First of all, the method performed a lot better on the benchmark Bayesian network than the self-constructed 11-node network. This can partly be explained by the fact that both the network (Asia) and the algorithm used to infer the network (our method) are defined using a Bayesian framework. Secondly, we observed that the models inferred for the 11-node network consisted of a high number of false edges. Moreover, for large datasets, this number of spurious edges increased further. The high false positive rate was probably the main reason why we couldn’t get a strong correlation between accuracy and sample size in the second test set (11-node network). One explanation for the above could be that we had encoded linear as well as non-linear relationships into the 11-node network. The MDL scoring criterion used in the algorithm might have been unable to pickup the non-linear relations from the network and instead introduced spurious edges to try and explain the relationships.

Effect of Noise in Data

The results obtained from introducing noise into the data were unexpected. At first glance, we came to the conclusion that increasing the noise parameter somehow aids the process of structural learning resulting in a higher number of true edges being inferred. However, on performing more detailed analysis, it was revealed that the results varied in their accuracy by a great deal. While some datasets performed extremely well in very noisy conditions, others performed equally bad. Nevertheless, we were unable to define a correlation between noise and accuracy. Intuition tells us that noisy data distorts the actual values, and introduces errors into the learning of network structure. Our results show otherwise. The problem could be due to the method by which noise is incorporated in the dataset. Further analysis is required to assess the affect of noise.

Small Gene Network

It must be noted that the method described in this project can only yet infer a gene network for a small number of genes. Knowing that a real genetic network consists of a large number of interacting genes, is this approach really justifiable?? We think it is. First of all, in Section 1.1.2, we learnt about the properties of real biological networks. Modularity is a key characteristic of genetic networks - genes tend to operate in small clusters. An approach which looks at a small subset of genes in the network and defines their interactions to a fine level of detail will always be more useful than an approach which infers a small number of interactions among a large sparsely connected network. Secondly, microarray datasets contain expression measurements for a large number of genes, but a large proportion of them are not useful in determining a gene network. The expression levels of several genes do not change, or give enough information to infer a network, the genes that do relay information form a small subset and can be used for further analysis. Finally the work of Ott & Miyano [2003] relates the present work and provides methods of extending it to work with larger gene subsets.

5 Conclusion

We now provide a summary of what has been achieved in this thesis. Our goal has been to model the relationships among genes that exist in our biological systems. A Bayesian network framework was used to learn the structure of a network from a given dataset. Bayesian network models were found to be appropriate due to their probabilistic approach. They are known to be good at dealing with the stochastic nature of real biological systems and noisy characteristics of gene expression measurements.

A dynamic programming technique was used to infer the optimal gene model. The algorithm performed well on synthetic datasets, and gave biologically plausible results when applied to real microarray data.

References

[Akutsu et al., 1999] Akutsu, T., Miyano, S., and Kuhara, S. Identification of Genetic Networks from a Small Number of Gene Expression Patterns Under the Boolean Network Model, *Proc. Pacific Symposium on Biocomputing*, 4: 17 – 28, 1999

[Alon 2003] Alon, U. Biological Networks: The Tinkerer as an Engineer, *Science*, 301: 1866 – 1867, 2003

[Brazhnik et al., 2002] Brazhnik, P., Fuente, A., and Mendes, P. Gene Networks: How to put the Function in Genomics, *Trends in Biotechnology*, 20: 11: 467 – 472, 2002

[Brazma et al., 1998] Brazma, A., Jonassen, I., Vilo, J., and Ukkonen, E. Predicting Gene Regulatory Elements in Silico on a Genomic Scale, *Genome Res*. 8: 1202 - 1215, 1998

[Brazma & Vilo 2000] Brazma, A., and Vilo, J. Minireview: Gene expression data analysis, *Federation of European Biochemical Societies*, 480, 17-24, 2000.

[Buntine 1996] Buntine, W. A guide to the literature on learning probabilistic networks from data, *IEEE Transactions on Knowledge and Data Engineering*, 8: 195 – 210, 1996

[Chen et al., 1999b] Chen, T., Hongyu, L. H., and Church, G. M. Modeling Gene Expression with Differential Equations, *Proc. Pacific Symposium of Biocomputing*, 4: 29 - 40, 1999

[Chickering 1995] Chickering, D. M. A Transformational Characterization of Equilavent Bayesian Network Structures, *UAI'95*: 87 – 98, 1995

[Cooper & Herskovits 1992] Cooper, G. F., and Herskovits, E. A Bayesian Method for the Induction of Probabilistic Networks from Data, *Machine Learning*, 9: 309 – 347, 1992

[D'Haeseleer et al., 1999a] D'Haeseleer, P., Liang, S., and Somogyi, R. Tutorial: Gene Expression Data Analysis and Modeling, Session on Gene Expression and Genetic Networks, *Proc. Pacific Symposium on Biocomputing*, 1999

[D'Haeseleer et al., 1999b] D'Haeseleer, P., Liang, S., and Somogyi, R. Genetic Network Inference: From Co-Expression Clustering to Reverse Engineering, *Proc. Pacific Symposium on Biocomputing*, 1999

[De Hoon et al., 2002] De Hoon, M. J. L., Imoto, S., and Miyano, S. Inferring Gene Regulatory Networks From Time-Ordered Gene Expression Data Using Differential Equations, Lange, S., Satoh, K., Smith, C. H. (editors): *Fifth International Conference on Discovery Science*, Germany, 2002

[De Hoon et al., 2003] De Hoon, M. J. L., Imoto, S., Kobayashi, K., Ogasawara, N., and Miyano, S. Inferring Gene Regulatory Networks From Time-Ordered Gene Expression Data Of Bacillus Subtilis Using Differential Equations, *Proc. of the Pacific Symposium on Biocomputing*, 8: 17 – 28, 2003

[De Jong 2002] De Jong, H. Modeling and Simulation of Genetic Regulatory Systems: a Literature Review, *Journal Of Computational Biology*, 9-1: 67 – 103, 2002

[Dejori 2002] Dejori, M. Analyzing Gene Expression Data with Bayesian Networks, *Masters Thesis*, Graz University of Technology, 2002

[Dutilh & Hogeweg 1999] Dutilh, B.E. and Hogeweg, P. Gene Networks from Microarray Data, *Report Bioinformatics*, Utrecht University: (http://www-binf.bio.uu.nl/dutilh/gene-networks), 1999

[Friedman & Goldszmidt 1998] Friedman, N., and Goldszmidt, M. Learning Bayesian Networks with Local Structure, In Jordan, M. I. (Ed.), Kluwer Academic Publishers, pp. 421 – 459, 1998

[Friedman et al., 2000] Friedman, N., Linial, M., Nachman I., and Pe'er, D. Using Bayesian Networks to Analyze Expression Data, *Journal of Computational Biology*, 7: 601 – 620, 2000

[Fuente et al., 2002] Fuente, A., Brazhnik, P., and Mendes, P. Linking the Genes: inferring quantitative gene networks from microarray data, *Trends in Genetics*, 18: 8: 395 – 398, 2002

[Golub et al., 1999] Golub T. R., Slonim D. K., Tamayo P., Huard C., Gaasenbeek M., Mesirov J. P., Coller H., Loh M. L., Downing J. R., Caligiuri M. A., Bloomfield C. D., and Lander E. S. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science*, 286: 531-537, 1999

[Greenbaum et al., 2003] Greenbaum, D., Colangelo, C., Williams, K., and Gerstein, M. Comparing protein abundance and mRNA expression levels on a genomic scale, *Genome Biology*, 4:117, 2003

[Hartemink et al., 2001] Hartemink, J., Gifford, D. K., Jaakkola, T. S., and Young, R. A. Using Graphical Models and Genomic Expression Data to Statistically Validate Models of Genetic Regulatory Networks, *Proc. Pacific Symposium on Biocomputing*, 422 – 433, 2001

[Hartemink et al., 2002] Hartemink, J., Gifford, D. K., Jaakkola, T. S., and Young, R. A. "Combining Location and Expression Data for Principled Discovery of Genetic Regulatory Network Models", Proceedings of Pacific Symposium on Biocomputing, 7: 437 – 449, 2002

[Heckerman 1998] Heckerman, D. A Tutorial on Learning with Bayesian Networks, in M.I. Jordan ed., Kluwer Academic Publisher, 301 – 354, 1998

[Hunter 1993] Hunter, L. AI and Molecular Biology, Chapter 1: 1 - 46, AAAI Press, 1993 :(http://www.aaai.org//Library/Books/Hunter/01-Hunter.pdf)

[Husmeier 2003] Husmeier, D. Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks, *Bioinformatics*, 19 – 17: 2271 – 2282, 2003

[Imoto et al., 2002] Imoto, S., Goto, T., and Miyano, S. Estimation of Gene Networks and Functional Structures between Genes by using Bayesian Networks and Nonparametric Regression, *Proc. Pacific Symposium on Biocomputing*, 7: 175 – 186, 2002

[Imoto et al., 2003a] Imoto, S., Kim, S., Goto, T., Aburatani, S., Tashiro, K., Kuhara, S., and Miyano, S. Bayesian Network and Nonparametric Heteroscedastic Regression for Nonlinear Modeling of Genetic Network, *Journal of Bioinformatics and Computational Biology*, 1:231 - 252, 2003

[Imoto et al., 2003b] Imoto, S., Higuchi, T., Goto, T., Tashiro, K., Kuhara, S., and Miyano, S. Combining Microarrays and Biological Knowledge for Estimating Gene Networks via Bayesian Networks, *Proc. Computational Systems Bioinformatics*, 2003

[Kerr et al., 2001] Kerr, M. K. and Churchill, G.A. Statistical design and the analysis of Gene Expression Microarrays, *Genet. Res*, 77: 123 – 128, 2001

[Kim et al., 2000] Kim, S., Dougherty, E.R., Chen, Y., Sivakumar, K., Meltzer, P., Trent, J. M., and Bittner, M. Multivariate Measurement of Gene Expression Relationships, *Genomics*, 67: 201 – 209, 2000

[Lauritzen et al., 1988] Lauritzen, S. L. and Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems, *Journal Royal Statistics Society B*, 50(2): 157-194, 1988

[Nariai et al., 2004] Nariai, N., Kim, S., Imoto, S., and Miyano, S. Using protein-protein interactions for refining gene networks estimated from microarray data by Bayesian Networks, *Proc. Pacific Symposium on Biocomputing*, 9: 336 – 347, 2004

[Norsys] Network library of Norsys Software Corp. http://www.norsys.com/netlib/

[Ott et al., 2004] Ott, S., Imoto, S., and Miyano, S. Finding Optimal Models for Small Gene Networks, *Proc. Pacific Symposium on Biocomputing*, World Scientific, Singapore, 556 – 567, 2004

[Ott & Miyano 2003] Ott, S., and Miyano, S. Finding Optimal Gene Networks Using Biological Constraints, *Genome Informatics*, 14: 124 – 133, 2003

[Pearl 1988] Pearl, J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, pages 116 – 131, Morgan Kaufmann, San Francisco, CA, 1988

[Pe'er et al., 2001] Pe'er, D., Regev, A., Elidan, G., and Friedman, N. Inferring subnetworks from perturbed expression profiles, *Bioinformatics*, 17, Suppl. 1 (ISMB 2001): S215-224, 2001

[Segal et al., 2002] Segal, E., Barash, Y., Simon, I., Friedman, N., and Koller, D. From promoter sequence to expression: a probabilistic framework, *Bioinformatics, Proc. 6th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 6: 263 – 272, 2002

[Szallasi 2001] Szallasi, Z. Tutorial: Genetic network analysis - From the bench to computers and back, *Proc. 2nd International Conference on Systems Biology*, 2001

[Tabus & Astola 2001] Tabus, and Astola, J. On the Use of MDL Principle in Gene Expression Prediction, *Journal of Applications for Signal Processing*, 4: 297 - 303, 2001.

[Weaver et al., 1999] Weaver, D. C., Workman, C. T., and Stormo, G. D. Modelling Regulatory Networks with Weight Matrices, *Proc. Pacific Symposium on Biocomputing*, 4: 112 – 123, 1999

Appendix A : Results Tables

DAG

	Rep 1		Rep 2		Rep 3		Rep 4		Rep 5		Average	
Number of Samples	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	Average TP Rate	Average FP Rate
10	0.000	0.000	0.125	0.000	0.125	0.036	0.000	0.036	0.000	0.000	0.050	0.014
20	0.250	0.000	0.250	0.054	0.000	0.036	0.125	0.000	0.000	0.036	0.125	0.025
40	0.250	0.018	0.250	0.018	0.500	0.036	0.250	0.000	0.125	0.036	0.275	0.021
80	0.500	0.018	0.250	0.054	0.500	0.054	0.500	0.000	0.250	0.054	0.400	0.036
100	0.250	0.036	0.625	0.018	0.500	0.018	0.625	0.000	0.625	0.000	0.525	0.014
200	0.500	0.054	0.625	0.036	0.750	0.000	0.625	0.054	0.500	0.036	0.600	0.036
400	0.625	0.054	0.875	0.000	0.625	0.036	0.625	0.036	0.500	0.054	0.650	0.036
800	0.625	0.036	0.875	0.000	0.625	0.036	0.500	0.054	0.875	0.000	0.700	0.025
1000	0.750	0.018	0.500	0.054	0.875	0.000	0.875	0.000	0.875	0.000	0.775	0.014

PDAG

	Rep 1		Rep 2		Rep 3		Rep 4		Rep 5		Average	
Number of Samples	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	TP rate	FP rate	Average TP Rate	Average FP Rate
10	0.000	0.000	0.091	0.000	0.091	0.038	0.091	0.019	0.000	0.000	0.055	0.011
20	0.182	0.000	0.182	0.057	0.000	0.038	0.091	0.000	0.000	0.038	0.091	0.026
40	0.182	0.019	0.182	0.019	0.364	0.038	0.182	0.000	0.091	0.038	0.200	0.023
80	0.364	0.019	0.182	0.057	0.545	0.019	0.364	0.000	0.182	0.057	0.327	0.030
100	0.182	0.038	0.545	0.000	0.364	0.019	0.455	0.000	0.455	0.000	0.400	0.011
200	0.455	0.038	0.545	0.019	0.545	0.000	0.636	0.019	0.455	0.019	0.527	0.019
400	0.636	0.019	0.636	0.000	0.636	0.000	0.545	0.019	0.545	0.019	0.600	0.011
800	0.636	0.000	0.636	0.000	0.636	0.000	0.545	0.019	0.636	0.000	0.618	0.004
1000	0.545	0.019	0.545	0.019	0.636	0.000	0.636	0.000	0.636	0.000	0.600	0.008

Performance of Asia Network

DAG

	Rep 1		Rep 2		Rep 3		Average	
Number of Samples	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
10	0.273	0.027	0.182	0.018	0.091	0.036	0.182	0.027
20	0.364	0.082	0.182	0.136	0.364	0.100	0.303	0.106
40	0.455	0.118	0.000	0.136	0.091	0.164	0.182	0.139
60	0.182	0.109	0.455	0.109	0.182	0.164	0.273	0.127
80	0.545	0.127	0.364	0.145	0.455	0.100	0.455	0.124
100	0.455	0.236	0.636	0.255	0.364	0.182	0.485	0.224
200	0.455	0.345	0.364	0.309	0.455	0.336	0.424	0.330
400	0.364	0.245	0.455	0.273	0.273	0.309	0.364	0.276
800	0.364	0.264	0.545	0.255	0.273	0.264	0.394	0.261
1000	0.273	0.264	0.364	0.255	0.545	0.236	0.394	0.252

5000	0.636	0.218					0.636	0.218
10000	0.455	0.273					0.455	0.273

PDAG

	Rep 1		Rep 2		Rep 3		Average	
Number of Samples	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
10	0.250	0.028	0.167	0.018	0.083	0.037	0.167	0.028
20	0.417	0.073	0.167	0.138	0.333	0.101	0.306	0.104
40	0.500	0.110	0.083	0.128	0.167	0.156	0.250	0.131
60	0.250	0.101	0.417	0.110	0.167	0.165	0.278	0.125
80	0.583	0.119	0.333	0.147	0.417	0.101	0.444	0.122
100	0.417	0.239	0.583	0.257	0.333	0.183	0.444	0.226
200	0.417	0.349	0.333	0.312	0.500	0.330	0.417	0.330
400	0.417	0.239	0.583	0.257	0.250	0.312	0.417	0.269
800	0.417	0.257	0.500	0.257	0.333	0.257	0.417	0.257
1000	0.250	0.266	0.417	0.248	0.500	0.239	0.389	0.251
5000	0.583	0.220					0.583	0.22
10000	0.417	0.275					0.417	0.275

JDAG

	Rep 1		Rep 2		Rep 3		Average	
Number of Samples	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
10	0.222	0.019	0.111	0.019	0.056	0.039	0.130	0.026
20	0.222	0.087	0.167	0.136	0.222	0.107	0.204	0.110
40	0.333	0.117	0.000	0.146	0.167	0.155	0.167	0.139
60	0.167	0.107	0.389	0.097	0.111	0.175	0.222	0.126
80	0.444	0.117	0.444	0.117	0.444	0.078	0.444	0.104
100	0.333	0.243	0.611	0.233	0.389	0.165	0.444	0.214
200	0.556	0.320	0.500	0.282	0.500	0.320	0.519	0.307
400	0.222	0.262	0.278	0.291	0.444	0.282	0.315	0.278
800	0.278	0.272	0.611	0.223	0.389	0.243	0.426	0.246
1000	0.333	0.252	0.278	0.262	0.556	0.214	0.389	0.243
5000	0.611	0.194					0.611	0.194
10000	0.444	0.262					0.444	0.262

Performance of 11-node Network (number of samples)

DAG

	Rep 1		Rep 2		Rep 3		Average	
Noise	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
0	0.273	0.264	0.364	0.255	0.545	0.236	0.394	0.252
1	0.545	0.236	0.545	0.236	0.364	0.255	0.485	0.242
2	0.455	0.245	0.364	0.255	0.273	0.264	0.364	0.255
5	0.091	0.282	0.364	0.255	0.455	0.236	0.303	0.258
10	0.273	0.273	0.545	0.236	0.636	0.227	0.485	0.245

20	0.545	0.236	0.455	0.245	0.727	0.209	0.576	0.230
50	0.273	0.264	0.545	0.236	0.545	0.236	0.455	0.245

PDAG

	Rep 1		Rep 2		Rep 3		Average	
Noise	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
0	0.250	0.266	0.417	0.248	0.500	0.239	0.389	0.251
1	0.500	0.239	0.500	0.239	0.333	0.257	0.444	0.245
2	0.417	0.248	0.333	0.257	0.333	0.257	0.361	0.254
5	0.167	0.275	0.417	0.248	0.500	0.229	0.361	0.251
10	0.250	0.275	0.500	0.239	0.583	0.229	0.444	0.248
20	0.500	0.239	0.500	0.239	0.667	0.211	0.556	0.229
50	0.333	0.257	0.500	0.239	0.429	0.243	0.421	0.246

JDAG

	Rep 1		Rep 2		Rep 3		Average	
Noise	TP Rate	FP Rate	TP Rate	FP Rate	TP Rate	FP Rate	Average TP Rate	Average FP Rate
0	0.333	0.252	0.278	0.262	0.556	0.214	0.389	0.243
1	0.611	0.204	0.444	0.233	0.333	0.252	0.463	0.230
2	0.444	0.233	0.611	0.204	0.278	0.262	0.444	0.233
5	0.056	0.301	0.556	0.214	0.444	0.223	0.352	0.246
10	0.167	0.291	0.500	0.223	0.500	0.223	0.389	0.246
20	0.500	0.223	0.556	0.214	0.722	0.175	0.593	0.204
50	0.278	0.262	0.667	0.194	0.444	0.233	0.463	0.230

Performance of 11-node Network (noise)

Appendix B – Source Code

Ott.cpp

```
#include <iostream>
#include <set>
#include <map>
#include <vector>
#include <algorithm>

#include "Scores.cpp"
#include "GenSubset.cpp"
#include "MDL.cpp"
#include "Results.cpp"

using namespace std;

int nvar = 11;
double ns = 1000; // no: samples
double cv = 3; // 2 values
int cal = 2;

int AsiaMat[8] [8];
int SynMat[11] [11];

string Data[] =
{
""
};

map < set < int >, Qset > QMap;
map < Fpair, Fset > FMap;

vector < set < int > >::iterator it;
vector < set < int > >::iterator it2;
set < int >::iterator sit;
set < int >::iterator sit1;
map < set < int >, Qset >::iterator iter;
map < Fpair, Fset >::iterator iter2;

void display( map < set < int >, Qset > a )
{
  for ( iter = a.begin(); iter != a.end(); iter++ )
  {
    display( "QSet =", iter->first );
    cout << ", Score = ";
    display( iter->second );
  }
}

void display( map < Fpair, Fset > b )
{
  for ( iter2 = b.begin(); iter2 != b.end(); iter2++ )
  {
    cout << "F=";
    display( iter2->first );
    cout << ", Score= ";
    display( iter2->second );
  }
}
```

```
void minF( float mval, Fpair mm1, int & mres )
{
  if ( mval <= FMap[mm1].get_Fval() )
  {
    mres = 1;
  }
  else
  {
    mres = 2;
  }
}

int main()
{
  int mres;
  float mval;
  float mqval = 1000000;
  float mqval1;
  set < int, less < int > > s; //geneset
  set < int > tset; //temp set
  set < int > dset; //difference set
  set < int > res; //result set
  set < int > res1; // intern result set
  set < int > mmres; // mdl result set
  set < int > eset; //empty set
  vector < set < int > > SetList;
  vector < set < int > > SList;
  vector < set < int > > S1List;
  Qset Qrec;
  Fset Frec;

  for ( int i = 1; i <= nvar; i++ ) s.insert( i );
  display( "S = ", s );
  cout << endl;

  // Compute F(g,{}) for each gene g: cardinality 0
  cout << "Generating sets of cardinality 0:" << endl;
  cout << "Adding F-scores for each (g,{})" << endl;
  for ( int i = 1; i <= nvar; i++ )
  {
    tset.insert( i );
    float r = mdl( tset, res, nvar, ns, cv, Data );
    Frec.sFval( r );
    Frec.sFpar( res );
    FMap[Fpair( tset, res )] = Frec;
    tset.clear();
  }
  cout << "Adding Q({})" << endl;
  QMap[res] = Qrec;
  //  display( FMap );

  // Compute F(g,A) for each gene g and subset A: cardinality 1
  cout << "Generating sets of cardinality 1:" << endl;
  cout << "Adding F-scores for each (g,{})" << endl;
  for ( int i = 1; i <= nvar; i++ )
  {
    tset.insert( i );
    set_diff( s, tset, dset );
    int kk = dset.size();
    int * string = new int[kk];
    generate( string, 0, 1, kk, dset, & SList );
    delete[] string;
    dset.clear();

    {
      for ( it = SList.begin(); it != SList.end(); it++ )
      {
        res = * it;
        float r = mdl( tset, res, nvar, ns, cv, Data );
        // F(a,{b}) = min { mdl(a,{b}), F(a,{}) }
        minF( r, Fpair( tset, eset ), mres );
        if ( mres == 2 )
        {
          Frec.sFval( FMap[Fpair( tset, eset )].get_Fval() );
          Frec.sFpar( eset );
        }
        else
        {
          Frec.sFval( r );
          Frec.sFpar( res );
        }
        FMap[Fpair( tset, res )] = Frec;
      }
      SList.resize( 0 );
    }
    tset.clear();
  }
  //display (FMap);

  cout << "Adding Q{A} for A of cardinality 1" << endl;
  int * string = new int[nvar];
  generate( string, 0, 1, nvar, s, & SetList );
  delete[] string;
  //display( "List: ", SetList );
  {
    for ( it = SetList.begin(); it != SetList.end(); it++ )
    {
      // Q{a} = F(a,{})
      res = * it;
      Qrec.setQval( FMap[Fpair( res, eset )].get_Fval() );
      Qrec.set_optg( res );
      Qrec.setQpar( FMap[Fpair( res, eset )].get_pars() );
      QMap[res] = Qrec;
      //display( "Qset = ", res );
      //cout << "Score=";
      //display( Qrec );
      // Clear F values of sets cardinality 0
      FMap.erase( Fpair( res, eset ) );
    }
  }
  SetList.resize( 0 );


  for ( int g = 1; g < nvar - 1; g++ )
  //  for ( int g = 1; g < 2; g++ )
  {

    // Compute F(g,A) for each gene g and subset A: cardinality g+1
    cout << "Generating sets of cardinality " << g + 1 << ":" <<
endl;
    cout << "Adding F-scores" << endl;
    for ( int i = 1; i <= nvar; i++ )
    {
      tset.insert( i );
      set_diff( s, tset, dset );
      int kk = dset.size();
      int * string = new int[kk];
      generate( string, 0, g + 1, kk, dset, & SList );
      delete[] string;
      {
        for ( it = SList.begin(); it != SList.end(); it++ )
        {
          res = * it;
          int ss = res.size();
          float r = mdl( tset, res, nvar, ns, cv, Data );
          // F(a,{b}) = min { mdl(a,{b}), F(a,{}) }
          int * string = new int[ss];
          generate( string, 0, g, ss, res, & S1List );
          delete[] string;
          mval = r;
          mmres = res;

          for ( it2 = S1List.begin(); it2 != S1List.end(); it2++ )
          {
            res1 = * it2;
            minF( mval, Fpair( tset, res1 ), mres );
            if ( mres == 2 )
            {
              mval = FMap[Fpair( tset, res1 )].get_Fval();
              //mmres = res1;
              mmres = FMap[Fpair( tset, res1 )].get_pars();
            }
          }
          Frec.sFval( mval );
          Frec.sFpar( mmres );
          FMap[Fpair( tset, res )] = Frec;
          S1List.resize( 0 );
        }
        SList.resize( 0 );
        mmres.clear();
      }
      dset.clear();
      tset.clear();
    }

    //display(FMap);

    cout << "Adding Q{A} for A of cardinality:" << g + 1 << endl;
    int * string = new int[nvar];
    generate( string, 0, g + 1, nvar, s, & SetList );
    delete[] string;
    for ( it = SetList.begin(); it != SetList.end(); it++ )
    {
      res = * it;
      for ( sit = res.begin(); sit != res.end(); sit++ )
      {
        tset.insert( * sit );
        set_diff( res, tset, dset );
        mqval1 = ( FMap[Fpair( tset, dset )].get_Fval() ) + (
QMap[dset].get_Qval() );
        if ( mqval1 <= mqval )
        {
          mqval = mqval1;
          Qrec.setQval( mqval1 );
          Qrec.set_optg( tset );
          Qrec.setQpar( FMap[Fpair( tset, dset )].get_pars() );

        }

        tset.clear();
        dset.clear();
      }
      mqval = 100000;
      QMap[res] = Qrec;
      //display( "Qset = ", res );
      //cout << "Score=";
      //display( Qrec );
    }

    SetList.resize( 0 );

    // Removing F{g,{A}} for A of cardinality g
    cout << " Removing Fvalues for A of cardinality " << g << endl;
    for ( int i = 1; i <= nvar; i++ )
    {
      tset.insert( i );
      set_diff( s, tset, dset );
      int kk = dset.size();
      int * string = new int[kk];
      generate( string, 0, g, kk, dset, & SList );
      delete[] string;
      {
        for ( it = SList.begin(); it != SList.end(); it++ )
        {
          res = * it;
          FMap.erase( Fpair( tset, res ) );
        }
      }
      tset.clear();
      dset.clear();
      SList.resize( 0 );
    }
  }

  cout << "Adding Q{A} for A of cardinality:" << nvar << endl;
  int * sring = new int[nvar];
  generate( sring, 0, nvar, nvar, s, & SetList );
  delete[] sring;
  display( "List: ", SetList );
  mqval = 100000;
  for ( it = SetList.begin(); it != SetList.end(); it++ )
  {
    res = * it;
    for ( sit = res.begin(); sit != res.end(); sit++ )
    {
      tset.insert( * sit );
      set_diff( res, tset, dset );
      mqval1 = ( FMap[Fpair( tset, dset )].get_Fval() ) + (
QMap[dset].get_Qval() );
      if ( mqval1 <= mqval )
      {
        mqval = mqval1;
        Qrec.setQval( mqval1 );
        Qrec.set_optg( tset );
        Qrec.setQpar( FMap[Fpair( tset, dset )].get_pars() );
      }

      tset.clear();
      dset.clear();
    }
    mqval = 100000;
    QMap[res] = Qrec;
    //display( "Qset = ", res );
    //cout << "Score=";
    //display( Qrec );
  }
  SetList.resize( 0 );
  //  display( QMap );

  // Removing F{g,{A}} for A of cardinality nvar-1
  cout << " Removing Fvalues for A of cardinality " << nvar - 1 <<
endl;
  for ( int i = 1; i <= nvar; i++ )
  {
    tset.insert( i );
    set_diff( s, tset, dset );
    int kk = dset.size();
    int * string = new int[kk];
    generate( string, 0, nvar - 1, kk, dset, & SList );
    delete[] string;
    {
      for ( it = SList.begin(); it != SList.end(); it++ )
      {
        res = * it;
        FMap.erase( Fpair( tset, res ) );
      }
    }
    tset.clear();
    dset.clear();
    SList.resize( 0 );
  }

  //  display( FMap );
  //Printing solution in gene matrix form
  {
    for ( int i = 0; i < nvar; i++ )   {
      for ( int j = 0; j < nvar; j++ ) {
        if ( cal == 1 ) {   AsiaMat[i] [j] = 0;  }
        else  {  SynMat[i] [j] = 0;   }
      }
    }
  }

  display( "S = ", s );
  {
    for ( int i = 0; i < nvar; i++ )
    {
      sit = QMap[s].get_optg().begin();
      int l = * sit;
      cout << ( l ) < ",";
      for ( sit1 = QMap[s].get_par().begin(); sit1 !=
QMap[s].get_par().end(); sit1++ )
      {
        int k = * sit1;
        cout << k;
        if ( cal == 1 )
        {  AsiaMat[l - 1] [k - 1] = 1;     }
        else
        {  SynMat[l - 1] [k - 1] = 1;      }
      }
      cout << ")" << endl;
      set_diff( s, QMap[s].get_optg(), dset );
      s.clear();
      s = dset;
      dset.clear();
      display( "S = ", s );
    }
  }
  cout << endl;

//Compare with others to get true positive and false positive rates
if(cal==1)
{
  roc(AsiaMat,AsiaD,nvar);
  roc(AsiaMat,AsiaPD,nvar);
}
else
{
  roc1(SynMat,SynD,nvar);
  roc1(SynMat,SynPD,nvar);
  roc1(SynMat,SynJD,nvar);
}

}
```

MDL.cpp

```
#include <iostream>
#include <stdio>
#include <math>
#include <set>
#include <vector>
#include <map>
#include <string>

#include "factorial.cpp"
#include "mclass.cpp"

using namespace std;

double lg2( double xx )
{
  if (xx==0.0) {
    return 0;
  }
  else{
  return log( xx ) / log( 2 );
  }
}

void display( map < string, mclass > a )
{
  map < string, mclass >::iterator it0;
  for ( it0 = a.begin(); it0 != a.end(); it0++ )
  {
    cout << "( " << it0->first << " , ";
    display( it0->second );
    cout << " ) " << endl;
  }
}


float mdl( set < int > first, set < int > last, int nvar,double
ns,double cv, string Data[] )
{
//  double ns = 10; // no: samples
 // double cv = 2; // 2 values

  map < string, mclass > MMap;
  map < string,mclass>::iterator it1;
  set < int >::iterator its1;
  set < int >::iterator its2;
  mclass mc;
  double cnt0 = 0.0;
  double cnt1 = 0.0;
  double cnt2 = 0.0;
  char psent;

  string hold = "";

  int P = last.size(); // no: parents of given variable

  double bin = combination( nvar, P );
  double mdl_graph = lg2( nvar ) + lg2( bin );
//double mdl_table = 0.5 * pow( cv, P ) * ( cv - 1 ) * lg2( ns );
  double mdl_table = 0.5 * P * ( cv - 1 ) * lg2( ns );
  double mdl_data = 0.0;

  if ( P == 0 )
  {
    its1 = first.begin();
    int sres = * its1;

    for ( int x = 0; x < ns; x++ )
    {
      if ( ( Data[x] ) [sres - 1] == '0' )
      {
        cnt0 = cnt0 + 1;
              }
      else if ( ( Data[x] ) [sres - 1] == '1' )
      {
        cnt1 = cnt1 + 1;
      }
      else
      {
        cnt2 = cnt2 + 1;
      }

    }

  int temp = cnt0 + cnt1+ cnt2;
  mdl_data = -
1*((cnt0*lg2((cnt0/temp)))+(cnt1*lg2((cnt1/temp)))+(cnt2*lg2((cnt2/te
mp))));
  }

  else
  {
    its1 = first.begin();
    int sres1 = * its1;

    for ( int x = 0; x < ns; x++ )
    {
      psent = ( Data[x] ) [sres1 - 1];
      for ( its2 = last.begin(); its2 != last.end(); its2++ )
      {
        int sres = * its2;
        hold = hold + ( Data[x] ) [sres - 1];
      }

      if ( psent == '0' )
      {
        cnt0 = MMap[hold].gf1();
        MMap[hold].sf1( cnt0 + 1 );
        }
      else if ( psent == '1' )
      {
        cnt1 = MMap[hold].gf2();
        MMap[hold].sf2( cnt1 + 1 );
      }
      else
      {
        cnt2 = MMap[hold].gf3();
        MMap[hold].sf3( cnt2 + 1 );
      }
      hold = "";
    }

    mdl_data = 0.0;
    for ( it1 = MMap.begin(); it1 != MMap.end(); it1++ )
  {
    double m1 = it1->second.gf1();
    double m2 = it1->second.gf2();
    double m3 = it1->second.gf3();
    double m4 = it1->second.gsum();

    mdl_data = mdl_data + (m1*lg2(m1/m4))+ (m2*lg2(m2/m4))+
(m3*lg2(m3/m4));
  }

  mdl_data = -1 * mdl_data;
  }

 return mdl_graph + mdl_table + mdl_data;
}
```

Appendix C

Script for generating data for the 11-node network

```
#--------------------------------------------------------------------
# generation of synthetic microarray data with R (include module ggm)
# author: Sascha Ott

# two-dimensional list of functions as input for synthetic_data
single_valued_functions <- list(function(x) 33*(1/(5+exp(x))),
                                                function(x) 0.9*x,
                                                function(x)
sign(x)*(abs(1.4*x)^0.5),
                                                function(x)
1/(0.2+exp(5*x)))
two_valued_functions <- list(function(x,y) 0.7*x+y)
three_valued_functions <- list(function(x,y,z) x+1.1*y+1.4*z)
function_list <-
list(single_valued_functions,two_valued_functions,three_valued_functions)
# n-th item must be a list of n-ary functions

# example graph input
network <- DAG(n1 ~ n1, n2 ~ n2, n3 ~ n3, n4 ~ n1, n5 ~ n2+n3,
             n6 ~ n2+n11, n7 ~ n2, n8 ~ n4+n6+n7, n9 ~ n6, n10 ~ n5, n11
~ n11, order = FALSE)

# function for generation of synthetic data
synthetic_data <- function(graph,functionlist,noiseratio,arraynumber) {
      topologicalsort <- topOrder(graph)
      num_genes <- length(topologicalsort)
      # assign functions to genes
      num_dim <- length(functionlist)
      positions <- vector('integer',num_dim)
      function_assignment <- vector('integer',num_genes)
      for (i in seq(1,num_genes,1)) {
            num_parents <- sum(graph[,i])
            if (num_parents>num_dim)
                  stop("Number of parents too high for supplied
functions!")
            else
                  if (num_parents!=0) {
                        function_assignment[i] <-
positions[num_parents]+1
                        positions[num_parents] <-
positions[num_parents]+1
                        if
(positions[num_parents]==length(functionlist[[num_parents]]))
                              positions[num_parents] <- 0
                  }
      }
      # apply functions
      arraydata <- matrix(0,num_genes,arraynumber)
      for (i in seq(1,num_genes,1)) {
            actualgene <- topologicalsort[i]
            num_parents <- sum(graph[,actualgene])
            parents <- vector('integer',num_parents)
            pos <- 1
            for (k in seq(1,num_genes,1)) {
                  if (graph[k,actualgene]) {
                        parents[pos] <- k
                        pos <- pos+1
                  }
            }
            for (j in seq(1,arraynumber,1)) {
                  if (num_parents==0)
                        arraydata[actualgene,j] <- rnorm(1,mean=0,sd=1)
                  else {
                        if (num_parents==1)
                              arraydata[actualgene,j] <-
functionlist[[1]][[function_assignment[actualgene]]](arraydata[parents[1]
,j])
                        else {
                              if (num_parents==2)
                                    arraydata[actualgene,j] <-
functionlist[[2]][[function_assignment[actualgene]]](arraydata[parents[1]
,j],arraydata[parents[2],j])
                              else {
                                    if (num_parents==3)
                                          arraydata[actualgene,j] <-
functionlist[[3]][[function_assignment[actualgene]]](arraydata[parents[1]
,j],



      arraydata[parents[2],j],



      arraydata[parents[3],j])
                                    else
                                          stop("This case is not
implemented.")
                              }
                        }
                  }
            }
            if (num_parents>0) {
                  # OLD sdthisgene <- (max(arraydata[actualgene,])-
min(arraydata[actualgene,]))*noiseratio
                  sdthisgene <- sd(arraydata[actualgene,])*noiseratio
                  arraydata[actualgene,] <- arraydata[actualgene,] +
rnorm(arraynumber,mean=0,sd=sdthisgene)
            }
      }
      arraydata
}

# function call
#data <- synthetic_data(network,function_list,0.5,10)

#-----------------------------------------------------------------------
-
# normalise and add measurement error (above is only system error)
normalise_and_add_error <- function(data,measurementnoisesd) {
      for (i in seq(1,dim(data)[1],1)) {
            genemean = mean(data[i,])
            genesd = sd(data[i,])
            for (j in seq(1,dim(data)[2],1)) {
                  data[i,j] = (data[i,j]-genemean)/genesd
            }
      }
      for (i in seq(1,dim(data)[1],1)) {
            for (j in seq(1,dim(data)[2],1)) {
                  data[i,j] =
data[i,j]+rnorm(1,mean=0,sd=measurementnoisesd)
            }
      }
      data
}

# function call
#normdata <- normalise_and_add_error(data,0.0)

#-------------------------------------------------------------------------
-
# output data to BN-software-compatible file
write_data <- function(getdata,filename,geneorder) {
      data = matrix(0,dim(getdata)[1],dim(getdata)[2])
      for (i in seq(1,dim(getdata)[1],1)) {
            data[geneorder[i],] = getdata[i,]
      }

      write(t(data),file=filename,append=FALSE,ncolumns=dim(data)[2])
      rownames <- vector('character',dim(data)[1])
      for (i in seq(1,dim(data)[1],1)) {
            rownames[i] <-  paste(sep="",'gene',i,"\t",'gene',i)
      }
      now_as_a_table <- read.table(file=filename,row.names=rownames)
      write.table(now_as_a_table,file=filename,col.names=FALSE,sep='\t')
      }

#function call
#write_data(data,'C:\\Documents and Settings\\Shiva\\My
Documents\\Edinburgh\\Project\\R_outputfile.txt',c(1,2,3,4,5,6,11,7,8,9,1
0)) # ggm orders nodes non-intuitively
#write_data(normdata,'C:\\Documents and Settings\\Shiva\\My
Documents\\Edinburgh\\Project\\R_normout.txt',c(1,2,3,4,5,6,11,7,8,9,10))
# ggm orders nodes non-intuitively

#-------------------------------------------------------------------------
-
# normalise for discrete data
norm_discrete <- function(data) {
      for (i in seq(1,dim(data)[1],1)) {
            genemean = mean(data[i,])
            genesd = sd(data[i,])
            for (j in seq(1,dim(data)[2],1)) {
                  if (data[i,j]< (genemean-genesd))data[i,j] = -1.0
                  else {
                         if (data[i,j]> (genemean+genesd)) data[i,j] =
1.0
                        else data[i,j]=0
                        }
            }
      }
      data
}

# function call
#discretedata <- norm_discrete(normdata)
#write_data(discretedata,'C:\\Documents and Settings\\Shiva\\My
Documents\\Edinburgh\\Project\\R_discout.txt',c(1,2,3,4,5,6,11,7,8,9,10))
```

```
# ggm orders nodes non-intuitively

#-----------------------------------------------------------------------
-
# build series of data sets
produce_data_sets <-
function(network,function_list,systemerror,variance_vector,array_number_v
ector,repetition_number,basicfilename,geneorder) {
      for (i in seq(1,length(variance_vector))) {
            for (j in seq(1,length(array_number_vector))) {
                  for (k in seq(1,repetition_number)) {
                        rawdata <-
synthetic_data(network,function_list,systemerror,array_number_vector[j])

                        normdata <-
normalise_and_add_error(rawdata,variance_vector[i]/100)
                        data <- norm_discrete(normdata)
                        filename =
sprintf("%s%s%s%s%s%s%s%s",basicfilename,"_var",toString(variance_vector[
i]),

"_num",toString(array_number_vector[j]),"_rep",toString(k),".txt")
                        write_data(data,filename,geneorder)
                  }
            }
      }
}

#function call
produce_data_sets(network,function_list,0.5,c(0,1,2,5,10,20,40,50),c(10,2
0,40,50,60,80,100,200,400,600,800,1000,5000,10000),10,
```