Suitability of Bayesian Networks for the Inference and Completion of Genetic Networks using Microarray Data

Abdelkader El Oufir

Master Thesis
Artificial Intelligence
Intelligent Autonomous Systems Group
University of Amsterdam
June 2007

Supervisors:

Frans Voorbraak (Academic Medical Center, Amsterdam)
Arnoud Visser (University of Amsterdam, Intelligent Autonomous Systems)

Acknowledgements

I wish to thank my supervisor Frans Voorbraak for his enthusiasm, guidance and support during this project. It was a lot of fun working with him. I also wish to thank Perry Moerland for bringing the "missing genes" problem to our attention and supplying us with the yeast metabolic network used in this research.

Summary

Genetic networks are models used to depict interactions between genes. With the advent of microarray technology enabling the measurement of gene expression values of thousand of genes simultaneously, inferring these networks from data has recently received considerable attention. In this thesis, the suitability of a probabilistic model called Bayesian networks for the recovery of biological networks from microarray data is investigated. Using standard techniques, Bayesian networks are applied on two different problems, in a way representing extremes of genetic network reconstruction using microarray data.

At one extreme, Bayesian networks are learned from data without using any prior knowledge about the presence or absence of interactions between genes. A biological simulator is used to generate synthetic microarray data for networks with different properties. A Bayesian network inference algorithm is then used to recover networks from this data. It is shown that, given sufficient number of samples, it is possible to recover the structure of these networks using only this data. However, the number of samples required for good recovery grows enormously with the number of regulators per gene in the network. These results are discouraging, as microarray datasets tend to be small in practice.

In contrast, at the other extreme, as more knowledge about pathways is gathered, researchers have made attempts to complete biological networks that are already nearly fully specified. This problem is known as the “missing genes” problem. Here, a probably novel approach based on Bayesian networks is proposed. Using synthetic data, it is shown that networks can be completed with considerably less samples, when compared to learning the complete structure using data only. The method is then applied with at least some partial success on ‘real’ data from *Saccharomyces Cerevisiae* (baker’s yeast), involving microarray measurements for 6207 genes.

Concluding, when learning networks from scratch, the number of samples required for good recovery grows enormously with the number of regulators per gene in the network. As datasets tend to be small in practice, it does not seem reasonable to expect to infer complex structures completely using gene expression values only. However, for the completion of nearly complete networks, the “missing genes” problem, results suggest that sample size requirements are reduced considerably. It seems that with the sample size of current datasets, Bayesian networks can be used to complete genetic networks with some success.

Contents

1 Introduction

Deoxyribonucleic acid (DNA) contains the genetic instructions for the development and functioning of living organisms. Genes are DNA segments which encode instructions for making proteins essential for biochemical processes. Every cell of a particular organism has the same DNA, but at any time, uses only a small part of its genes to produce proteins. Which, how much and when proteins are created, determines the structure and function of cells. In order for a cell to function properly, the amount of each protein produced must be precisely regulated. Understanding the way genes operate and interact with each other could lead to advances in the diagnosis and treatment of diseases. An important goal of molecular biology is therefore to understand the regulatory processes involved in protein synthesis.

The process by which cells produce proteins from the DNA code is called *gene expression*. In essence, gene expression can be considered a two step process: first, specialized proteins *transcribe* a segment of DNA (a gene) into a RNA molecule (also called a *transcript*); then, other proteins process the RNA transcript and *translate* it into a protein. A gene is expressed when its corresponding RNA and/or protein are present. During both steps, the concentration and form of products can be influenced by regulatory molecules (see Figure 1). These regulators are usually fullyformed proteins, but any of the intermediate products (RNA, polypeptides, or proteins) can act as regulators of gene expression (Gardner et al, 2005). Hunter (1993) describes gene expression as an elaborate dance with thousands of participating biological substances; genes code for products that turn on and off other genes, which in turn regulate other genes, and so on.

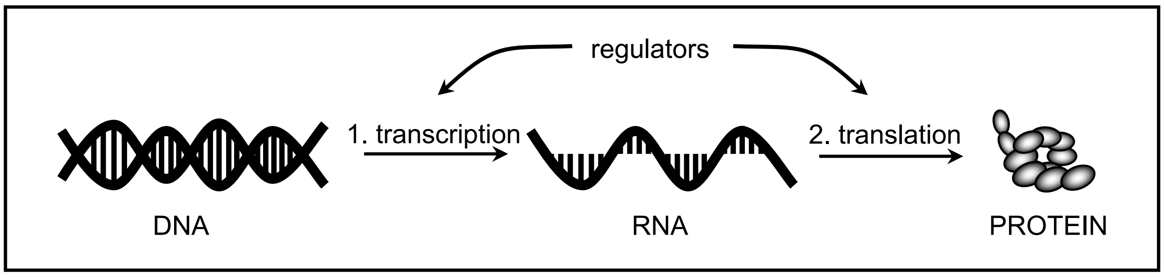


Figure 1: Gene expression is a two step process. First, a segment of DNA – a gene – is transcribed into RNA. Second, RNA is translated into a protein. During both steps, the concentration and forms of the product can be influenced or regulated by other proteins (regulators).

The translation of genes into proteins is not all proteins do in the biochemical activity in a cell. Proteins are in large part responsible for the management of energy-flow, the synthesis, degradation and transport of materials, sending and receiving signals, exerting forces on the world, and providing structural support (Hunter, 1993). These processes are collectively referred to as *metabolism*. The substances consumed and produced in metabolism are called *metabolites*. The biochemical processes in metabolism considered here are *catalyzed reactions*. That is, these reactions are hard to trigger at normal temperatures and pressures, and would hardly take place without the help of other compounds called *catalysts* or *enzymes*. Combinations of these reactions, which accomplish tasks like turning foods into useable energy or compounds, are called *metabolic pathways*.

The relatively new DNA microarray technology has enabled researchers to efficiently measure the concentration of all RNA transcripts in a cell. Measuring protein and metabolite regulators of gene expression is more difficult in general, and such data are not often available (Gardner et al, 2005). In a nutshell, microarray technology consists of glass slides or silicon chips containing thousands of DNA probes, each of which is complementary to a specific RNA species in the cell. Each probe can bind to, and quantitatively measure, the concentration of an individual RNA species. Due to variations in probe sensitivities, the technology can reliably measure only relative changes in RNA concentrations. So, RNA measurements are reported as concentration ratios for each transcript relative to its baseline state. (Gardner et al, 2005).

The advent of microarray data offered the possibility to infer, or “reverse-engineer”, models representing gene networks (or metabolic pathways). A variety of network inference algorithms have recently been proposed for this task. Of these, Bayesian network (BN) inference algorithms seem promising (Handley, 2002; Smith et al., 2002, 2003; Husmeier, 2003). They were first applied to the problem of reverse engineering genetic networks from microarray expression data by Friedman *et al.* (2000), Pe’er *et al.* (2001) and Hartemink *et al.* (2001). A BN is a probabilistic graph model describing the multivariate probability distribution for a set of variables. Within the context of genetic networks, the expression level of each gene should be seen as a random variable, and regulatory interactions between genes as dependencies between the variables. Due to their probabilistic nature, BN can cope with the noise present in microarray measurements, while their graphical nature makes it easy to convey dependencies between genes. BN can capture (non linear) relationships between all the variables in the domain (rather than only between variables and a target variable). BN algorithms can be extended to dynamic Bayesian networks (DBN) to model time series (Friedman et al 1998). Another interesting feature of BN is that prior knowledge can be incorporated into them. For instance, it might be known that some genes are (not) related. Finally, algorithms for learning BN from data are well understood (Heckerman et al 1996).

The inference of networks from microarray data is not easy however, and remains a challenging area of research. Challenges mainly arise from the characteristics of the data; typically noisy, high dimensional, and significantly undersampled (Gardner et al, 2005). Quoting Berlo et al (2003), “due to the high costs of microarrays, the number of time-course measurements are, generally, few with respect to the number of genes (thousands)”. Husmeier (2003) states “ … the application of reverse-engineering techniques to gene expression data is particularly hard in that interactions between hundreds of genes have to be learned from very sparse data sets, typically containing only a few dozen time points during a cell cycle”.

2 Objectives and overview

In this thesis, the viability of Bayesian networks for learning genetic networks from microarray data is investigated. Using standard techniques, the suitability of Bayesian networks is assessed using two different problems encountered in literature, in a way representing two extremes of genetic network reconstruction using microarray data. At one end, learning networks from data without using any prior biological knowledge is examined. At the other end, attempts are made to complete biological networks that are already nearly fully specified. Reflecting this, this thesis is presented in two parts.

In part 1, ‘learning from scratch’, networks are inferred from synthetic data assuming no *a priori* biological knowledge about the presence or absence of connections between genes. The effect of network topology (i.e. structural details of the networks) on the number of samples required for good recovery is investigated using a known simulator of biological networks. Networks of known structure are used to generate synthetic microarray datasets and attempts are then made to recover the structure of these networks from the data using a Bayesian network inference algorithm. Results show that, given enough samples, Bayesian networks can be completely recovered from the data. However, sample size requirements for good recovery grow rapidly with the number of regulators to a gene present in a network, maybe even to an extent that the collection of this data is not practical.

As more and more information about genomes is collected, biologists are able to reconstruct large parts of gene networks. In many cases however, these partially known network contain gaps, meaning that evidence exists that a gene should be present at a particular location in a network, but it is unknown which gene should fill the function. The identification of the genes in a well characterized or nearly complete network has been referred to as the “missing genes” problem (Osterman et al, 2003), or as “filling gaps in metabolic pathways” (Kharchenko et al, 2004). The missing genes problem is the focus of the second part of this thesis. A probably novel approach based on Bayesian networks is proposed. The viability of the approach is demonstrated using a synthetic network. Gaps are created in this network, and the number of samples required to fill the gaps is investigated. Results show that single gaps can be filled using considerably less data than when learning networks from scratch, which of course is an encouraging result. Then, leaving the relatively small synthetic problems behind, the missing genes problem is investigated using ‘real’ data from *Saccharomyces Cerevisiae* (a well known species of yeast), involving a genetic network of hundreds of nodes and microarray

measurements for around 6200 genes. Like before, attempts are made to fill self-made gaps. Experiments demonstrate that discretization is a critical factor in the performance of the Bayesian gene placement algorithm and should be chosen with care. Finally, although the Bayesian approach achieves some reasonable results, it does not outperform a non-Bayesian approach encountered in the literature (Kharchenko et al 2004).

Part 1 Learning from scratch

With the availability of large-scale expression data, effort has been made towards learning gene networks using Bayesian networks and other techniques, as already mentioned in the introduction. In this part of this thesis, the suitability of learning dynamic Bayesian networks from microarray datasets is investigated in a simulation study. In particular, it is explored how the sample size required for good performance depends on the topology of the network, i.e. structural details of the network to be inferred. To this end, both networks consisting of genes regulated by at most one regulator and networks containing a gene regulated by multiple genes are designed in a more or less systematic manner. Using a simulator, datasets of various sizes are sampled from the networks and fed to the inference algorithm. The number of samples required by the algorithm to recover the structure of the known networks is then estimated. It is beyond the scope of this research to develop new Bayesian approaches, or to provide a comparison of existing approaches in order to determine witch is somehow best suited for the task. Here, the focus is on learning Bayesian networks from microarray data using a common, available technique, configured using reasonable assumptions. Finally, although all networks used have known structure, none of this information is supplied to the inference algorithm. All learning is performed without any prior knowledge about the presence or absence of relations between genes; i.e. the structures are learned ‘from scratch’.

The most important reason for using artificial data instead of ‘real’ data in this research is the need to understand and control the problem under investigation. With a simulator, networks with different properties can be simulated at will, and any number of datasets of desired size can be sampled from them according to the purpose of the experiments. As the networks used to generate the data are fully known, the performance of the inference algorithm can be evaluated relatively easily by comparing the known and inferred networks. In contrast, when using real data, recovered relations between genes can not always be validated easily. If the inference algorithm finds a relation that has not been documented in literature, it is not possible to judge its validity without conducting biological experiments. To circumvent this problem, Smith et al (2002), followed by Husmeier (2003), propose using a simulation framework for the evaluation of learning algorithms. As Yu et al (2004) mention, the simulator does not need to be an exact match to, or even model all features of, a real regulatory network, as long as many of the important biological features are modelled. The simulator is useful as long as the qualitative phenomena present in biological systems are exhibited.

Results show that the quality of recovered solutions depends on the amount of gene expression data available to the network inference algorithm. It is found that, given enough samples, Bayesian networks can be completely recovered from the data. However, sample size requirements for good recovery grow considerably with the number of regulators to a gene present in a network. As currently available datasets tend be small, these results are not encouraging. It is however stressed that results are meant to be comparative and should not be taken as absolute for the number of samples required to recover any particular network structure in general. Results are potentially affected by a large number of parameters and it is not possible to investigate them all within this thesis.

The remainder of this part is structured as follows: first, in the following sections, methods used in this research are described, including the biological simulator, Bayesian networks and ways of learning these. In 2.7, a detailed description of experiments conducted in this research is given and results are presented. Finally, results are briefly summarized, followed by a discussion.

2.1 Approach & methods

The experimental framework used for artificial microarray data in this research can globally be described as follows: first, the biological network under investigation is simulated using a plausible simulator. During simulations, the data is sampled at the desired frequency (the sampling rate), as researchers would from a real biological system. The sampled data is fed to a network inference algorithm that attempts to discover the structure of the network used to generate the data. The quality of the recovered network is then determined by comparing it to the network used to generate the data; the more similar the networks, the better the quality of the recovered network. Ideally, both networks are identical. These concepts are illustrated in Figure 2. This experimental framework was introduced by Smith et al (2002) and applied by for instance by Husmeier (2003) and Berlo (2003).

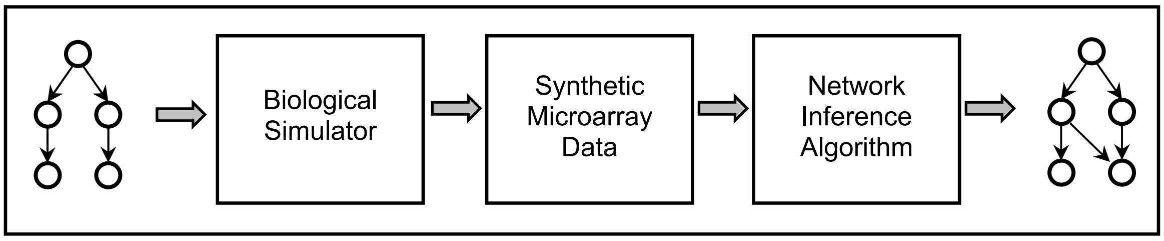


Figure 2: experimental framework for synthetic data experiments. A known gene network is fed to a biological simulator. During simulations, the data is sampled at the desired rate, as researchers would from a real biological system. The sampled data is passed to a network inference algorithm that attempts to discover the structure of the network used to generate the data. Finally, the performance of the algorithm can be determined by comparing the networks. Adapted from Smith et al (2002).

The biological simulator used in this research, GeneSim is a tool designed by Yu and colleagues (2002, 2004) to model genetic networks and sample artificial microarray datasets. With GeneSim, gene networks of arbitrary structure can be simulated and values for gene expression levels can be obtained at discrete time steps. The networks simulated by GeneSim are not Bayesian, introducing a mismatch with the representation used for inferred networks. This contributes to the realism of the experiments, as presumably such mismatches are to be expected in practice. GeneSim is described in detail in 2.2.

The network inference algorithm applied in this research uses Bayesian networks to model biological systems and infer these from microarray data. An introduction to Bayesian networks is given in 2.3, and in 0 different existing strategies to infer these Bayesian networks from data are described. Then, Banjo, the software package used for learning Bayesian networks from data in this research, is presented.

Since the structure of the network used to generate data is known, the performance of the network inference algorithm can be evaluated by comparing the structures of the original or reference network and the recovered network. The metrics used to measure performance are given in 2.6.

2.2 Biological simulation: GeneSim

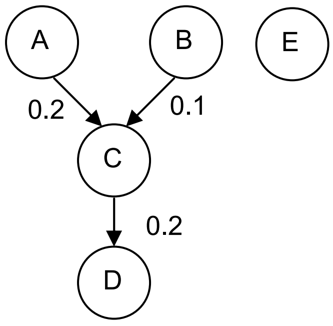
GeneSim is a tool designed by Yu et al (2002, 2004) to model gene networks and produce artificial microarray datasets. With GeneSim, genetic networks of arbitrary structure can be simulated and values for gene expression levels can be obtained at discrete time steps like during microarray experiments.

Genetic networks are encoded in matrices and expression levels are adjusted as a function of the expression level of a gene's regulator(s) and regulation strength. In particular, the expression values of genes are governed by the following equation:

$$Y_{t+1} - Y_t = M(Y_t - T) + \varepsilon$$

In this equation, Y_t is a vector representing the expression levels of all genes at a given time t. To prevent expression levels to become negative or very large, they are kept between 0 and 100 by applying a floor and ceiling function at each update.

M is the matrix encoding the relations existing between the genes in the network. Each column and row of the matrix represents one gene, while matrix cells represent relationships between the genes. Positive values for an element indicate activation of the target gene while negative values indicate inhibition of the target gene. Of course, zero values means the genes do not influence each other. Figure 3: below shows an example network and its corresponding matrix.



	A	B	C	D	E
A	0	0	0.2	0	0
B	0	0	0.1	0	0
C	0	0	0	0.2	0
D	0	0	0	0	0
E	0	0	0	0	0

Figure 3: regulatory network consisting of five genes A,B,C,D and E. On the left, a graphical representation of the gene network is given. To the right, the corresponding matrix GeneSim used by GeneSim. Genes A and B are regulators of gene C, which in turn regulates gene D. Gene E is independent of all other genes since its has neither incoming nor outgoing connections (in the matrix, both the row and column for E contains only zero values).

T holds "constitutive" expression values for each gene in the network (Yu et al 2004). The influence of a gene on a child depends on its difference with its constitutive value (here, all constitutive values are set to 50, halfway the minimum and maximum). If the parent gene is present at a level above its constitutive value, then the effect on its target genes occurs as specified in A; the higher the regulator's level, the stronger the effect on its children. In contrast, if the regulator gene is present at a level below its constitutive value, then its effect is in the opposite direction of that specified in A; the lower the regulator's level, the stronger the opposite effect on its target genes. According to Yu et al (2004), "this is meant to capture basic processes that are not explicitly modelled, such as mRNA degradation or release from repression, that act to return the target gene to its constitutive expression level".

ε is a parameter that is included to model noise. At each time step, for each gene, e is drawn uniformly from -10 to 10 and added to the expression level of the gene.

During simulations, the data are sampled in pre-specified intervals and the samples are saved in a text-file. For example, using a sampling rate of 5 (collecting data every five time steps) the output is the series of expression level vectors (Y_0, Y_5, Y_{10}, ….) analogous to data gathered in a microarray time course experiment.

Example. Suppose that at a given timestep t, for the network given above, the following expression values are known for genes A, B and C: A_t=60, B_t=70and C_t=40. At t+1, recalling that all constitutive values are set to 50, C will be computed as follows:

$$C_{t+1} = C_t + 0.2 * (A_t - 50) + 0.1 * (B_t - 50) + \varepsilon$$
$$C_{t+1} = 40 + 2 + 2 + \varepsilon$$

For convenience, the "effect" of other genes is not included in the equations since their net contribution to C is zero according to the matrix. Ignoring the noise factor ε, the value for C will be increased with 4 to 44. Note that if B_t=30, the change to C would be 0, as B is below its constitutive value and its contribution to C would cancel out A's contribution.

2.3 Bayesian networks

A Bayesian network is a probabilistic graph model which describes the probability distribution for a set of variables. (Pearl (1988); Heckerman et al (1996); Friedman et al (2000)). The main idea behind these networks is that to describe the world, it is not necessary to build a huge joint probability table in which the probabilities for all possible combinations of events are described. Most events are (conditionally) independent of each other, so the way they interact does not have to be considered. Instead, a more compact representation describing groups of events that do influence each other is possible.

Nodes in the graph correspond to observed variables and directed edges (or links, connections, arcs) in the graph denote dependencies between the nodes. If there is a link from a node A to a node B, A is called a parent of B, and B is called a child of A. The absence of edges denotes conditional independence. If there is no link between two nodes A and B, there is a set of variables C such that A and B are independent given C. In more biological terms, the observed variables represent measured gene expression levels, and dependencies between variables denote regulatory interactions between the genes. Parents to variables are referred as regulators.

Stated more formally, a Bayesian network BN for a set of random variables X = {X1, ..., XN} is a pair <G, P>, where G represents a directed acyclic graph (DAG) on X and P is a joint probability distribution on X such that the following Markov condition is satisfied: each variable is independent of its non-descendants, given its parents in G.

The graph G encodes conditional independence assumptions that allow the joint distribution P to be described economically, since, using the Markov condition and the chain rule of probability, the joint distribution can be rewritten into the following product:

$$P(X_i,\ldots,X_n)=\prod_{i=1}^{n}P(X_i|parents(X_i))$$

where parents(Xi) denotes the set of parents for each variable Xi. Conditional probabilities given parents can be stored at each node in so called conditional probability tables (or CPTs).

Figure 4 shows an example of a Bayesian network consisting of the variables X = {A, B, C, D, E}.

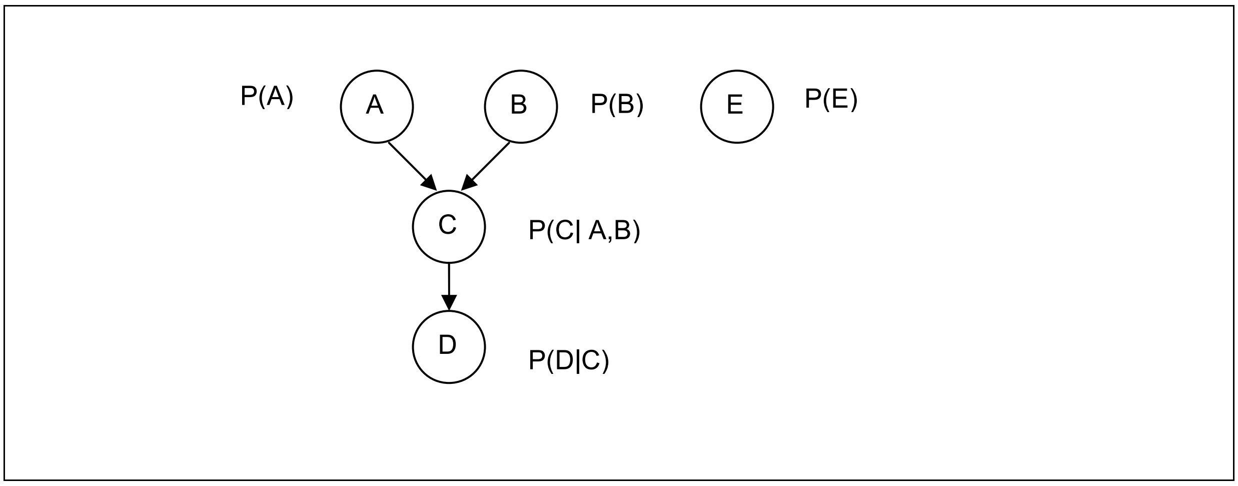


Figure 4: simple Bayesian network consisting of 5 variables A,B,C,D,E. Conditional probabilities given parents are stored at each node. Variable C has two parents, A and B. Variable C is D’s parent. In more biological terminology, this is a simple gene network consisting of 5 genes. Gene C is regulated by genes A and B. Gene C is gene D’s regulator.

Using the chain rule of probability, the joint probability function P(A,B,C,D,E) can be rewritten as:

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

Then, using that each variable is only dependent on its parents and independent of the rest, the joint probability distribution can be written as the product of local dependencies:

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

In the figure it is amongst other represented that E is independent of all other variables, that A and B are independent of each other, that C is independent of other variables given A and B, and that D and all other variables are independent of each other given C.

A *dynamic Bayesian network* (DBN) is a Bayesian network that models the stochastic evolution of a set of variables over time (see e.g. Friedman et al (1998), Neapolitan (2003)). When modelling networks of genes using DBNs, it is assumed that the expression values for genes at a given timestep can be determined using the expression values of the gene itself and its parents at the previous timestep only (variables with Markov lag 1), and never using information from, for instance, 3 steps ago (variables with Markov lag 3). This assumption is called the “first-order Markov assumption”. Another important assumption that is made is that the model is *stationary*, meaning that the transition probabilities do not change over time (thus, “dynamic network” in this context does not mean that the network itself changes over time). Using these assumptions, several researchers have shown that DBNs can be inferred from expression data, see for example Smith et al (2002), Berlo et al (2003). One big advantage of these networks over their “static” counterpart is their ability to model feedback loops, since temporal information is contained in the DAG.

Consider Figure 5. On the left, a dynamic Bayesian network is depicted. It can be used to illustrate the assumptions just described: the expression value for genes is predicted using expression values from the previous timestep only. At t=2, the expression value for B is predicted using the expression values of A and B at t=1. Further, the transition probabilities do not change over time, meaning that the same probabilities are used for transitions from t=1 to t=2, but also from t=2 to t=3 and so on.

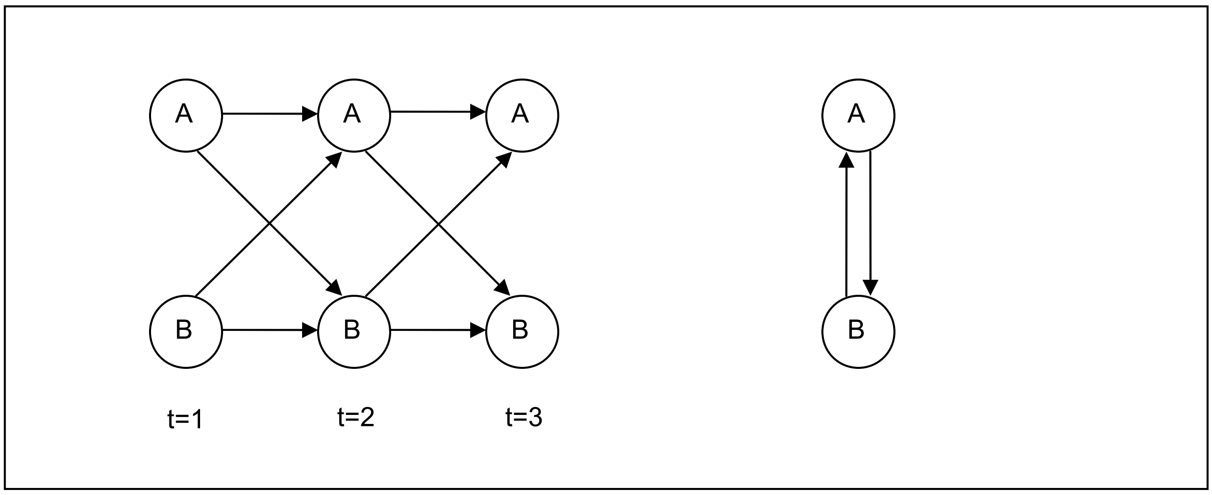


Figure 5: on the left, a dynamic Bayesian network explicitly denoting time. Using that all dependencies are from the previous time slice (first-order Markov assumption), the network can be ‘collapsed’ along the time dimension into the one shown on the right. Time has now become implicit (note that self-loops are not shown either, but are assumed). This figure is taken and slightly adapted from Husmeier (2003).

In the remainder of this thesis however, the factor time is not shown explicitly in graphical examples of networks. Since all dependencies between genes are by definition dependencies from the previous time slice, the temporal aspect can be left out for convenience. This applies for self-loops (transitions from a gene at time t to the same gene at t+1) as well. Each time, it should be evident what kind of network is meant from the context. Thus, the dynamic network on the left is represented by the graph on the right. Note that this graph is not a Bayesian network, since it is cyclic.

2.4 Learning Bayesian networks

Initially, Bayesian networks were constructed by domain experts (Neapolitan 2003). Unfortunately, eliciting Bayesian networks from experts is not easy, especially in the case of large networks. This has motivated researchers to design methods that can learn the networks from data. The problem of learning Bayesian networks from data can be stated as follows: given a dataset, find a Bayesian network BN = <G, P> that best matches this dataset. In order to do this, algorithms for learning Bayesian networks from data consist (at least) of two components: a scoring metric and a search procedure. The scoring metric computes a score indicating how well a given network fits the data. The search procedure tries to generate network structures with high scores. The score and search methods used in this research are the focus of this section. The following is for the largest part based on Heckerman et al (1996) and a textbook by Neapolitan (2003), both tutorials on learning Bayesian networks.

2.4.1 Scoring criterion

A scoring criterion for a Bayesian network is a function that assigns a value to each network under consideration based on the data (Neapolitan 2003). The metric used in this research to evaluate a network given a dataset is the Bayesian Dirichlet equivalent or BDe metric (Heckerman et al 1995). The metric is statistically motivated, derived from the posterior probability of the network structure. Using Bayes' rule the posterior probability can be written:

$$P(B|D) = \frac{P(D|B)P(B)}{P(D)}$$

In this equation, P(B) is the prior probability of the network structure, P(D|B) is the marginal likelihood of the data given the Bayesian network. P(D) does not depend on the network and is not required to compare networks (meaning that P(B|D) is taken to be proportional to P(D|B) P(B)).

The Bayesian scoring metric (BSM) can be generally described (Heckerman et al 1996) as:

$$Score(B:D) = \log P(B|D) = \log P(D|B) + \log P(B) - \log P(D)$$

The likelihood of the data given a network structure P(D|B) can be computed by conditioning on the associated network parameters:

$$P(D|B) = \int P(D|B,Q)P(Q|B)dQ$$

Cooper et. al. (1992) derive a Bayesian metric, called the BD (Bayesian Dirichlet) metric, for learning networks containing only discrete variables, based on multinomial distributions. Building on this work, Heckerman et al. (1995) derive a new metric, the BDe metric (Bayesian Dirichlet equivalent), which has the property of likelihood equivalence. Likelihood equivalence means that the data cannot help to discriminate structures which denote the same independencies. The BDe metric requires the prior over parameters P(Q|B) to have Dirichlet prior distributions and a uniform prior P(B) for the structure. The latter acts as a penalty for complexity, which is essential when learning the structure of Bayesian networks, since the maximum-likelihood network is usually the completely connected network (Friedman et al 1998).

Some properties of these priors are relevant when the data is complete, that is, each instance in the set contains the values of all the variables in the network (Friedman et al 2000). In this case, the following properties hold. First, the priors are structure equivalent, i.e., if two graphs are equivalent they are guaranteed to have the same score. Second, the score is decomposable. That is, the score can be rewritten as a sum where the contribution of every variable to the total network score depends only on its own value and the values of its parents. Finally, these local contributions for each variable can be computed using a closed form equation (see Heckerman et al 1995 for full details).

The BDe criterion is a common criterion and has been applied to learn networks from simulated data by for example Smith et al (2002), Berlo et al (2003).

2.4.2 Searching

Finding the optimal network for a given dataset is not as straightforward as it perhaps may seem at first glance. The number of networks that can be constructed is forbiddingly large, to the extent that it is computationally too expensive to generate and evaluate all possible networks and simply pick the best one. For this reason, researchers have developed heuristic search methods such as greedy search and simulated annealing to identify Bayesian networks with high scores. After a brief discussion of the complexity of structure learning, the Bayesian criterion used in this research and a number of commonly used heuristic methods for searching are described.

2.4.2.1 Complexity of structure learning

For very small problems, all possible networks can be generated and evaluated against the data. The model with the best score can then be determined with certainty. However, this approach is computationally infeasible for larger problems. Neapolitan (2003) gives a recursive formula derived by Robinson (1977) for the number of static networks f(n) that can be constructed using a given number of nodes n:

$$f(n)=\sum_{i=1}^{n}(-1)^{i+1}\binom{n}{i}2^{i(n-i)}f(n-i) \qquad n>2$$

$$f(0)=1$$

$$f(1)=1$$

For illustration purposes: F(3)=25, F(5)=29,000 and F(10) = 4.2 * 10^{18}. For dynamic networks, the number of possible structures G is for a given number of nodes n even larger. Under the assumptions described in the methods section, the following equation can be given for the number of dynamic networks:

$$G(N)=2^{n(n-1)}$$

DAGs can be constructed given n nodes. Thus, G(3)=64, G(5)=1,048,576, G(10)=1.23794 * 10^{27}. Chickering (1996) has proven that the problem of finding the best network is NP-complete.

2.4.2.2 Heuristic methods

In general, search methods for Bayesian networks make successive mutations to the network, using that the effect of local changes can be computed efficiently without recomputing the score for the whole network. At each iteration, the following changes are possible: an existing link between variables can be reversed, or it can be removed. New links can be created to connect variables that were not connected previously. It should however be ensured that all changes result in an acyclic network. If a link to a node X is added or removed, only the scores of X given its parents need to be recomputed to determine the effect of the change. If a link between X and Y is reversed, only the scores for X given its parents and Y given its parents need to be recomputed.

One of the simplest heuristic methods is the greedy hill-climbing search. The method starts with an initial network structure, typically an empty graph, a random graph or a domain specific prior network. At each cycle, possible changes are applied to the current network and the resulting network with the highest score is retained as the current network. The search is terminated when a better network cannot be obtained by applying one of the local changes.

A common problem with greedy search is that the algorithm gets stuck in a local optimum and halts without returning the optimal model. A local optimum in this case is a Bayesian network that is better than all other networks that can be produced by applying one of the local changes. This problem is partly overcome by repeating the greedy search several times, each time starting with another (random) initial network (e.g. Heckerman 1996, Berlo et al 2003, Yu et al 2004). Stopping criteria for a search could be a maximum number of restarts, a number of evaluated networks or simply a specified period of time a researcher is willing to wait. Although this procedure does not necessarily find a global maximum, it is reported to perform well in practice (see for example Hartemink et al 2002).

Simulated annealing is another heuristic search method (Heckerman 1996, Hartemink et al 2002). Designed to escape local maxima, it allows some changes that cause the score of the network to decrease. It starts with an initial network structure and picks an edge change from the set of possible mutations to the network at random. The change is accepted as an improvement to the current network if its score is higher. If its score is lower, it is accepted with a probability based on a system parameter T known as the “temperature”. As the process is repeated, the value for T is lowered gradually. At the start of the process, when the temperature is high, a lot of changes are accepted, even if the score is not improved. As the temperature decreases, less changes are accepted. The idea here is to do enough exploration of the space of possible networks early on so that the final solution is relatively insensitive to the starting state. This should lower the chances of getting caught at local maxima.

The discussion of search algorithms is limited to the ones given above as these are part of the software package used in this research. Naturally, other search heuristics exist. For instance, (Heckerman 1996) also mention “best-first search”. In this approach, the space of all network structures is searched systematically using a heuristic measure that determines the next best structure to examine. Another example can be found in (Berlo et al 2003), where “beam-search” is applied. This algorithm starts by selecting from all possible structures with a single parent the K networks with the highest score. Only for these networks are all combinations with a second parent evaluated. Again, the best K are selected upon which the procedure is repeated. (Yu et al 2004) describe an “evolutionary algorithm” to search for high scoring networks.

2.5 Banjo

Banjo (Bayesian Network Inference with Java Objects) is the software application used in this research for learning the structure of Bayesian networks from data, implementing the concepts described earlier. Banjo is developed under the direction of Alexander J. Hartemink in the Department of Computer Science at Duke University. Banjo focuses on structure inference of (static and dynamic) Bayesian networks. Available heuristic search strategies include greedy hill-climbing and simulated annealing (as described earlier). Networks are evaluated with the Bde scoring metric as described by Heckerman (1996). Banjo can return any number of highest scoring networks that were found in the search. Banjo can be downloaded from [Banjo Homepage].

Banjo’s behaviour is determined by a set of parameters stored in a settingsfile. In this file, users can specify the location of the dataset, which discretization methods and search strategy should be used etc. The file and settings that are most relevant to this research are discussed in 2.5.1. In order to give the reader a good impression of Banjo’s workings, a sample run and screenshots are provided in 2.5.2. It is however beyond the scope of this section to describe all Banjo’s features in detail; the interested reader is referred to the User Guide ([Banjo Homepage]).

2.5.1 Configuring Banjo

Before attempting to find high scoring networks with Banjo, several decisions have to be made. For example, it should be decided what kind of networks should be inferred (e.g. static or dynamic), which discretization method, scoring metric should be applied etc. These choices can be passed to Banjo as arguments in the command line or using a settingsfile (plain text). This file can be divided in a number of sections, as described below. Only the most important and relevant sections for this research are described (see the User Guide for a complete description [Banjo Homepage]).

Scoring metric

There is currently only one metric available in Banjo to compute a network's score: the BDe metric (see the methods section). The value of setting evaluatorChoice in the settings file determines the metric that will be used. The valid choices are "default" and "EvaluatorBDe" which will both cause Banjo to select the BDe metric for the evaluation of networks.

Discretization

If the data used is continuous, it has to be discretized as the BDe score is based on discrete multinomial distributions. Banjo provides 2 simple discretization strategies: interval and quantile discretization. Discretization policy is controlled by two settings. The first, called discretizationPolicy, specifies a default policy for all variables; the second, called discretizationExceptions, specifies a list of potential exceptions to the default policy. The default policy can be either the token "none" or a token like "q2" or "i4". The latter start with "q" for quantile discretization or an "i" for interval discretization, followed by a number specifying the desired number of discrete values.

Network properties

Banjo can learn static and dynamic Bayesian networks from data. This is determined by the settings minMarkovLag and maxMarkovLag. To learn static networks, the minMarkovLag and maxMarkovLag are set to 0. For the inference of dynamic networks, both are to set to 1, under the first-order Markov assumption (meaning that the expression value of a gene at a given step can be predicted using expression values of genes at the previous step only).

Another parameter that affects network structure is maxParentCount. This parameter determines the maximum number of parents any node is allowed to have. This is useful to limit the number of possible networks and memory requirements. The User Guide warns: "Note that any number greater than 4 or 5 probably won't make much sense for the underlying problem. Generally, numbers greater than 7 will raise the memory requirements for Banjo substantially."

Search specifications

Banjo implements two basic search strategies: greedy search with restarts and simulated annealing (setting searcherChoice). The way changes to the current network are applied is determined by setting proposerChoice: a local mutation can be chosen at random (randomLocalMove), or all changes arising from a single addition, deletion, or reversal of an edge in the current network (AllLocalMoves) can be applied to the current network.

The greedy algorithm will accept a new network as the current if and only if its score is better than or equal to that of the current network. The simulated annealing algorithm will accept a network if its score is higher, a network with a lower score is accepted with a probability based on system parameter known as the temperature.

The exact behaviour of the greedy search algorithm with random restarts in Banjo is configured with the following parameters in the settings file:

- maxProposedNetworks: the total number of networks the algorithm is allowed to evaluate during the search before exiting and returning the results of the search.

- minProposedNetworksAfterHighScore: the minimum number of search iterations that Banjo will execute after it has found a new high score, before a restart is initiated.

- minProposedNetworksBeforeRestart: the minimum number of search iterations that Banjo will execute after a restart, before the next restart is initiated.

- maxProposedNetworksBeforeRestart: the maximum number of search iterations that Banjo will execute after a restart, before the next restart is initiated.

The minimum and maximum number of restarts can then be estimated from the above settings as follows:

- minimum number of restarts is maxProposedNetworks divided by maxProposedNetworksBeforeRestart

- maximum number of restarts: maxProposedNetworks divided by minProposedNetworksBeforeRestart

Stopping criteria

There are a number of ways to define stopping criteria for a search. A search can be stopped after an amount of time (as specified by maxTime), or after a certain number of networks is evaluated (maxProposedNetworks). Another way is to stop after a certain number of restarts, as specified by Maxrestarts (or using settings maxProposedNetworks and the number of proposed networks before a restart, as described above in search specifications).

Initial network

Banjo can be supplied with an initial network structure. This structure can be used as starting point for a search. Also, it is possible to supply a network and let Banjo only compute the score of the network given the data, by setting the maxProposedNetworks parameter to 0. This feature of Banjo is used extensively in this research: in part 1, to determine the score of a known network used to generate a particular dataset and compare it to the score of the best network returned by the inference algorithm; in part 2, the feature is used to evaluate candidates for gaps in known networks.

The text file containing the initial structure can be specified using the setting initialStructureFile. The format in which the network should be supplied will be discussed later, as Banjo uses the same format to output recovered solutions.

2.5.2 Sample Output

Banjo is not provided with a windows based graphical user interface; it can be executed from the command line or shell. Results of the search are displayed on screen and to a specified text-file. When executed, Banjo first displays the settings it has been supplied with:

```
-----------------------------------------------------------------------------
- Banjo                        Bayesian Network Inference with Java Objects -
- Release 1.0.5                                                 30 Nov 2005 -
- Licensed from Duke University                                             -
- Copyright (c) 2005 by Alexander J. Hartemink                              -
- All rights reserved                                                       -
-----------------------------------------------------------------------------
- Project:                                             banjo dynamic example
- User:                                                                 demo
- Data set:                               10-vars-1500-temporal-observations
- Notes:                                  dynamic bayesian network inference
-----------------------------------------------------------------------------
- Searcher:                                                   SearcherGreedy
-----------------------------------------------------------------------------
- Settings file:                           data\dynamic\dynamic.settings.txt
- Input directory:                                        data/dynamic/input
- Observations file:                                                   1.dat
- Output directory:                                      data/dynamic/output
- Report file:                                                         1.txt
- Variable count:                                                         10
- Number of observations:                                               1500
- 'Effective' number of observations with DBN:                          1499
- Discretization policy:                                                  i3
- Discretization exceptions:                                            none
- Min. Markov lag:                                                         1
- Max. Markov lag:                                                         1
- DBN mandatory lag(s):                                                    1
- Equiv. sample size:                                                    1.0
- Max. parent count:                                                       5
- Initial structure file:
- 'Must be present' edges file:
- 'Must not be present' edges file:
- Max. time:
- Max. proposed networks:                                              10000
- Max. restarts:
- Min. networks before checking:                                        1000
- Number of best networks tracked:                                         3
- Number of progress reports:                                             10
- Write to file interval:
- Statistics:                                               RecorderStandard
- Proposer:                                            ProposerAllLocalMoves
- Evaluator:                                       defaulted to EvaluatorBDe
- Cycle checker:                                             CycleCheckerDFS
- Decider:                                        defaulted to DeciderGreedy
-----------------------------------------------------------------------------
- Min. proposed networks after high score:                              1000
- Min. proposed networks before restart:                                3000
- Max. proposed networks before restart:                                5000
- Restart method:                                         use random network
-   with max. parent count:                                                3
-----------------------------------------------------------------------------
```

The feedback about settings is given in a number of sections. Below the general header info indicating the Banjo version being used, the general project information is listed, including four free-form settings 'project', 'user', 'data set', and 'notes'. Then, the general parameters that set up a search, including the names and locations of input and output files, discretization of the supplied data, stopping criteria (in terms of time, number of networks, or number of restarts), number of high-scoring networks tracked, frequency of intermediate feedback and writing results to file, and the names of the core components. Finally there is a section of parameters that are specific to the search strategy being selected.

In this case, as can be seen, Banjo is supplied with 1500 samples (‘number of observations’) which are to be discretized into 3 intervals of equal width (as indicated by ‘i3’ for parameter ‘Discretization policy’). Banjo is directed to learn a dynamic network (since both Markov-lags are set to 1) and is allowed to evaluate 1000 networks (maxProposedNetworks) during the greedy search.

Immediately after that, Banjo displays a discretization report that lists some important characteristics of the data supplied. Here, the 10 variables are all discretized into 3 intervals of equal width (i3) and have values between 0 and 100.

```
---------------------------------------------------------------------------------
- Pre-processing                                            Discretization report
---------------------------------------------------------------------------------
 Variable | Discr.  | Min. Val.  | Max. Val.  |  Orig.  |  Used  |
          |         |            |            | points  | points |
---------------------------------------------------------------------------------
      0   |    i3   |      0.0   |     100.0  |  1381   |     3  |
      1   |    i3   |      0.0   |     100.0  |   646   |     3  |
      2   |    i3   |      0.0   |     100.0  |   240   |     3  |
      3   |    i3   |      0.0   |     100.0  |   146   |     3  |
      4   |    i3   |      0.0   |     100.0  |  1392   |     3  |
      5   |    i3   |      0.0   |     100.0  |  1379   |     3  |
      6   |    i3   |      0.0   |     100.0  |  1374   |     3  |
      7   |    i3   |      0.0   |     100.0  |  1396   |     3  |
      8   |    i3   |      0.0   |     100.0  |  1381   |     3  |
      9   |    i3   |      0.0   |     100.0  |  1378   |     3  |
---------------------------------------------------------------------------------
```

Banjo then provides periodic feedback on its progress, and, when the search is completed, it supplies the final results. In this particular case this includes the 3 highest scoring networks, the statistical information about the search, and a basic output of the best network for generating a graph in dot.

The best networks are displayed in the following format (here, only the best network is shown, as no purpose is served by displaying them all):

```
Network #1, score: -6308.514426, first found at iteration 271
10
 0   0:   0                1:   1 0
 1   0:   0                1:   2 0 1
 2   0:   0                1:   2 1 2
 3   0:   0                1:   2 2 3
 4   0:   0                1:   1 4
 5   0:   0                1:   1 5
 6   0:   0                1:   1 6
 7   0:   0                1:   1 7
 8   0:   0                1:   1 8
 9   0:   0                1:   1 9
```

The first line indicates the ranking of the network (here: #1), and its associated BDe score (in this case: -6308.514426), first encountered at iteration 271. The second line indicates the number of variables used to model the problem, in this case 10.

Recall that the network is dynamic under the first-order Markov assumption, e.g. it is assumed that the expression values for genes at a given timestep can be determined using the expression values of the gene itself and its parents at the previous timestep only (variables with Markov lag 1, see 2.3). For each node, the parents in each lag are listed in a separate “block”, starting with the respective Markov lag and a colon (“:”). The last 10 lines (one for each of the 10 variables) first list the id of a variable, the block for Markov lag 0 (here, “0: 0” indicates that there is no parent of Markov lag 0), and then the block for Markov lag 1 (here, for node id = 1, “1: 2 0 1” indicates that variable id = 1 has 2 parents of Markov lag 1, namely variable id = 0 and variable id = 1). The network represented here will be displayed further on. (As already alluded to, the format shown here for networks is the same used to supply a known network to Banjo).

For the highest network it has found, Banjo creates a set of instructions in so called “dot format”. This is a format used by the free GraphViz library from AT&T. Given a text file in this format, GraphViz dot lays out graphs and then generates images. In this case, the following dot commands are generated for the best network:

```
-----------------------------------------------------------------------------
- Post-processing                                  DOT graphics format output
-----------------------------------------------------------------------------

digraph abstract {

label = "Banjo Version 1.0.5\nHigh scoring network, score: -6308.51\nProject: banjo
dynamic example\nUser: demo\nData set: 10-vars-1500-temporal-observations\nNetworks
searched: 10801";
labeljust="l";

    0->1;
    1->2;
    2->3;
}
```

Then, after copying this string into a textfile, the following graphical representation can be generated with GraphViz dot:

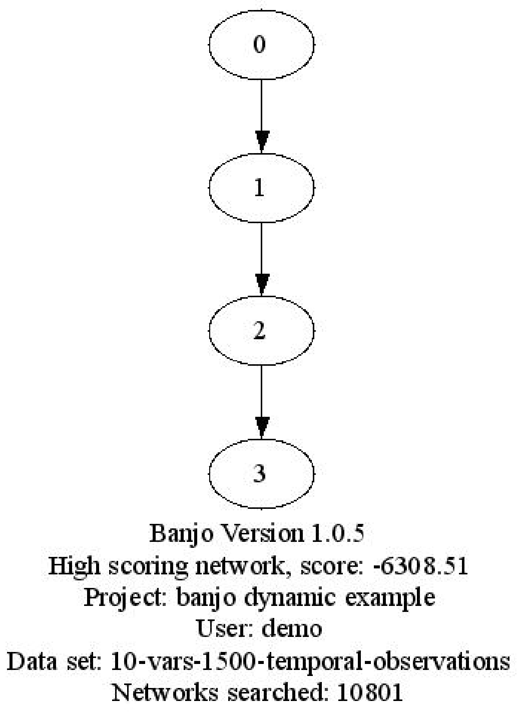

0
1
2
3
Banjo Version 1.0.5
High scoring network, score: -6308.51
Project: banjo dynamic example
User: demo
Data set: 10-vars-1500-temporal-observations
Networks searched: 10801

Figure 6: example dynamic Bayesian network generated using GraphViz dot

Recall that although time is not represented explicitly in Figure 6, the network is dynamic, i.e. the links between genes in the figure denote dependencies in time. The expression value of genes is determined using the expression values of the genes at the previous timestep only (see the Bayesian networks section for more details).

2.6 Evaluating performance

The performance of the Bayesian network inference algorithm is evaluated by comparing the reference networks (those used to create the data) with the networks recovered by the algorithm from the data. The more similar the networks, the better the quality of the solutions. Ideally, reference and recovered networks are identical, meaning that all the links in the reference network are recovered, and that the recovered network does not contain connections that do not exist in the reference network. This leads to the following definitions:

- If a link exists both in the reference network and the recovered network, it is considered a *true positive* (TP)
- If a link exists in the reference network but is not recovered, it is called a *false negative* (FN).
- When an edge is absent in both networks, it is counted as a *true negative* (TN).
- Finally, if a recovered link does not exist in the reference network, it is called a *false positive* (FP).

These measures are then used to calculate the quality of a recovered network in terms of its sensitivity and specificity. The sensitivity of a model is the proportion of true positives and is computed as follows:

$$Sensitivity = \frac{TP}{TP + FN}$$

The specificity or the fraction of true negatives is calculated as follows:

$$Specificity = \frac{TN}{TN + FP}$$

These measures are well known; see for instance Husmeier (2003). If the discovered network contains all the links present in the reference network (or more), the sensitivity is 1 (or 100%). The specificity on the other hand, is 1 when no spurious link is found. Both are 1, the perfect score, only when the structure of the discovered and reference networks are identical.

2.7 Results

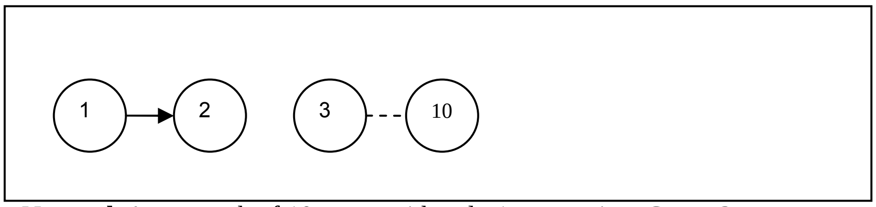
The experimental framework is applied to investigate the recovery of *dynamic* Bayesian networks from artificial microarray data. The experiments are made no more complicated than necessary. The aim of the experiments presented here is to investigate the impact of the network structure on the required number of samples for good recovery, and not the impact of the network size. As will become apparent, this effect of structure can be investigated with very small networks. Again, it is stressed that results presented in this section are meant to be comparative and should not be taken as absolute for the number of samples required to recover any particular network structure in general. Results are potentially affected by a large number of parameters and it is not possible to investigate them all within this thesis.

It is shown empirically that the topology of the network, i.e. structural details of the network, is an important factor for the number of samples required for good structure recovery. More specifically, the required amount of data is strongly influenced by the maximum number of parents per node in a network rather than by the number of connections in the network. To show this, regulatory networks with different structures are designed in a more or less systematic manner, and required sample size for good recovery is estimated.

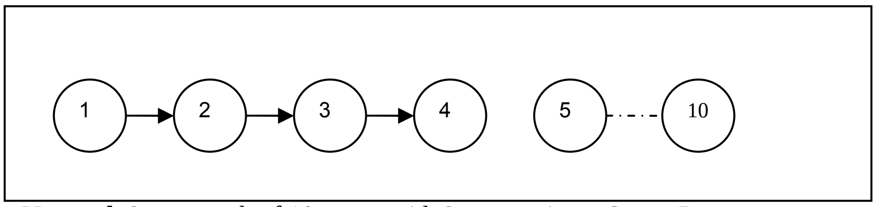
The remainder is structured as follows: first, experiments with regulatory networks consisting of only single parent connections are conducted. The networks used are described in detail and required sample size for recovery of the networks from the data is investigated. The same procedure is then repeated for networks containing a gene regulated by an increasing number of parents. In 2.7.3, the choices made for the search parameters used in the experiments are clarified. Finally, results are summarized and discussed.

2.7.1 Learning single parent networks

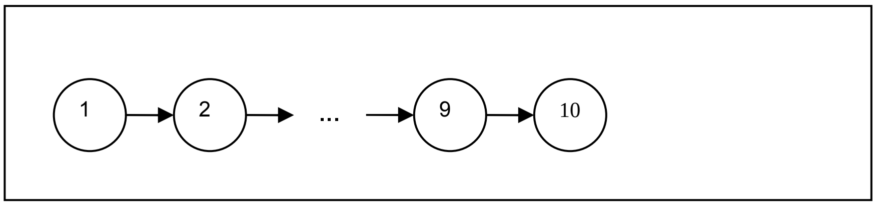
The first experiments with synthetic data presented here involve networks of genes consisting of a number of single parent connections only. Four networks of 10 genes containing 1 to 9 connections are constructed and sampled using GeneSim. The networks used are depicted below.



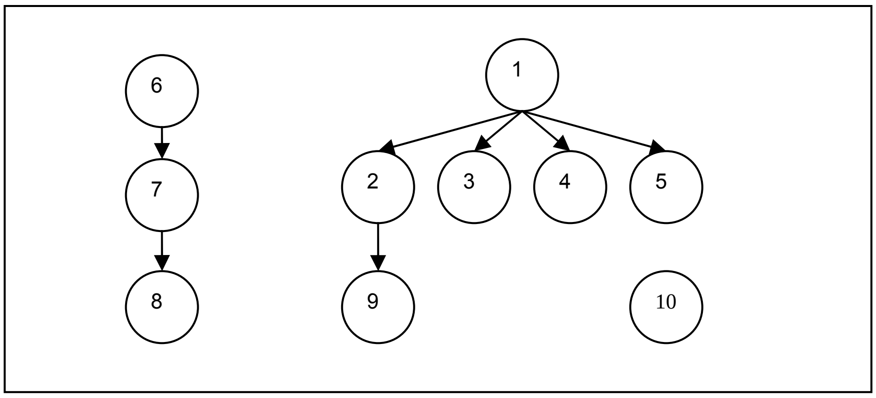
Network 1: network of 10 genes with only 1 connection. Genes 3 To 10 are not regulated and act as distracters.



Network 2: network of 10 genes with 3 connections. Genes 5 to 10 are not regulated and act as distracters.



Network 3: network of 10 genes with 9 connections. All genes are regulated except gene 1. There are no distracter genes in this network.



Network 4: network with 7 single parent connections distributed over 2 independent subnetworks. Gene 1 is connected to 4 other genes.

Genes that are not regulated by any other gene behave according to the noise level in GeneSim. Genes not connected to other genes at all act as distracters to the DBN inference algorithm. In all networks depicted, connections between genes are assigned positive regulation strength of 0.2. This corresponds to the highest value used in Yu et al (2002, 2004).

All experiments with the single parent networks are conducted using datasets of 7 different sizes ranging from 50 to 5000 in the number of samples. For each sample size, 10 independent datasets were generated, effectively yielding a total of 70 datasets per network. As Yu et al (2004) obtained the best results using a sampling interval of 5 time steps with GeneSim and a discretization into 3 categories, the same settings were used in these experiments. The DBN inference algorithm was applied 5 times on each dataset using greedy search with random restarts, considering all possible local moves at each iteration. The algorithm was allowed to evaluate a maximum of 100000 networks after which the search was terminated. The choices for the particular search strategy and settings are clarified in section 2.7.3.

Table 1 shows the obtained sensitivity and specificity scores for the described single parent networks. The table shows average scores since the experiments were repeated 5 times on 10 independent datasets for each sample size.

For interest, the (Banjo) BDe scores of the ‘true’ networks were computed for each dataset and compared to the scores obtained by Banjo during each of the runs. Every network returned by Banjo was found to be at least as good as the true network. In other words, every time Banjo did not return the true network, this was justified by the BDe score of the recovered network and this was not caused by the particular choice of search strategy and settings.

Not surprisingly, these results show that the quantity of data is an important factor in the recovery of the structures. The sensitivity scores increase for all four networks as a function of the available number of samples. Close to perfect scores for both sensitivity and specificity are obtained with datasets consisting of 250 to 500 samples.

	Network 1		Network 2		Network 3		Network 4	
Samples	Sens.	Spec.	Sens.	Spec.	Sens.	Spec .	Sens.	Spec.
50	50.00	95.86	68.00	94.27	77.56	95.30	38.57	90.84
100	60.00	99.49	80.00	98.66	93.33	96.81	70.86	93.05
250	100.00	99.80	93.33	99.48	100.00	99.12	95.14	97.83
500	100.00	100.00	100.00	99.90	100.00	99.78	100.00	100.00
750	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1000	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
5000	100.00	98.99	100.00	99.18	100.00	98.79	100.00	97.85

Table 1: average sensitivity and specificity scores obtained by the Bayesian inference algorithm on the single parent networks. Averages are calculated on 10 datasets for each sample size. A sensitivity of 100 means that all the links in the original network have been recovered. A specificity of 100 means that no spurious links were recovered. When both are 100 the original and recovered networks are identical. (Very close to) perfect scores are obtained using 250 to 500 samples. For these networks, the number of connections does not seem to have significant impact on the number of samples required for perfect recovery.

In spite of the different number of links in the networks, no substantial difference in the required dataset size for good recovery is observed between the networks consisting of 3, 7 or 9 links, especially when compared to the results of the multi-parents experiments of 2.7.2.

Although specificity initially increases with sample size, it is lower for the datasets consisting of 5000 samples for all networks. With these larger datasets, the DBN algorithm starts to learn connections that are not present in the original networks used to generate the data (false positives). This effect has been observed and reported by Yu et al (2004); these researchers attributed this to the fact that the original networks are not Bayesian. Because of the mismatch between the dynamic Bayesian network and GeneSim, supplying more data results in over-fitting the network to the data, leading to the inclusion of false positives.

When comparing the sensitivity scores for regulatory networks 1, 2 and 3, at first glance it seems surprising that higher scores are obtained for networks 2 and 3 when using datasets of size 50 and 100 samples in spite of the larger number of connections. In order to understand this better, the recovered networks for networks 1 and 3 are examined more closely for the datasets consisting of 100 samples since the difference for this sample size seems most outspoken. The first thing noticed is that for both networks, the 5 runs for each of the 10 datasets resulted in the same top network so the results can be discussed as if only 1 run was conducted on each of the datasets. For network 1, the only link present in the network (from gene 1 to gene 2) is recovered by the DBN inference algorithm 6 times out of 10. Recalling that the sensitivity is calculated as the number of connection recovered that are also present in the original network divided by the total number of connections present in the original network, this results in an average sensitivity of 60% for network 1.

For network 3, the situation is slightly more complicated since more links are involved:

In 6 cases out of 10, all 9 links present in the original network are recovered. This results in a sensitivity of 100% in 60% of the trials.

In two cases, 8 links out of 9 are recovered, but the link from gene 1 to gene 2 is not found. This results in a sensitivity of 88.89% in 20% of the trials.

In the remaining two cases, 7 links out of 9 are recovered, again without finding the link from gene 1 to gene 2. This results in a sensitivity of 77.78% in 20% of the trials.

The average sensitivity for network 3 is then calculated as (0.6*100) + (0.2*88.89) + (0.2*77.78) = 93.33%. The fact that the other connections besides the connections from gene 1 to gene 2 are frequently recovered results in a higher sensitivity for network 3.

The link from gene 1 to gene 2 is recovered as frequently for both networks (6 times out of 10). For network 3, the remaining links are recovered in almost all trials. Since gene 1 is the only gene without a parent in both networks, these findings suggest that when a gene has a grandparent (the gene's parent has a parent), the DBN inference algorithm is more likely to recover the connection between the gene and its immediate parent.

Knowing GeneSim's dynamics, the behaviours of (un)regulated genes can be explained as follows: at any given time step, the expression level of an unregulated gene is determined by its value in the previous time step and the specified noise level in GeneSim. This means that in the long run such a gene performs a random walk between the minimum and maximum expression levels (in this case 0 and 100). On the other hand, the expression level of a regulated gene depends on its previous value, the noise and also on the expression level of its regulator at the previous time step. For each time step that the expression level of the regulator is (below) above its constitutive level (in this case 50), the expression level of the regulated gene is (decreased) increased according to the equation described in the methods section. This means that - unless the regulator oscillates very close to its constitutive level - a gene with a parent is directed towards its minimum or maximum expression levels and it is less likely to have an expression value in between. To illustrate this, a large dataset sampled from network 3 is selected and the expression levels of genes 1, 2 and 3 for 300 consecutive samples are plotted (Figure 7). The plot confirms the analysis just given.

For a more quantitative view, expression values in the selected dataset are discretized into three categories 'low', 'normal' and 'high' (in the same way as Banjo would, each category spanning one third of the range 0..100). For genes 1, 2 and 3 in the datasets relative frequencies for each value are determined, as shown in Table 2. As expected, the table shows that the three categories are approximately equally probable for gene 1. For the regulated genes 2 and 3, the frequencies confirm that the values 'low' and 'high' are much more probable than the value 'normal'.

	Relative frequencies (%)		
Gene	Low	Normal	High
1	34.67	32.19	33.14
2	47.30	7.58	45.12
3	49.36	3.04	47.60

Table 2: relative frequencies of values for the genes in network 3

Then, the conditional probabilities of the values of gene 2 given the previous value of gene 1 and of gene 3 given the previous values of gene 2 were computed as shown in table 4 and 5 respectively.

		P(Gene 2| Gene 1)		
Value Gene 1	P(Gene 1=Value)	Low	normal	High
Low	34.67	96.05	2.77	1.18
Normal	32.19	42.47	16.62	40.91
High	33.14	0.97	3.8	95.23

Table 3: conditional probabilities (%) for the expression levels of gene 2 given the levels of gene 1.

		P(Gene 3| Gene 2)		
Value Gene 2	P(Gene 2=Value)	Low	normal	High
Low	47.3	97.4	2.24	0.36
Normal	7.58	40.9	13.98	45.12
High	45.12	0.42	2.02	97.56

Table 4: conditional probabilities (%) for the expression levels of gene 3 given the levels of gene 2.

These figures indicate the following:

- if the regulator has the value ‘low’ or ‘high’ then its child will have the same value in the next step in more than 95% of the cases. The conditional probabilities in both tables for these values look similar.

- if the regulator has the value ‘normal’, its child will have the value ‘low’ or ‘high’ in more than 80% of the cases, and these individual values are almost equally probable. The value ‘normal’ is not as predictive for the values of the regulated gene as the values ‘low’ or ‘high’.

Gene 1 has the value ‘normal’ in approximately one third of the cases, whereas gene 2 has this value in only 7 to 8% of the cases in favour of the more predictive values ‘low’ and ‘high’. This means that a larger portion of the cases in the dataset is predictive for the connection from gene 2 to gene 3 than for the link from gene 1 to gene 2. This is why the DBN inference algorithm needs fewer samples to recover the link between gene 2 and gene 3 than the link between gene 1 and gene 2. Or more generally, in the single parent networks used within GeneSim, a link between a gene G and its child C can be recovered using fewer samples when G is regulated by another gene than when it is not.

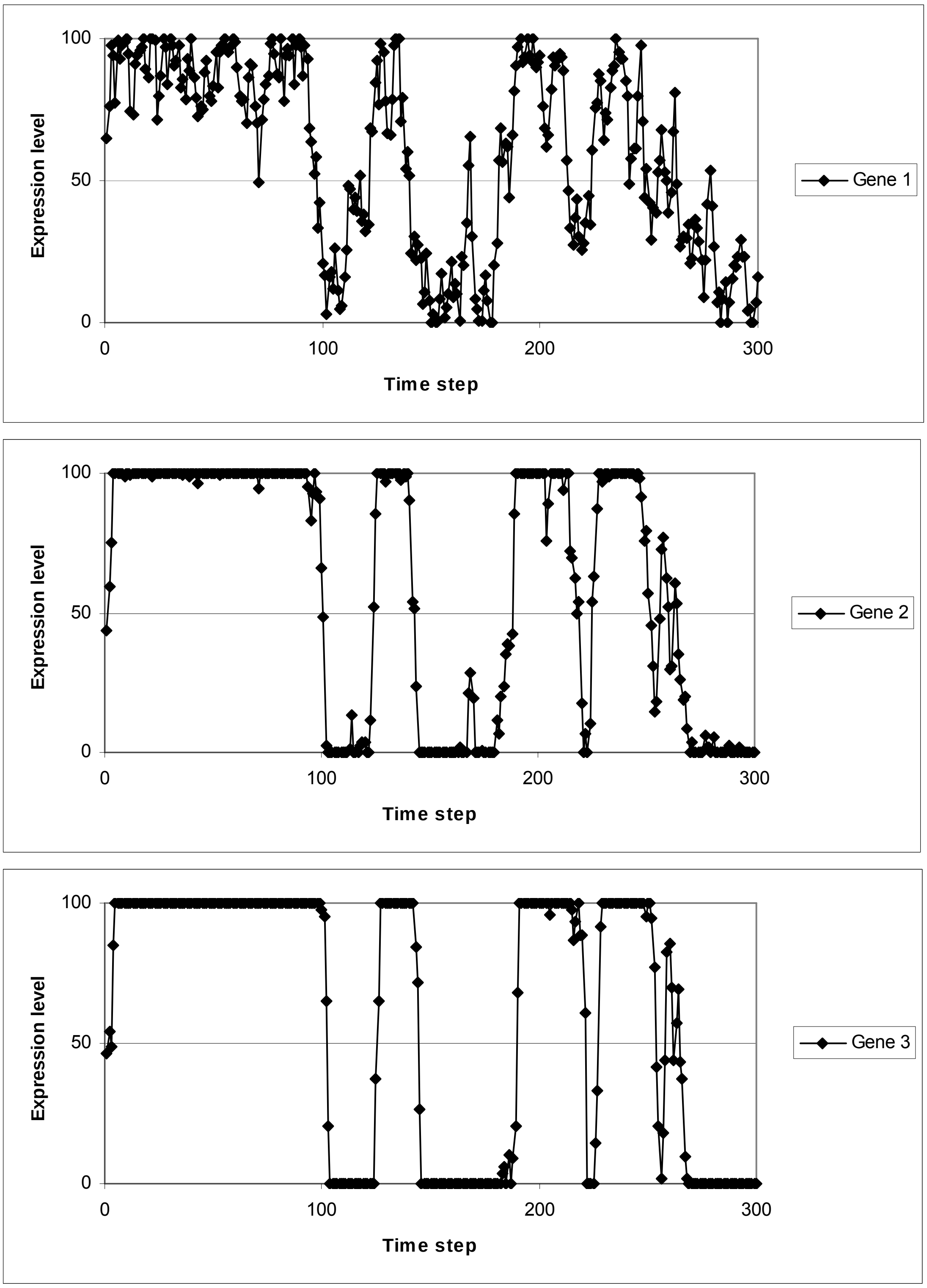


Figure 7: expression levels for genes 1, 2 and 3 in network 3 for 300 consecutive time steps. Gene 1 is not regulated and moves according to the noise level in GeneSim. Gene 2 is regulated by gene 1. Gene 3 is regulated by Gene 2. It can be seen that the expression levels of regulated genes (genes 2 and 3) are more often at the extremes (0 and 100) than the unregulated gene 1.

2.7.2 Learning multi parent networks

In the previous experiments, networks with different number of connections were constructed, thereby making sure that no gene had more than one regulator. In contrast, in this paragraph networks containing genes with multiple regulators are investigated. In particular, different networks containing a gene with 1, 2, 3 and 4 parents are constructed systematically, as shown below.

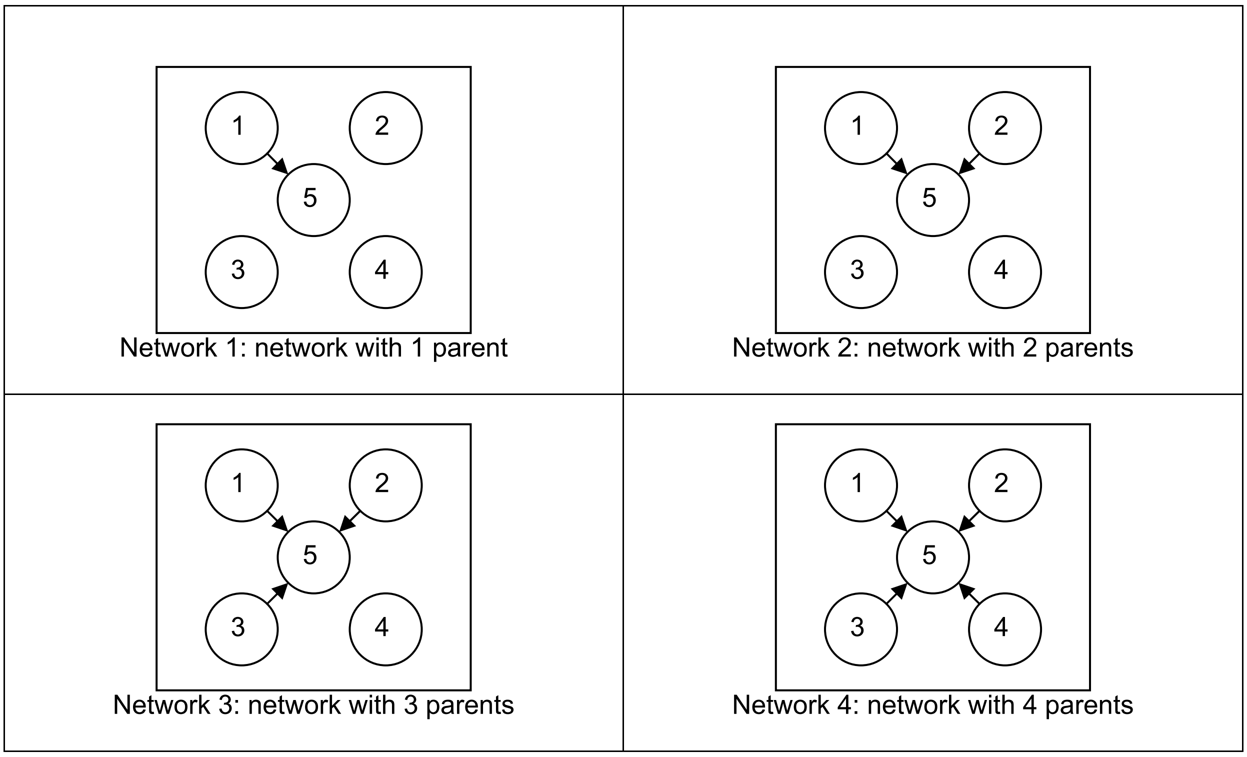

Network 1: network with 1 parent
Network 2: network with 2 parents
Network 3: network with 3 parents
Network 4: network with 4 parents

Figure 8: networks used in the experiments to assess the influence of the number of regulators on the required sample size for good structure recovery. All networks consist of 5 genes. Connections between genes all have strength of 0.2

Since the largest number of parents investigated is 4, the minimal number of genes for the largest network investigated required is 5. To keep the number of possible networks constant and measure the influence of the different number of regulators only, all networks are constructed using 5 genes. Note that network 1 has the same structure as network 1 in the previous paragraph and that only the total number of genes in the network differs. For these experiments, more and larger datasets are used than in the previous paragraph. Datasets of 12 different sizes ranging from 50 to 10000 in the number of samples are used. Except for this difference, the experiments are conducted using the same settings as in the previous paragraph.

Table 5 shows the average sensitivity and specificity scores on the networks with genes with different number of parents. To make the comparison easier, sensitivity scores are also presented in graphical form in Figure 9. As the specificity scores are fairly constant no graph is provided for them.

As in the previous section, these results show that the quantity of data is an important factor in the recovery of the structures. As the sample size grows larger, network structures are recovered more completely and networks with 1, 2 and 3 parents are recovered entirely (also note the similar results for the 1 parent network and network 1 in the previous section). However, the graph indicates that the performance of the DBN first deteriorates with growing sample size before rising to higher levels. This unexpected behaviour was not observed with the experiments in the previous section and unfortunately, no explanation was found for this. It cannot be explained away as merely coincidental, since the shape is observed for all the networks with multiple parents.

	1 parent		2 parents		3 parents		4 parents	
Samples	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.
50	50.00	97.50	25.00	98.26	16.67	97.27	17.50	96.19
100	50.00	100.00	20.00	99.57	6.67	99.09	10.00	100.00
250	100.00	100.00	25.00	100.00	6.67	100.00	0.00	100.00
500	100.00	100.00	20.00	100.00	3.33	100.00	0.00	100.00
750	100.00	100.00	10.00	100.00	0.00	100.00	0.00	100.00
1000	100.00	100.00	80.00	100.00	0.00	100.00	0.00	100.00
1500	100.00	100.00	100.00	100.00	0.00	100.00	0.00	100.00
2000	100.00	100.00	100.00	100.00	13.33	100.00	0.00	100.00
4000	100.00	97.50	100.00	100.00	40.00	100.00	7.50	100.00
5000	100.00	95.83	100.00	100.00	63.33	100.00	20.00	100.00
7500	100.00	95.83	100.00	100.00	100.00	100.00	25.00	100.00
10000	100.00	95.83	100.00	99.13	100.00	99.55	25.00	100.00

Table 5: average sensitivity and specificity scores obtained by the DBN inference algorithm on the multi-parent networks (averages calculated on 10 datasets for each sample size and 5 runs of the DBN algorithm per dataset). A sensitivity of 100 means that all the links in the original network have been recovered. A specificity of 100 means that no spurious links were recovered. When both are 100, the original and recovered networks are identical. The number of samples required for perfect score increases rapidly with the number of parents.

The effect of adding a connection - in this case adding a regulator to a gene - on the required sample size for perfect recovery is considerable: for 1 parent, at most 250 samples are required, for 2 parents at least 1000 samples and for 3 parents the required sample size is at least 5000. With the used sample sizes and settings, the DBN inference algorithm was not able to recover the entire 4 parents network (with 10000 samples, only 1 link was recovered with each of the 10 datasets). The effect of adding a connection such that the maximum number of parents is increased clearly has a much larger impact than adding one or more single parent nodes as in the experiments of 2.7.1.

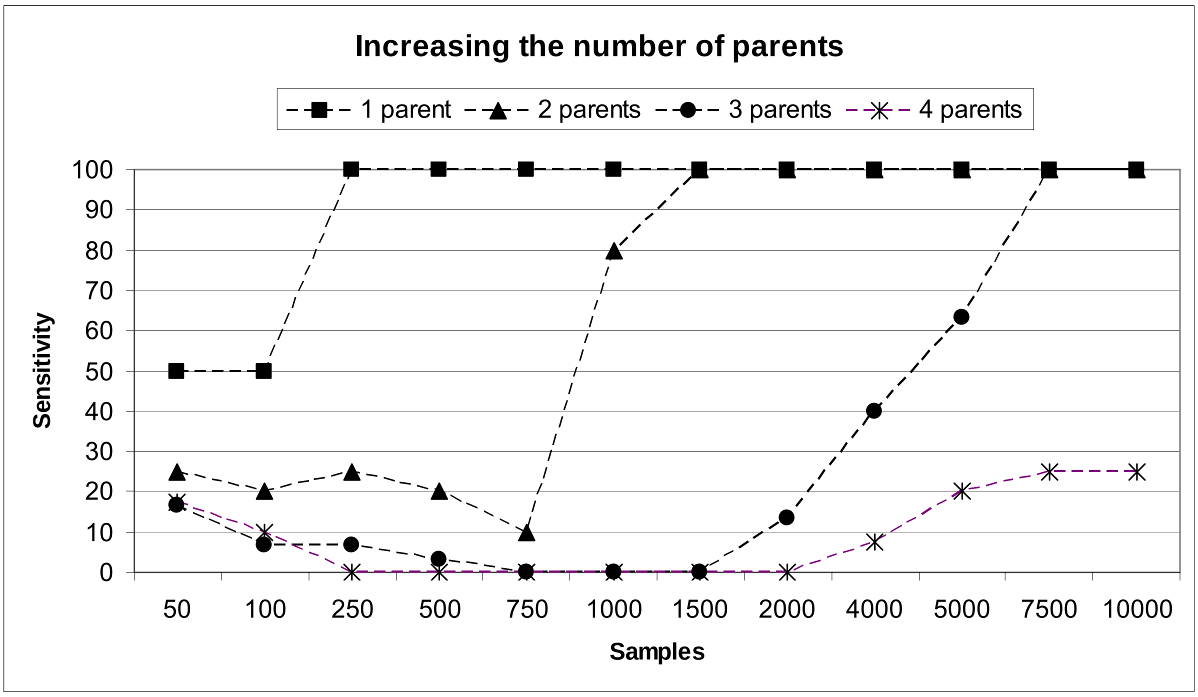


Figure 9: sensitivity graph for the networks containing multi-parent node. The graph shows the effect of sample size on sensitivity for networks with different number of parents. Note that the scale is not linear, masking the effect of increasing the number of regulators.

Earlier, it was shown that connections to genes with a grandparent are easier to learn than genes without grandparent. This is the case with the networks used in this section as well. To demonstrate this, networks 1 through 4 are simulated again in GeneSim in the following way: a parent is added to each regulator such that, in each network, gene 5 has as many grandparents as parents (stated otherwise, each parent of gene 5 now has a parent of its own). New datasets are sampled as before using the GeneSim simulator, but the expression levels of the new grandparents are omitted. This results in datasets containing expression levels for gene 1 to 5 as before with the difference that the parents to gene 5 behave *as if* they were regulated by another gene (see Figure 10 for an illustration for network 2). As far as the DBN inference algorithm is concerned, it is processing a dataset consisting of 5 variables like before. In the following, these networks will be referred to as the multi parents network 'within a context' to distinguish them from networks 1 to 4.

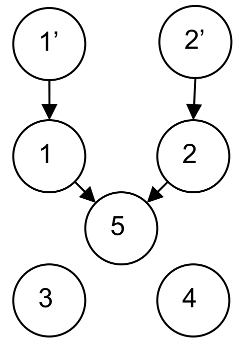


Figure 10: the networks are simulated again in GeneSim after a parent is added to each regulator in the network. For example, in network 2 shown on the left, genes 1 and 2 are the regulators. Two parent genes 1' and 2' respectively are added to each of these. New datasets are sampled from this network, but the expression levels for the 'new' genes 1' and 2' are omitted when passed to the DBN inference algorithm. The algorithm is passed datasets consisting of 5 variables with the regulators behaving as if they in turn were regulated.

Table 6 shows the average sensitivity and specificity scores on the networks with increasing number of parents within a context. Sensitivity scores are presented in a graph in Figure 11.

Required sample sizes for good recovery are sharply lower when compared to the results shown in Figure 9. For instance, only 1500 samples are needed to consistently recover the 3 parents network versus 7500 samples earlier. The 4 parents network is consistently recovered using 7500 – 10000 samples, whereas earlier only 1 link was recovered with the largest datasets.

Although considerably fewer samples are needed, it can still be observed that the effect of adding a connection such that the maximum number of parents is increased clearly has a very strong impact.

	1 parent		2 parents		3 parents		4 parents	
Samples	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.	Sens.	Spec.
50	100.00	97.50	20.00	94.35	30.00	91.82	25.00	93.33
100	100.00	98.33	20.00	100.00	43.33	99.09	15.00	98.10
250	100.00	100.00	10.00	100.00	36.67	100.00	5.00	100.00
500	100.00	100.00	60.00	100.00	40.00	100.00	5.00	100.00
750	100.00	100.00	100.00	100.00	80.00	100.00	0.00	100.00
1000	100.00	100.00	100.00	100.00	80.00	100.00	0.00	100.00
1500	100.00	100.00	100.00	100.00	100.00	100.00	0.00	100.00
2000	100.00	100.00	100.00	100.00	100.00	100.00	0.00	100.00
4000	100.00	100.00	100.00	100.00	100.00	100.00	10.00	100.00
5000	100.00	100.00	100.00	100.00	100.00	100.00	35.00	100.00
7500	100.00	99.17	100.00	100.00	100.00	100.00	92.50	100.00
10000	100.00	97.08	100.00	100.00	100.00	100.00	100.00	100.00

Table 6: average sensitivity and specificity scores obtained by the DBN inference algorithm on the multi-parent networks within a context (averages calculated on 10 datasets for each sample size and 5 runs of the DBN algorithm per dataset). A sensitivity of 100 means that all the links in the original network have been recovered. A specificity of 100 means that no spurious links were recovered. When both are (almost) 100, the original and recovered networks are (almost) identical.

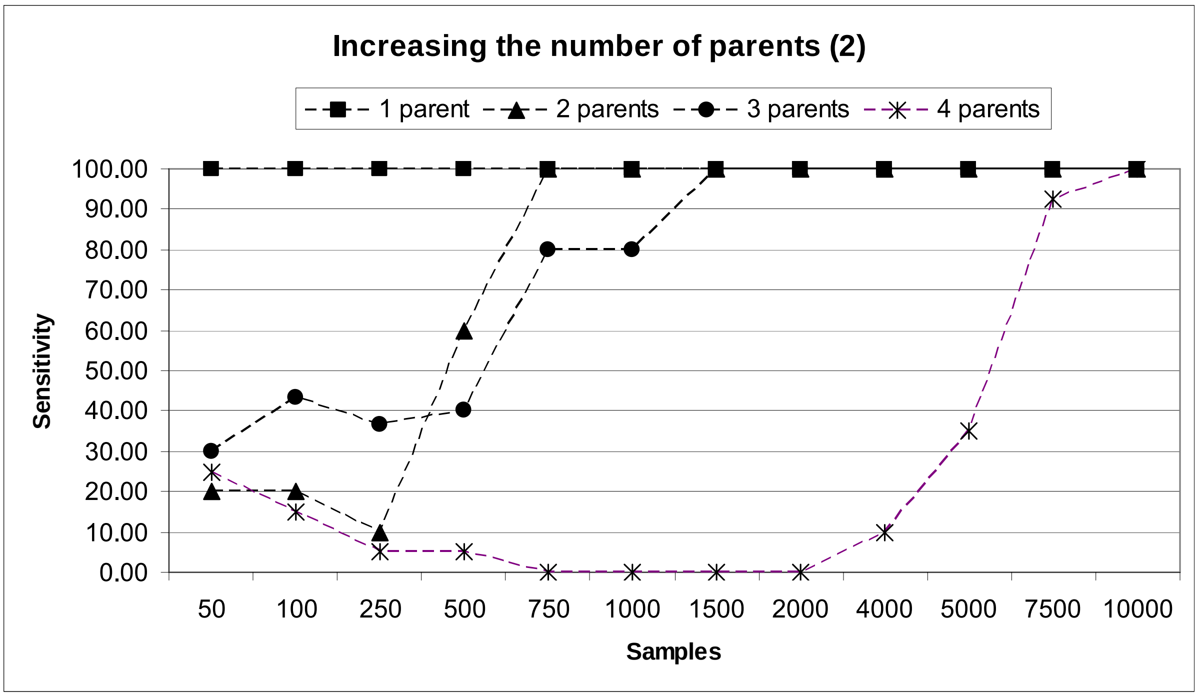


Figure 11: sensitivity graph for the networks containing a multi-parent node within a larger network (within a context). The graph shows the effect of sample size on sensitivity for networks with different number of parents. Note that the scale is not linear, masking the effect of increasing the number of regulators.

2.7.3 Choice of search strategy & settings

The choice for the search strategy used in this research is based on the work of Yu et al (2002, 2004). In these papers, 3 different search strategies were compared: greedy search with random restarts (considering all local changes at each iteration), simulated annealing and an evolutionary algorithm. Both the greedy and simulated annealing algorithm frameworks are described by Heckerman (1996) and are implemented in Banjo. The evolutionary algorithm was designed by Yu et. al for their research and is not offered in Banjo. In both papers, Yu et al report that the 3 strategies return (nearly) identical networks but greedy search was favoured owing to the considerable shorter running times (greedy search was reported to be at least 10 times faster than simulated annealing). Another argument in favour of greedy search is given by Chickering et al (1996) who have shown that for a fixed amount of computation time, greedy search with multiple restarts outperforms simulated annealing (amongst others). For these reasons, greedy search with random restarts was chosen for the experiments in this research.

As already discussed in the methods section, the exact behaviour of the greedy search algorithm with random restarts in Banjo is configured with the following parameters in the settings file:

- “maxProposedNetworks”, the total number of networks the algorithm is allowed to evaluate during the search before exiting and returning the results of the search.
- “minProposedNetworksAfterHighScore”, the minimum number of search iterations that Banjo will execute after it has found a new high score, before a restart is initiated.
- “minProposedNetworksBeforeRestart”, the minimum number of search iterations that Banjo will execute after a restart, before the next restart is initiated.
- “maxProposedNetworksBeforeRestart”, the maximum number of search iterations that Banjo will execute after a restart, before the next restart is initiated.

The minimum and maximum number of restarts can then be estimated from the above settings as follows:

- minimum number of restarts is “maxProposedNetworks” divided by “maxProposedNetworksBeforeRestart”
- maximum number of restarts: “maxProposedNetworks” divided by “minProposedNetworksBeforeRestart”

In order to find values for these settings representing a reasonable trade-off between quality of returned solutions and running time, Banjo was executed using three different configurations using one of the single parent networks (network 4 of section 2.7.1) and one of the multi parent networks (network 3 of section 2.7.2). The configurations considered are shown in Table 7 below.

Setting	Configuration 1	Configuration 2	Configuration 3
maxProposedNetworks	100000	1000000	1000000
minProposedNetworksAfterHighScore	1000	1000	1000
minProposedNetworksBeforeRestart	3000	3000	6000
maxProposedNetworksBeforeRestart	5000	5000	10000
Restarts (derived, not banjo’s setting)	20 to 33	200 to 330	100 to 166

Table 7: parameters considered for the greedy search using random restarts and all local changes at each iteration.

For both networks, datasets of 4 different sizes were sampled using GeneSim. For each given sample size, 10 independent datasets were generated. To average out the stochastic effects of random restarts, Banjo was executed 5 times on each of the datasets. Results of these experiments are presented in Table 8 and Table 9.

For the single parent network, only small differences are observed between configurations 1 and the others. Configurations 2 and 3 yield the same results. For the multi parent network, no difference is found at all between any of the configurations. Perfect scores on sensitivity and specificity are obtained with the same amount of data independent of the chosen configuration. Hardly any variance is observed across runs in the results obtained: the only variance found is in the experiments with the single parent network and datasets of 100 and 250 samples. Since only small differences in results exist across configurations, it is concluded that applying Banjo 5 times on each dataset is sufficient.

	Configuration 1		Configuration 2		Configuration 3	
Samples	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
100	70.86	93.05	71.43	92.80	71.43	92.80
250	95.14	97.83	94.29	97.63	94.29	97.63
500	100.00	100.00	100.00	100.00	100.00	100.00
750	100.00	100.00	100.00	100.00	100.00	100.00

Table 8: results for network 4 of 2.7.1 (consisting of 7 single parent links) using the different configurations of the greedy search.

	Configuration 1		Configuration 2		Configuration 3	
Samples	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
2000	13.33	100.00	13.33	100.00	13.33	100.00
4000	40.00	100.00	40.00	100.00	40.00	100.00
5000	63.33	100.00	63.33	100.00	63.33	100.00
7500	100.00	100.00	100.00	100.00	100.00	100.00

Table 9: results for the 3 parent network (network 3 of 2.7.2) using the different configurations of the greedy search.

To check if the ‘true’ networks could be found more frequently by evaluating more networks, the Banjo BDe scores of the true networks were computed for each dataset and compared to the scores obtained by Banjo during each of the runs. Every network returned by Banjo was found to be at least as good as the true network. This means that every time Banjo did not return the true network, this was justified by the BDe score of the recovered network and that changing the parameters of the search would not have resulted in finding the true network more often.

Given the observations above, and the fact that execution times are factors shorter using configuration 1, this configuration was favoured to conduct the experiments in this research.

2.8 Discussion & conclusion

With the availability of large-scale gene expression data, effort has been made towards learning gene networks using Bayesian networks and other techniques. In this research, the suitability of Bayesian networks for this task was examined. Using a known biological simulator, regulatory networks with different topologies were simulated and values for gene expression levels were sampled from simulations at discrete time steps, like during real microarray experiments. The obtained data was fed to a standard Bayesian network inference algorithm and sample sizes required for good recovery was estimated. In particular, the required amount of data for the recovery of networks containing only single parents was contrasted with the amount required for networks containing a gene with multiple parents.

Results show that the quantity of data is a critical factor in the recovery of the structures. Although unexpected results were obtained for networks with multiple parents using smaller datasets, performance clearly increased with sample size. It was observed that required sample size for good recovery is strongly influenced by the number of regulators to a gene. Although the single parents networks used in 2.7.1 contained a different number of connections - ranging from 1 to 9 connections - the difference in the required sample size for good recovery seems insignificant when compared to the multiple parents experiments of 2.7.2 which contained a maximum of only 4 connections. This strong effect of the number of regulators on required amount of data is also reported in the literature. For instance, Berlo et al (2003) claim in their analysis that an exponential increase in the number of samples is necessary to obtain a reliable hypothesis when the number of parents increases. Yu et al, (2002, 2004) report difficulties identifying more than one parent for genes with multiple parents (using up to 5000 samples) and regard this as “a serious problem for genetic pathway recovery, as combinatorial regulatory control is a basic property of genetic pathways”. As microarray datasets tend to be small in practice, these results are not encouraging.

On a more positive side, it has been noted by Philip et al (2004) that the inference algorithm does not have to recover the complete structure in order to be useful; learning only a part of the network with reasonable certainty would be an accomplishment from the biologist’s point of view. Another promising way to cope with the small amount of current datasets is combining prior biological knowledge with the gene expression data. There are many potential sources of such prior knowledge about edge relationships such as the scientific literature, or experimental data, as Philip et al (2004) mention. Hartemink et al. (2002), Imoto et al. (2003), Husmeier (2003), and more recently Bernard et al (2005) have used prior knowledge from different sources along with gene expression data. In a synthetic study, Phillip et al. (2004) attempt to quantify the effect of using prior knowledge on required sample size, and conclude that “incorporating prior knowledge into the learning scheme greatly reduces the data required, allowing these reverse engineering techniques to be used to learn regulatory interactions from microarray data sets of realistic size”.

It was found that specificity scores initially increase with sample size, but decrease when large datasets are used. This was also observed by Yu et al (2004) who attributed this to the fact that the original networks are not Bayesian networks. Because of the mismatch between the dynamic Bayesian network and GeneSim, supplying more data results in over-fitting the network to the data, leading to the inclusion of false positives. Experiments (not included in this paper) conducted with data sampled from ‘real’ Bayesian networks and datasets of up to 10000 samples seem to confirm this. Whether the algorithm would include false positives when larger datasets are used remains an open question.

As stated in the introduction, the figures presented in this section are meant to be comparative and should not be taken as absolute for the number of samples required to recover network structures in general. A large number of factors not examined in the experiments could influence structure recovery, for example the exact configuration of the DBN inference algorithm, strength of the connections in the network, the sampling interval used to generate the datasets, the number of discretization categories and the way networks are simulated. It was beyond the scope of this study to investigate these factors, but each is reviewed briefly in the remainder.

The experiments in this research were carried out using greedy search with random restarts for reasons already detailed in 2.7.3. However, it is not clear that greedy search would yield equally good results as simulated annealing or genetic algorithm in all cases. In general, greedy search algorithms are more likely to get stuck in local maxima than simulated annealing and genetic algorithms. Another aspect of the DBN inference algorithm that could potentially impact results is the scoring function. Currently, the BDe metric is the only one implemented in Banjo, but other Bayesian metrics exist. For instance, Yu et. al (2002, 2004) applied the Bayesian Information Criterion (BIC) and concluded it did not work as well as the BDe metric with small datasets because BIC applies a strong penalty for the complexity of a structure (yielding networks with fewer edges or no edges at all). Which search strategy and metric are the best suited for the recovery of networks from expression data remains an open question.

In all networks used in the experiments above, the strength of the connections between genes was set to 0.2. This corresponds to the highest value used in Yu et. al (2002, 2004) and was kept constant in this research in order to investigate the effect of the increased number of connections only. Early experiments on 2 parent networks conducted at the start of this research showed that (without changing other parameters) more samples were required to recover networks with weaker connections. Also, if the connections in the 2 parent network are of different strength (for instance 0.2 and 0.1), the algorithm finds the stronger connection more frequently than the weaker one. The exact amount of data required for complete recovery of the generating structure will depend on the strength of the connections in the network and perhaps it will not be practical to recover connections with small strength (close to zero).

Another critical factor in the recovery of structure is the sampling rate. The true continuous signals are sampled at discrete time points, yielding a loss in information, especially if the sampling intervals are not matched to the relaxation times of the true biological processes (Husmeier (2003)). In Yu et. al, (2004) it was found that the best results were obtained applying a sampling interval of 5 (the sampling interval used in the present research is based on this result, although early experiments indicated that intervals of 3 or 4 yielded comparable results). But they also note that the optimal interval varies with the number of data points measured and that it is presumably determined by the internal dynamics of the system at hand. Smith et al (2003) suggest a strategy for finding the optimal sampling regime for a biological system by slowly increasing the interval. According to Yu et. al (2004), educated guesses about system’s dynamics have to be made in order to infer the most accurate network.

After the data is sampled, it has to be discretized before it can be presented to the network inference algorithm. When discretizing data, one has to decide on both a discretization method and a number of categories or bins (i.e. the number of discrete values). As discussed in the methods section, Banjo offers two discretization methods: equal range and equal frequency (or quantile) discretization. Based on the comparison given by Yu et al (2004) and on experiments conducted at the start of this research, discretization into 3 bins of equal range was used for the experiments (when 4 categories were used, more data was required for good recovery. Using 2 categories led to the inclusion of many false positives). However, no silver bullet can be given for discretization, and researchers use the method they somehow find appropriate. For instance, Berlo et al (2003) use a method known as K-means clustering to discretize their data into three categories. Friedman et. al (2000) discretize values into 3 categories depending on whether the expression rate is significantly lower than, similar to, or greater than a control value, respectively (wherein the control expression level of a gene can be the average expression level of the gene across experiments but can be also be determined experimentally in other ways). When discretizing his data, the researcher somehow needs to finds a method and a number of categories such that the discretization is neither too coarse, and associated loss in information too high, nor too fine, leading to more parameters than perhaps necessary.

Finally, results can be influenced by details of the simulation used to generate the data. The simulator used here obviously models regulatory networks in a heuristic way (for instance, it does not model transcription and translation steps in protein production). However, as stated earlier, the simulator does not need to be an exact match to, or even model all features of, a real regulatory network, as long as many of the important biological features are modelled. The simulator is useful as long as the qualitative phenomena present in biological systems are exhibited. In this research, the underlying networks were not Bayesian networks and, as noted earlier, this was presumably the reason that the learning algorithm included false positives when using large datasets.

Part 2 Filling gaps in metabolic networks

Genetic networks present many opportunities for machine learning techniques. Being of the most fundamental processes in the cell, metabolic processes are involved in virtually all other processes. As such, their reconstruction is crucial for understanding metabolic diseases. In part 1, at one extreme of network reconstruction, Bayesian networks were applied to infer genetic networks from microarray data without using any biological prior knowledge about the structure of the network. It was shown using artificial data experiments that genetic networks with various topologies could be recovered successfully given sufficient data, but also that sample size requirements for good performance increase dramatically with the number of regulators per gene. As the sample size of microarray datasets tend to be small in practice, these results were not particularly encouraging.

In this part, the focus is on the other extreme of network reconstruction: the completion of almost fully known networks. As more and more information about genomes is collected, biologists are able to reconstruct large parts of gene networks. In many cases however, these partially known networks contain gaps, meaning that evidence exists that a gene should be present at a particular location in a network, but it is unknown which gene should fill the function. The identification of the genes in a well characterized or nearly complete network has been referred to as the “missing genes” problem (Osterman et al, 2003), or as “filling gaps in metabolic pathways” (Kharchenko et al, 2004). In the following, the suitability of Bayesian networks for the missing genes problem is investigated. An automated method for completing a partially known metabolic network using microarray data only is described. Given a (single gene) gap G in a supplied network, and a set of candidate genes for the gap, the placement algorithm ranks the list of candidate genes: the first gene in the list being the most probable candidate for the gap represented by G, and the last gene being the least probable candidate. The ordering is determined by a Bayesian metric which assigns a score to each candidate indicating how likely the data was generated by the network containing the candidate.

The approach described is based on the work presented by Kharchenko et al (2004), who propose a concrete instance of the missing genes problem involving *Saccharomyces Cerevisiae*, more commonly known as Baker’s yeast. Their approach however does not rely on Bayesian networks, but on a metabolic network with undirected links, while candidates are sorted according to a similarity metric. By the time of writing, we have not been able to find a similar application of Bayesian networks, so the approach proposed here is probably a novel one. The viability of the approach is demonstrated using a synthetic network. Gaps are created in this network, and the number of samples required to fill the gaps is investigated. Results show that single gaps can be filled using considerably less data than when learning networks from scratch, which of course is an encouraging result. Then, following Kharchenko et al (2004), the performance of the gene placement algorithm is assessed on yeast data. The Bayesian method does reasonably well, but it does not outperform Kharchenko et al (2004).

The remainder is structured as follows: first, the approach and methods used in this study are described in detail. Then, the viability of the proposed Bayesian gene placement method is demonstrated using a relatively small synthetic problem. The impact of sample size on performance is investigated. After this, experiments with the yeast data are described. Impact of discretization on performance is investigated and two different methods for filling gaps are evaluated. One method relies on the structure provided in the supplied metabolic network whereas the other assumes conditional independence between genes given a candidate. Finally, results of this project are briefly summarized and discussed.

2.9 Approach & Methods

The gene placement algorithm described here is designed to evaluate candidate genes for filling gaps in a known genetic network. Given a gap in a known network, a set of candidate genes for the gap, and gene expression information, the gene placement algorithm sorts the list of candidates according to how well they fit the gap according to a Bayesian criterion. Stated like this, the evaluation of candidates for a given gap seems straightforward: simply fill a given gap with each candidate in turn in the network as a whole, and evaluate each using the data and the Bayesian scoring algorithm.

However, biological networks can contain hundreds or thousands of genes (for instance, the metabolic yeast network used in this research contains around 800 genes, while the total number of yeast genes considered in this study is 6207). It is not practical to fill gaps and evaluate candidates in networks taken as a whole. Fortunately, it turns out that it seems reasonable to evaluate candidates using only a small subnetwork consisting of genes somehow close to the gap (so called “Markov Blanket”) instead of the whole network. One important limitation in the evaluation of candidates however, is that given the software package used in this research *as is*, only networks consisting of genes with no more than 10 parents can be evaluated. Networks containing more parents caused memory related errors. It turns out that with this limitation, the approach can be tested using around a third of the gaps in the yeast network than would be the case ideally. It is assumed here that this should provide enough experiments to give a good indication of the performance of the proposed method, and it was decided to accept this limitation without altering the software.

Further, using Bayesian networks imposes a number of constraints on both the data and the structure of the networks used. As the Bayesian algorithm used can handle only discrete values, a suitable discretization scheme has to be selected. In addition, the subnetwork used to evaluate candidates should be both directed and acyclic. Unfortunately, the network supplied for yeast in this research is not acyclic, as it contains many connections going back and forth between pairs of genes. Some method is required to eliminate cycles from this network before the Bayesian gene placement method can be applied on this problem.

Summarizing the above, the Bayesian gene placement algorithm proposed here can be globally described as follows. Given a partially known network, a gap, a list of candidates for the gap, and gene expression data:

1. (If necessary), discretize the data.
2. Determine the subnetwork around the gap that will be used to evaluate candidate genes.
3. For each candidate in the list:
    a. Fill the gap in the subnetwork with the candidate.
    b. Prepare a dataset containing the expression information for the genes in the subnetwork.
    c. Compute the Bayesian score of the subnetwork given the dataset.
4. Order the list of candidates on the Bayesian score, the first in the list being the most probable candidate and the last one in the list being the least probable.

The methods used in this research are described in further detail in the remainder. First, the rationale for using a subnetwork around gaps is given and two different ways to build the subnetwork for gaps and deal with cycles are detailed. Then the metric used to evaluate the performance of the placement methods, the self-rank, is presented. Finally, the supplied yeast network and microarray data are described.

2.9.1 Determining the subnetwork

As stated before, it is not practical to fill gaps and evaluate candidates in networks taken as a whole. However, it turns out that it seems reasonable to evaluate candidates using only a small subnetwork consisting of genes close to the gap (the so called “Markov Blanket”) instead of the whole network. A procedure for the evaluation of candidates for directed acyclic graphs is illustrated in section 2.9.1.1.

If the network supplied is cyclic – as is the case with the yeast network used in this research – some reasonable means to eliminate cycles is required before the Bayesian gene placement method can be applied. One way to solve cycles would be to eliminate them in the supplied network taken as a whole and then apply the method for DAGs just mentioned. On the other hand, since the method described involves evaluating only the Markov Blanket of a given gap, it makes sense to attempt to eliminate cycles within such a subnetwork only instead of the complete network. An important advantage of this approach, as will become apparent, is that the removal of cycles can be tailored to the gap under investigation, which would not be possible in the complete network. Mainly for this reason, it was decided to remove cycles in network neighbourhoods rather than in the network as a whole. This procedure, which is an extension of the one presented for DAGs, is detailed in 2.9.1.2

With the methods just mentioned for determining the subnetwork for a gap, care is taken to preserve the structural information given in the supplied metabolic network (while deleting cycles if necessary). For interest, a placement method largely ignoring this structural information supplied is also investigated. Genes are placed assuming conditional independence between genes given a candidate, i.e. the candidate is made parent to all genes in the subnetwork ignoring known connections. This approach is described in 2.9.1.3.

2.9.1.1 Markov Blanket

A Bayesian network can have a large number of nodes and the behaviour of a given node can be affected by a distant node. However, it can be shown that knowledge about a set of nodes ‘close’ to the given node is sufficient to determine the behaviour of that node. Such a set of nodes shielding the node from the rest of the network is called a *Markov Blanket*. Neapolitan (2003) defines a Markov Blanket M(X) of a node X to be “any set of variables such that X is conditionally independent of all the other variables given M(X)”. Further, Neapolitan (2003) shows that for any node X, the set of all parents of X, children of X and parents of children of X is a Markov Blanket of X (in the remainder, this particular Markov Blanket will be referred to as *the* Markov Blanket).

This means that in order to determine the behaviour of a given node, it is sufficient to have knowledge about the parent of the node, the children of the node and the parents of the children of the node. It is not necessary to determine the behaviour of all the nodes in the network. Given this, it seems a reasonable assumption to make that when evaluating candidate genes, a ranking of scores for Markov Blankets would be similar to a ranking of scores for complete networks. Using this assumption, candidate genes can be evaluated using much smaller networks.

This is illustrated in Figure 12. The subnetwork around the gap consists of the genes in the Markov Blanket of the gap and the set of all connections between these genes. If the network neighbourhood happens to contain other gaps besides the one being tested, these are simply removed, since they cannot contribute any information relevant to the gap being investigated. If the network from which the network neighbourhood is taken is a DAG, the network neighbourhood is also a DAG and can be evaluated using the Bayesian metric.

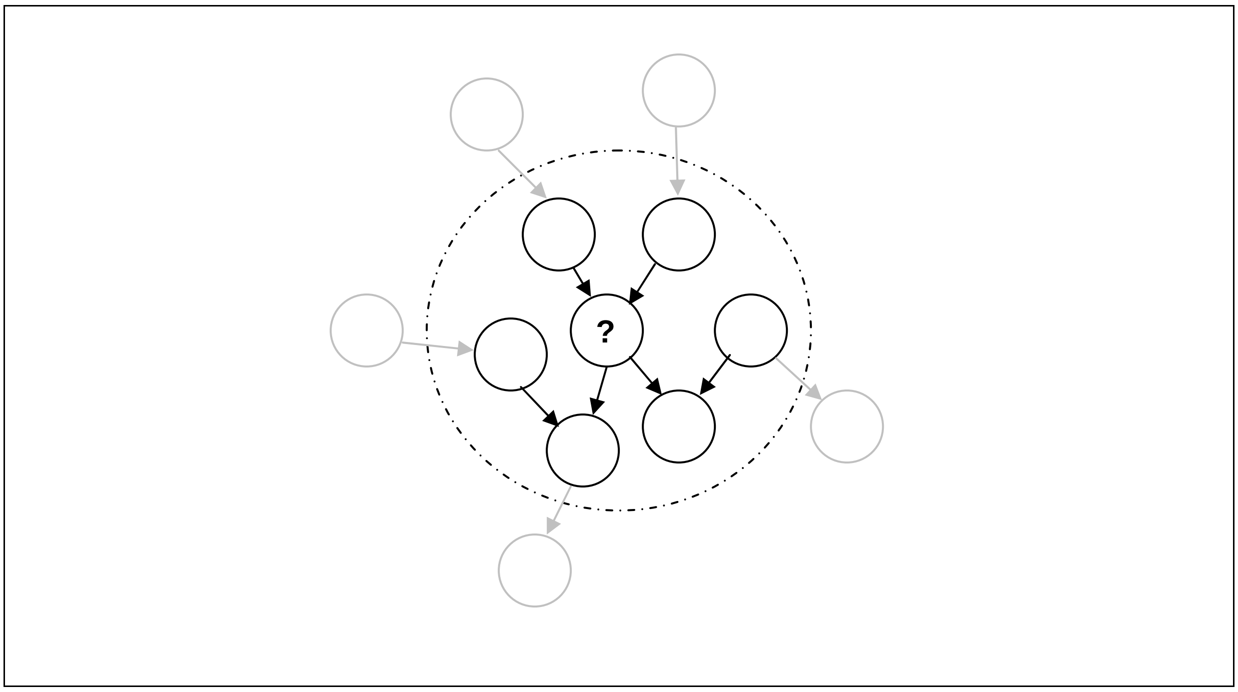


Figure 12: to evaluate candidates for a given gap, only the Markov Blanket of the gap (denoted by the question mark) is considered. The Markov Blanket of a node consists of the subnetwork containing the parents of the node, the children of the node, and the parents of the children of the node (shown within the dotted circle). The remainder of the network (light grey) is not used in the evaluation of candidates.

2.9.1.2 Dealing with cyclic networks

If the network supplied is cyclic – as is the case with the yeast network used in this research – some reasonable means to eliminate cycles is required before the Bayesian gene placement method can be applied. Such a method should remove cycles from the network, while somehow preserving as much of the information present in the supplied network as possible. Here, no elaborate procedure to eliminate cycles in graphs is described, but effort is made to eliminate cycles caused by connections going back and forth between two genes (“bidirectional” links or “local loops” in the remainder), as most of the cycles in the supplied yeast network are caused by this type of connections.

An obvious way to solve a cycle caused by a bidirectional link is to simply remove one of the links involved in it. Since there is in general *a priori* no reason to prefer one link over the other, the choice for the link to be removed is arbitrary. In some cases however, an argument could be made in favour of a certain link. Consider for instance a bidirectional link between a gene and the gap under investigation. To solve this loop, two possibilities exist: (1) deleting the link from the gene to the gap or (2) deleting the link from the gap to the gene. If the link from the gene to the gap is removed – the gene becomes a child of the gap – it is not possible to measure the effect of the gene on the candidate combined with the effect of other co-parents to the candidate. This means that potentially valuable information could be lost. Thus, it is felt that the link from the gap to the gene should be removed, making the gap a child of the gene. Similarly, when considering a bidirectional link between a gene and one of the gap’s children, the gene is made a parent to the child so the combined effect of that gene, the gap and other parents of the child can be measured on the children of the gap. In all other cases, local loops are solved by removing one of the links arbitrarily.

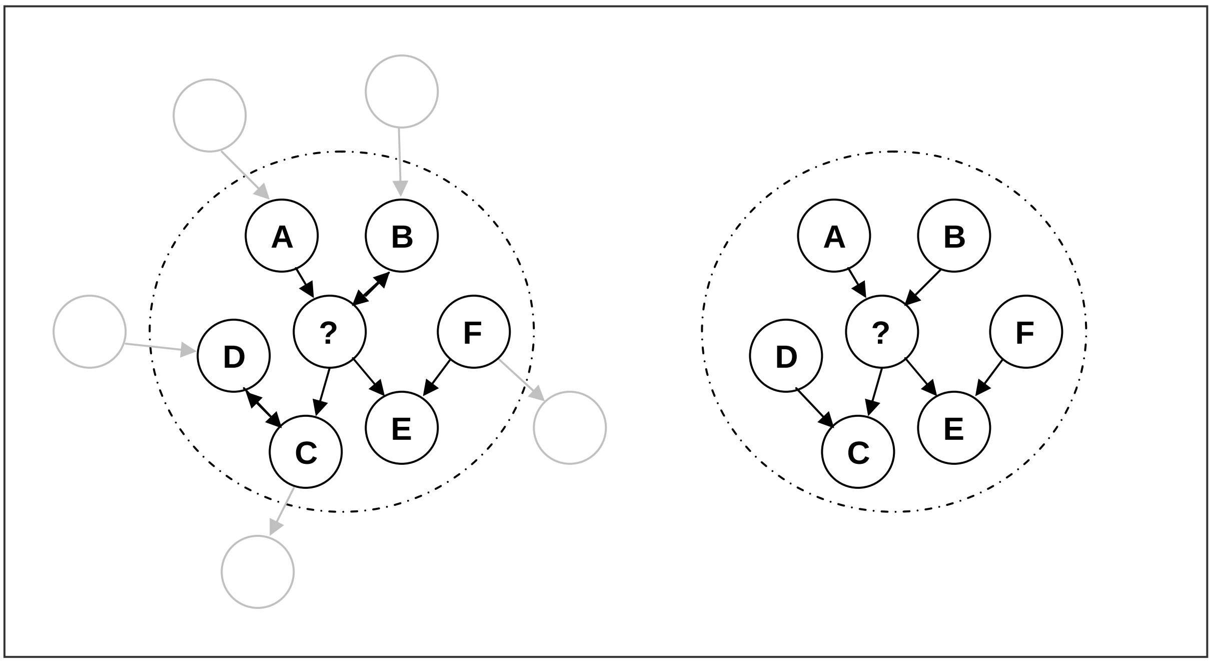


Figure 13: determination of the subnetwork for cyclic networks. On the left, a partially known metabolic network containing a gap is displayed. This is not a Bayesian network, since it contains cycles caused by the bidirectional links. Removing cycles with the procedure described results in the network shown on the right. This is the subnetwork that will be used to evaluate candidates. Gene B is kept as a parent to the gap (the link from the gap to B is removed), so the combined effect of A and B on the gap can be measured. Similarly, D is kept as a parent to C (the link from C to D is removed), so the combined effect of D with the gap on C can be measured.

Given the ideas above, the procedure described for DAGs to determine the network neighbourhood of a given gap G can then be modified into the following:

- The set of all parents of G, including all genes with a local loop with G.
- The set of children of G, excluding any gene with a local loop with G.
- The set of parents of children of G, including all genes with a bidirectional link with one of these children.

The network neighbourhood for the gap consists of the genes thus gathered and the set of connections between these genes. Then, any local loops in the neighbourhood are eliminated by keeping only one of the links as follows:

- A local loop between a gene and the gap, keep the link from the gene to the gap.
- Otherwise, a local loop between a gene that is not a child of the gap and a gene that is: keep the link from the gene to the child of the gap.
- In all other cases, retain one of the links arbitrarily (although making sure not to create a local loop)

This is illustrated in Figure 13. As before, if gaps other than the gap investigated are encountered in the neighbourhood, they are simply removed from it. The procedure described above produces a network without bidirectional links between genes, but does not guarantee that the graph is acyclic. The procedure results in a sufficient amount of gaps with an acyclic network for the purposes of this research and no further effort is made to solve these cycles. Thus, any cyclic networks that remain after removal of local loops are excluded from the experiments.

2.9.1.3 Naive Bayes

With the previous methods for determining the subnetworks, care was taken to preserve the structural information contained in the supplied network. Here, a rather different approach is followed. The genes involved in the subnetwork are determined as described earlier for cyclic networks, but the structural information present in the supplied network is largely ignored and the gap is simply made parent to all the genes in the subnetwork. This in fact represents the assumption that all genes in the neighbourhood of a gap are conditionally independent of each other given the candidate (the assumption of conditional independence is used in a supervised learning context in so called “naive Bayes classifiers”; see [Wiki – Naive Bayes] for an informal introduction).

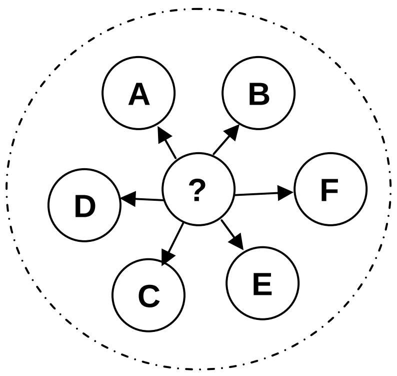


Figure 14: “naive Bayes” approach for the evaluation of candidate genes for gaps for the network in Figure 13. The same genes are involved in the subnetwork as before, but the structural information in the known metabolic network is ignored and the gap is made parent to all the genes in the subnetwork. This corresponds to the assumption that all genes in the subnetwork are conditionally independent of each other given the candidate gene. The resulting subnetwork is always acyclic and each gene (except the gap) has exactly 1 parent, making the approach less demanding in resources than the previous approach.

The potential loss of information is obvious, but it is interesting to compare the performance of this method to the more “conservative” approaches described earlier. However, the approach is implemented for comparison purposes only. It is not the aim here to somehow determine the “best” naive subnetwork for a given gap; the same genes are used as before, while ignoring known connections.

This simple approach has the following advantages: first, the resulting subnetwork is always acyclic, as elimination of cycles is implicit. Second, the maximum number of parents in the neighbourhood is always 1, making the scheme less demanding in resources and requiring a relatively simple implementation (no complex network structure is needed). Using this strategy, it is in theory possible to tackle all the gaps in the supplied network given the implementation of Bayesian networks used in this research.

2.9.2 Bayesian score

The gene placement algorithm evaluates candidate genes for a given gap and sorts them according to a given criterion. Given a gap G in the supplied network, and a set of candidate genes, the gene placement algorithm ranks the list of candidate genes: the first gene in the list being the most likely candidate for the gap represented by G, and the last gene being the least likely candidate. In the Bayesian approach proposed here, the ordering is determined by a Bayesian metric which assigns a score to each candidate indicating how likely it is that the data was generated by the network neighbourhood.

The Bayesian metric used in this research is the BDe metric, as implemented in Banjo, the software package used in this research. Given the (acyclic) network neighbourhood of the gap, a candidate to fill the gap and a dataset containing gene expression information for the genes in the network, Banjo computes the BDe score of the network. The score indicates how likely it is that the data was generated by the given network, and thus how likely it is that the gap should be filled by the candidate. Note that the standard search capabilities of Banjo are not used for this task as the network structure is already determined and only the score for the data is required for each of the available candidates. The BDe criterion is described in 2.4.1; details about Banjo are given in 2.5

2.9.3 Evaluation of performance through self-ranking

The gene placement method is evaluated by applying the algorithm to known metabolic genes. Gaps are created deliberately in the partially known network by removing genes from it. The candidate set for filling the gap consists of (a subset of) the genes not comprised in the known partial network plus the gene being tested. The self-rank of a known gene is its rank in the candidate gene list ordered using the score of the candidate. The self-rank can range anywhere from 1 to the number of candidates. A self-rank of 1 indicates that the “correct” gene was determined to be the top candidate to fill the gap. A perfect prediction algorithm would return a self-rank of 1 for every known gene, and a random prediction would result in a uniform distribution from 1 to the number of candidates. The overall performance of the algorithm is quantified by calculating the fraction of well-ranked genes, i.e. the fraction of genes that rank among the top K candidates, where K is chosen according to the desired stringency (for example, 20% of the genes are ranked within the top 100 candidates). The self-rank performance criterion is taken from Kharchenko et al (2004).

2.9.4 GeneSim

As a proof of concept, the gene placement method is evaluated on a synthetic problem. GeneSim is used to generate synthetic microarray data for a self created network. GeneSim has already been described in detail in 2.2.

2.9.5 Saccharomyces Cerevisiae

The gene placement method is evaluated using a metabolic network for the *Saccharomyces Cerevisiae*, a species of yeast. A partially known network for this organism was compiled based on biological knowledge (Forster et al., 2003) and attempts are made to fill these using an available microarray dataset (Hughes et al., 2000). The remainder is structured as follows: first, some background information on the organism is given. Then, the way the metabolic network was compiled is explained. Finally, a description of the microarray dataset is given.

2.9.5.1 Background information

"Saccharomyces" derives from Greek, and means "sugar mold". "Cerevisiae" comes from Latin, and means "of beer". It has been used since ancient times in baking and brewing (brewer's yeast and baker's yeast are other names for the organism). It is believed that it was originally isolated from the skins of grapes. It is one of the most intensively studied eukaryotic model organisms (cells with a nucleus) in molecular and cell biology, much like Escherichia coli as the model prokaryote (cells without nuclei). It is the micro-organism behind the most common type of fermentation. *Saccharomyces Cerevisiae* cells are round to ovoid, 5–10 micrometers in diameter. It reproduces by a division process known as budding (please see [Wiki - Yeast] for more details).

Yeasts are interesting organisms for biologists to study. They are safe, easy and cheap to culture, and widely available. Although simple organisms, their study can provide information on molecular processes which are relevant for humans. In fact, many insights about the molecular processes involved in metabolism, biosynthesis, cell division, and other crucial areas of biology have come from the investigation of yeasts (Hunter 1993). According to [Wiki – Yeast], it is estimated that yeast shares about 23% of its genome with that of humans. As some of the most valuable methods in biological research are invasive, or require organisms to be killed, or require several generations of observation, or very large populations, much of this work is impractical or unethical to carry out on humans (Hunter 1993). Yeasts can form an alternative, as the latter is less of a problem with these organisms.

2.9.5.2 Metabolic Network

The genetic network for yeast used in this research was provided by Perry Moerland[1]. A manually curated metabolic network model of the *Saccharomyces Cerevisiae* (the first comprehensive network for a eukaryotic organism, by Forster et al. 2003) was used to construct a network consisting of 785 nodes: 581 nodes representing known genes and 204 nodes representing gaps or "missing" genes. The network contains a total of 5122 connections, each corresponding to dependencies established by metabolic reactions.

Dependencies between genes were established according to the following definition: a gene X is a parent of (regulates) Y if and only if there exists a metabolite that is

(1) Produced by a reaction catalyzed by the product of gene X and
(2) Consumed by a reaction catalyzed by the product of gene Y.

Genes comprised in the metabolic network are in the remainder referred to as "known metabolic genes". While any metabolite can be used to deduce dependencies between genes, the relationships established by common metabolites, like ATP, are not likely to connect genes with similar metabolic functions. In building a global metabolic dependency graph all metabolites were considered, excluding the following highly connected metabolites: ATP, ADP, AMP, CO2, CoA, glutamate, H, NAD, NADH, NADP, NADPH, NH3, orthophosphate and pyrophosphate.

Note that although for the Bayesian gene placement scheme a directed acyclic graph is required, the definition used to establish dependencies does not guard from generating a cyclic graph. In fact, the

[1] Assistant professor, Bioinformatics Laboratory
Department of Clinical Epidemiology, Biostatistics and Bioinformatics
Academic Medical Center, University of Amsterdam

network generated for *Saccharomyces Cerevisiae* contains many cycles, often caused by connections going back and forth between two genes.

2.9.5.3 Microarray dataset

The gene placement algorithm relies on gene expression information to measure how well candidates fit a given gap. The microarray dataset used in this research is known as Rosetta's 'compendium' dataset (Hughes et al., 2000). This dataset measures 300 expression profiles of 6207 genes. The 300 samples were obtained across 287 deletions and 13 chemical perturbations. The data is 'static', i.e. does not represent a time series.

2.9.6 Discretization methods

Microarray measurements used (both synthetic and real) consist of continuous values. As the Bayesian scoring criterion used in this research can handle discrete values only, the measurements have to be discretized before they can be presented to the Bayesian algorithm. In this research, 4 discretization schemes are used: interval discretization, quantile discretization, K-means and a statistically motivated method. Each of these methods is described briefly below:

- With interval discretization, measurements for a given variable are divided into discrete categories, each category spanning a range of equal width.
- With quantile discretization, the values for variables are divided in a number of bins, thereby making sure that each bin contains the same amount of measurements.
- The third method used in this research is the so called K-means method. The K-means algorithm divides the data into K discrete categories, thereby attempting to find the centres of "natural" clusters in the data by minimizing the total variance within each category. Berlo et al (2003) applied this scheme within the context of learning biological networks from data.
- Finally, the fourth discretization scheme is a statistically motivated method taken from Friedman et al (2000). Values are discretized into 3 categories: *under-expressed*, *normal*, and *over-expressed* depending on whether the expression level is significantly lower than, similar to, or greater than a control value (here, the average expression value across experiments). Values with ratio to control lower than $2^{-0.5}$ are considered under-expressed, and values higher than $2^{0.5}$ are considered over-expressed.

The first two schemes are implemented in Banjo. The latter two were implemented for the purpose of this thesis.

2.10 Results

2.10.1 Experiments with synthetic data

As a proof of concept, the performance of the Bayesian gene placement method is first assessed using a relatively simple synthetic problem before it is applied to the more complex 'real world' data. Using GeneSim, datasets of various sizes are sampled from the network depicted in Figure 15. The network consists of a total of 50 genes of which 30 are independent. All connections in the network have strength 0.2. Data was sampled at a rate of 5, and discretized into 3 intervals of equal width using Banjo (these are the same settings that were used in part 1).

To determine the influence of the quantity of data on the performance of the gene placement algorithm, the self-rank of each the 20 genes in the known network was determined using datasets of different size. Each gene was ranked against 31 candidates: the 30 genes not in the network and the gene being ranked. Results are shown in Table 10.

These results show that overall performance of the placement algorithm improves with increasing number of samples. Using 50 samples, the algorithm replaced 13 genes out of 20 perfectly (i.e. the self-rank obtained was 1). This number increases with sample size and when using 300 samples, 18 genes are replaced perfectly.

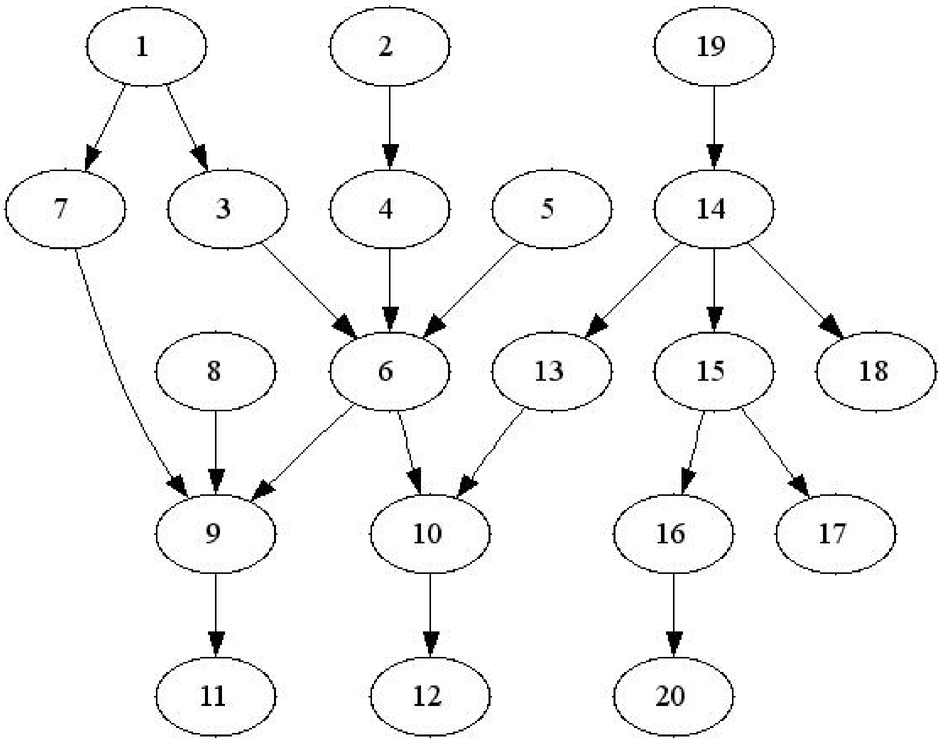


Figure 15: network used with GeneSim in the synthetic data experiments. The network consists of a total of 50 genes of which 30 are independent (not displayed). All connections in the network have strength 0.2. The network is used to evaluate the Bayesian gene placement method by determining the self-rank of each of the genes.

	Number of samples in dataset				
	50	*100*	*300*	*1000*	*5000*
Top1	13	16	18	18	19
Top 5	17	17	18	20	20
Top 10	18	18	18	20	20

Table 10: Results of the self-rank experiments obtained with the synthetic network .The table shows the number of genes (out of 20) that rank amongst the top K candidates (K=1, 5, 10) in the self-rank experiments, using datasets of various size (50, 100, 300, 1000 and 5000 samples).

In Table 11, results for individual genes are presented. For convenience, only the self-rank for genes that were not replaced perfectly are given. It is apparent from the table that genes without parent are the hardest to replace. Of the 7 genes “misplaced” when using 50 samples, only genes 3 and 4 have a parent and their self-rank is almost perfect. When using 100 samples or more, only genes without parents are not replaced perfectly. This is consistent with the results in part I, where it was shown that more samples were required to recover connections from a gene without a parent. It is at least remarkable to see that genes with multiple parents (like genes 6, 9 and 10) are found using 50 samples only. In part I, hundreds of samples were needed to recover this kind of structure completely from scratch.

	Number of samples in dataset				
Gene	*50*	*100*	*300*	*1000*	*5000*
1	10				
2	21	5			
3	2				
4	2				
5	4	25	18	11	
8	5	8	15	3	2
19	31	17			

Table 11: Results of the self-rank experiments obtained with the synthetic network. The table shows obtained self-ranks using datasets of various size (50, 100, 300, 1000, 5000 samples) for individual genes (left column). Genes are ranked against 31 candidates (the 30 independent variables and the gene itself). Only the ranks for genes that were not replaced perfectly by the algorithm are displayed (i.e. only the genes with a self-rank other than 1).

2.10.2 Experiments with yeast

In this section, results for self-rank experiments involving yeast are presented. For practical reasons detailed earlier, the experiments are executed on a third of the known metabolic genes. For clarity, the selection of genes for the self-rank experiments is first described. Then, to provide an impression of the subnetworks involved in the evaluation of candidates, examples of subnetworks for self-made gaps are given. Then, a reasonable discretization method is selected experimentally for the problem at hand by determining the self-rank of a small number of genes using a number of discretization techniques. Finally, the self-rank of the selected genes is determined using the supplied structure and the naive method.

2.10.2.1 Selection of metabolic genes

A selection of known metabolic genes is made as follows: for each of the 581 known genes in the known yeast metabolic network, the subnetwork is determined using the procedure described in the methods section (see 2.9.1.2). Any gene for which the subnetwork is still cyclic after the removal of bidirectional connections or contains one or more genes with more than 10 parents is removed from the selection. In total, 233 known genes qualified on the parent count. Of these, 194 genes had an acyclic subnetwork after removal of bidirectional links. This represents a little more than 33% of the total number of genes in the known metabolic network, enough to provide a good impression of the performance of the gene placement approach proposed.

2.10.2.2 Examples of subnetworks

To give the reader an impression of the subnetworks involved in the ranking of candidates, two simple examples of decycled networks are presented below. The images were generated using the dot format and GraphViz, as described earlier in the methods section (see 2.5.2). The names of the genes are not given; the numbers correspond to the internal references used by Banjo to output networks. The question mark denotes the gap under investigation (the mark was added by hand and is not part of Banjo's output).

It should be noted that these particular examples were selected as illustrations here because they can be displayed conveniently. Many networks however, consist of more genes and can contain many connections between genes, making them hardly suitable for display.

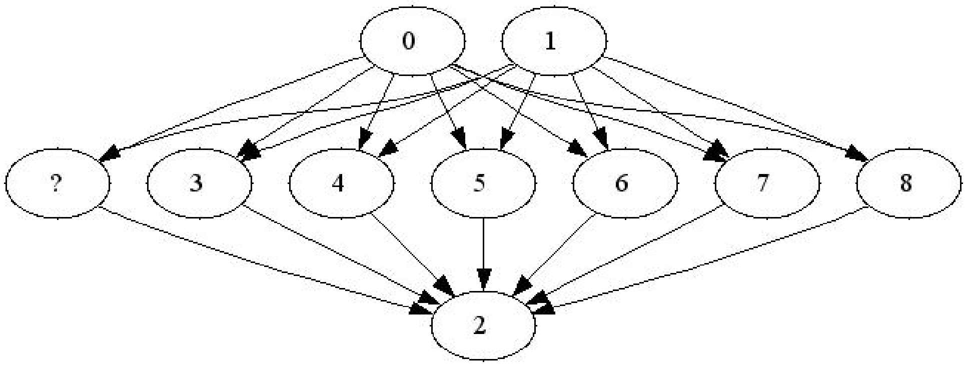


Figure 16: example subnetwork for yeast, consisting of 10 genes including the gap.
In this case, the gap has 6 co-parents to gene number 2.

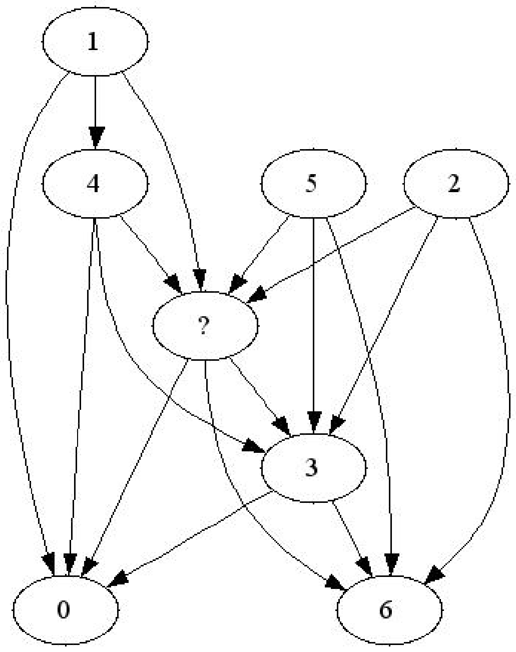


Figure 17: example subnetwork for yeast, consisting of 8 genes including the gap

2.10.2.3 Experiments with discretization

Before the score of any given candidate can be computed by Banjo, the continuous expression data needs to be discretized into a number of categories. It is however *a priori* not clear which discretization method and the number of bins are most appropriate for the particular problem at hand. Here, a reasonable discretization method is selected experimentally by determining the self-rank of 10 genes selected randomly from the set of 194 genes against a set of 500 candidates (out of the possible 5626) using the following four discretization schemes: interval, quantile, K-means and a statistically motivated method followed by Friedman, as described in the methods section. As stated before, the first two schemes are implemented in Banjo. The latter two were implemented for the purpose of these experiments. Results of these experiments are presented below.

For comparison purposes, a random gene placement method would on average result in 1 gene ranked in the top 50 and 5 genes in the top 250.

It can be seen from Table 12 that interval discretization does not perform any better than a random placement method. With interval discretization, none of the 10 genes is ranked within the top 50. Results seem to get worse as the number of used bins grows. Using quantile discretization, the Bayesian placement algorithm does perform better than a random placement strategy. Best results are obtained using 3 bins, yielding 4 genes being placed within the top 20, and 7 in the top 250. Table 14 shows the results obtained with K-means. Using 4 bins, it seems to perform as well as with quantiles. As performance seems to improve with the number of bins, this method is also investigated using 5 bins, but this resulted in a drop in performance.

Rank	2 bins	3 bins	4 bins
<=10	0	0	0
<=20	0	0	0
<=50	0	0	0
<=100	2	0	1
<=250	5	3	4

Table 12: results using interval discretization

Rank	2 bins	3 bins	4 bins
<=10	2	3	2
<=20	2	4	3
<=50	4	4	3
<=100	4	4	4
<=250	6	7	6

Table 13: results using quantile discretization

Rank	2 bins	3 bins	4 bins	5 bins
<=10	0	0	2	2
<=20	0	0	2	2
<=50	1	2	2	2
<=100	2	2	3	2
<=250	5	5	7	4

Table 14: results obtained using K-means discretization

Rank	3 bins
<=10	2
<=20	2
<=50	2
<=100	3
<=250	4

Table 15: results obtained using the statistically motivated method.

Concluding, quantile and K-means perform best. It seems that K-means with 4 bins is competitive to the quantile scheme with 3 bins. However, with quantiles, more genes are ranked better: 4 genes are found in the top 20, versus only 2 using K-means. These results indicate that quantile discretization in 3 bins seems to be a reasonable method to discretize the data and this method will therefore be used to conduct the remainder of the experiments.

2.10.2.4 Self-rank of known metabolic genes

The Bayesian gene placement method is evaluated by determining the self-rank of each of the selected metabolic yeast genes. Based on the results of the experiments with discretization, quantile discretization is used to discretize the data into 3 bins. For practical reasons, not all the genes are ranked against all the 5626 candidates available, rather it is determined which genes can be ranked within the top 100 by the gene placement algorithm. This is done by first computing the score of the removed gene itself and then evaluating candidates for the gap until at least 100 candidates with a better score are found or there are no more candidates left. This scheme greatly reduces computing time. Results are presented in Table 16 and Figure 18.

	Supplied structure		Naive Bayesian		Random
Self-Rank	Number of genes	Percentage of genes	Number of genes	Percentage of genes	Percentage of genes
Top 1	1	0.52	5	2.6	0.02
Top 5	3	1.55	7	3.6	0.09
Top10	11	5.67	15	7.7	0.18
Top 20	18	9.28	25	12.9	0.36
Top 50	30	15.46	36	18.5	0.89
Top 100	38	19.59	47	24.2	1.78
*	194	100	194	100	100

Table 16: Validation of known metabolic yeast genes. The table shows the distribution of self-ranks for the 194 genes selected from the known metabolic network, as predicted by the algorithm. Data was discretized using quantile discretization into 3 bins. For comparison, the right column shows the results that can be expected for a random gene replacement method.

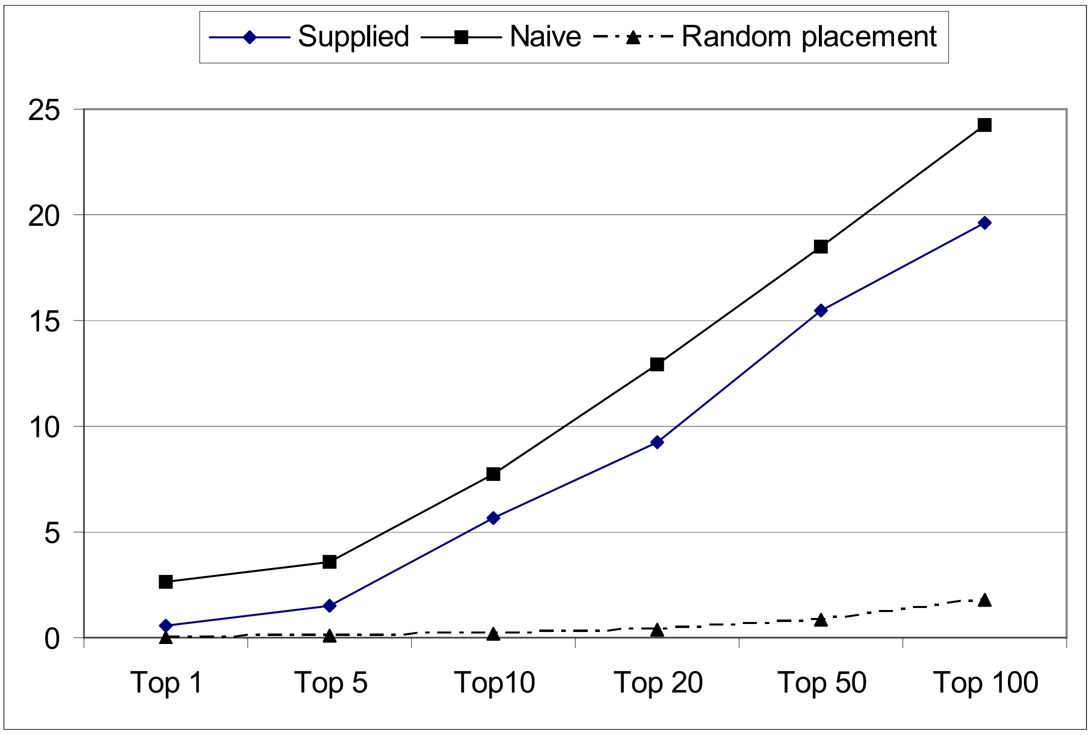


Figure 18: Percentage of known yeast genes that were ranked within given thresholds for different placement methods, using the structure in the supplied metabolic network and naive ranking. For comparison, results that would be obtained using a random placement strategy are shown as well.

2.11 Conclusions & discussion

A strategy for evaluating candidates for filling gaps in a partially known network was described. The Bayesian gene placement algorithm is designed to evaluate candidate genes for a particular gap. Given a gap in the supplied network, a set of candidate genes, and gene expression information, the placement algorithm ranks the list of candidate genes: the first gene in the list being the most probable candidate for the gap, and the last gene being the least probable candidate. The ordering is determined by a Bayesian metric which assigns a score to each candidate indicating how likely it is that the data was generated by the supplied network neighbourhood.

To demonstrate its viability, the Bayesian gene placement method was first evaluated on a relatively small synthetic network simulated using GeneSim, with datasets of different size. The results showed that overall performance of the method increases with the number of samples, as the number of genes replaced perfectly grew with sample size. It was shown that - in GeneSim - genes without regulators were the hardest to replace. Genes with multiple regulators however, were replaced using 50 samples. In contrast, in part 1, hundreds of samples were needed to recover structures with multiple parents from scratch. Although these figures are comparative, they suggest that data requirements for completing biological pathways are considerably lower than when learning networks from scratch using gene expression values only. This is an encouraging result, as generally, microarray datasets tend to be small.

The method was also evaluated on a known problem involving a yeast (*Saccharomyces Cerevisiae)* network. It was shown that discretization is a critical factor on the performance in self-rank experiments. The following four methods were examined: interval discretization, quantile discretization, k-means and a statistically motivated method. Worse than random results were obtained using interval discretization, while best results were obtained using quantile discretization with 3 bins. These results show that discretization has to be chosen with care before attempting to fill gaps with the Bayesian networks approach. A suitable discretization method could be selected using self-rank experiments on known genes before trying to fill real gaps, as was done in this research.

Two different methods for determining the network neighbourhood were tested on a third of the genes in the supplied yeast network. Using the structure of the network neighbourhood given in the supplied network, 15% of the genes were predicted within the top 50, and 19% within the top 100 (out of 5626 possible candidates). Using the naive approach, ignoring the relations given in the supplied network and assuming conditional independence between genes given a candidate, 18% of the genes were predicted within the top 50 and 24% within the top 100. It is interesting to see that the naive approach performed better in spite the far-reaching simplifying assumptions of conditional independence. An explanation for this could be that the dataset used is too small to justify the relatively complex structure of the supplied subnetworks; with subnetworks containing genes with many parents, it is possible to overfit or ‘explain away’ the data. With the much simpler structure of the naive subnetworks, this is less likely.

Although results are reasonable for a first attempt with Banjo ‘out of the box’, neither Bayesian gene placement methods performed better than Kharchenko et al (2004), as they claim that their method ranked more than 20% of the genes within the top 50. No clear reason can be given for this, and a number of issues render an exact comparison difficult. Slightly different metabolic networks were used in the studies. As mentioned before, Kharchenko et al (2004) use a representation for networks with undirected edges. Perhaps this representation is more suited for the problem at hand, recalling that many links in the Bayesian network used in this study were bidirectional. Further, experiments in this study were conducted on a limited number of genes, whereas Kharchenko et al (2004) used all genes in the known metabolic network. Although it is felt that the selection of genes was large enough to give a good idea of the performance of the Bayesian gene placement method, it is of course possible that different selections of genes would yield different impressions of the performance.

However, no definitive conclusions can be drawn about Bayesian networks and the missing genes problem in general based on the experiments presented in this thesis. Several factors could affect results, for example implementation details of the approach used, the way the metabolic network was prepared, or the particular organism studied. It was not possible to investigate the impact of all these factors within this thesis, but some of these are briefly reviewed in the following.

All missing genes experiments were carried out with a standard implementation of Bayesian networks, without any modifications to the algorithm. One Bayesian metric, the Bde score, was used to rank candidates, and it is quite possible that different results could be obtained with other existing Bayesian metrics (see Heckerman (1996) for an overview of often used metrics). Also, for practical reasons, only gaps for which the subnetwork contained a limited number of parents per gene were investigated. Perhaps that including networks with a larger number of parents in the experiments would provide a different impression of the performance of the placement algorithm.

Another important factor that could affect results is the way the metabolic network is prepared. Here, as described in the methods section, the network was built considering a single enzyme at a time as catalyst for a given reaction between metabolites. It is however conceivable that a combination of enzymes should be considered instead; perhaps some metabolites would react in a different way in the presence of several enzymes. This could yield a metabolic network which is quite different than the one used in this study (and possibly one which will be harder to model accurately with a naive Bayes approach). Another crucial step in the preparation of the network is the determination of the subnetwork around gaps. Here, the network contained many bidirectional links, and a strategy for removing these cycles had to be devised. Of course, the resulting network will depend on details of the particular scheme chosen. For instance, when building subnetworks using the approach described in 2.9.1.2 “Dealing with cyclic networks”, it was decided that a gene with a bidirectional link with the gap was to be kept as a parent to the gap in the subnetwork. This seemed reasonable, but the gene could have been made a child to the gap instead (this applies to genes with a bidirectional links with the gap’s children as well). This would result in different subnetworks, and possibly, in different self-rankings.

Finally, in this thesis, attempts were made to complete genetic networks for one organism only. Although yeast is a model eukaryotic organism (with a lot in common with human beings), results cannot be generalized with certainty to other eukaryotic organisms. Good (or bad) performance of the gene placement algorithm in the completion of pathways for yeast does not necessarily imply that similar results on pathways for say human beings. As more knowledge is accumulated, and more genetic pathways are reconstructed, it will be interesting to see how the proposed Bayesian gene placement approach performs on other organisms.

To conclude, it was shown in part 1 that when learning networks from scratch, the number of samples required for good recovery grows rapidly with the number of regulators per gene in the network. As datasets tend to be of small sample size in practice, it does not seem reasonable to expect to infer complex structures completely using gene expression values only. However, for the completion of nearly complete networks, the “missing genes” problem, results in this part of this thesis suggest that sample size requirements are reduced considerably. It seems that with the sample size of current datasets, Bayesian networks can be used to complete genetic networks with some success.

3 References

[Banjo Homepage] download location for the software package Banjo and its User Guide. http://www.cs.duke.edu/~amink/software/banjo/

[Berlo et al, 2003] R.J.P. van Berlo, E. P. Van Someren, M. J..T Reinders, “Studying the Conditions for Learning Dynamic Bayesian Networks to Discover Genetic Regulatory Networks”. Simulation, Vol. 79, Issue 12, December 2003 689-702.

[Bernard et al, 2005] Bernard, A, Hartemink, A. (2005) “Informative Structure Priors: Joint Learning of Dynamic Regulatory Networks from Multiple Types of Data.” In Pacific Symposium on Biocomputing 2005 (PSB05), Altman, R., Dunker, A.K., Hunter, L., Jung, T., & Klein, T., eds. World Scientific: New Jersey. pp. 459–470.

[Chickering et al, 1996] Chickering, D. M., and D. Heckerman. (1996) “Efficient approximation for the marginal likelihood for incomplete data given a Bayesian network”. Tech. Report MSR-TR-96-08, Microsoft Research, Redmond, WA.

[Cooper et al, 1992] Cooper, G. F. & Herskovits, E. (1992), “A Bayesian method for the induction of probabilistic networks from data”, Machine Learning 9, 309–347.

[Forster, 2003] Forster, J., Famili,I. Fu,P., Palsson, B.O. and Nielsen,J. (2003) *Genome-scale reconstruction of the Saccharomyces Cerevisiae metabolic network*. Genome Res., 13, 244–253.

[Friedman et al, 1998] Friedman, N., Murphy, K., Russell, S. (1998), Learning the structure of dynamic probabilistic networks, in `Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)', pp. 139–147.

[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, pages 601-620

[Friedman] Friedman N, “Inferring Cellular Networks Using Probabilistic Graphical Models”. Science, Vol 303, 6 February 2004

[Gardner et al, 2005] Gardner T.S., Faith J.J. Reverse-engineering transcription control networks. Physics of Life Reviews 2 (2005) 65–88

[Handley 2002] Evaluating Machine Learning Algorithms Used to Infer Gene Regulatory Network Structure, Msc Thesis 2002, Carnegie Mellon University, Departement of Philosophy (Logic and Computation), Dan Handley

[Hartemink et al 2001] Hartemink, A. 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”. Proceedings Pacific Symposium on Biocomputing, 422-433, 2001

[Hartemink et al 2002] Hartemink, A. J., Gifford, D. K., Jaakkola, T. S. and Young, R. A. (2002). “Combining location and expression data for principled discovery of genetic regulatory network models”. Pac. Symp. Biocomput. 7, 437-449.

[Heckerman D, 1996] “A Tutorial on Learning with Bayesian Networks, March 1995 (revised November 1996)”. Technical Report MSR TR 95 06, Microsoft Research.

[Hughes, 2000] Hughes,T.R., Marton,M.J., Jones,A.R., Roberts,C.J., Stoughton,R., Armour,C.D., Bennett,H.A.,Coffey,E., Dai,H., He,Y.D. et al. (2000) *Functional discovery via a compendium of expression profiles*. Cell 102, 109–126.

[Hunter, 1993] Lawrence Hunter (editor), Artificial Intelligence and Molecular Biology (AAAI Press Copublications). Publisher: AAAI Press (March 18, 1993) ISBN-10: 0262581159, ISBN-13: 978-0262581158

[Husmeier, 2003] Husmeier D., "Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks". Bioinformatics, vol 19 no 17 2003, pages 2271-2282
[Imoto et al, 2003] 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”, Proceedings of the computational Systems Bioinformatics, 2003

[Kharchenko et al, 2004] Peter Kharchenko, Dennis Vitkup, George M. Church (2004) “Filling Gaps Filling gaps in a metabolic network using expression information”, Bioinformatics, Vol. 20 Suppl. 1 2004, pages i178–i185

[Markowetz] Markowetz F., Spang R., “Evaluating the Effect of Perturbations in Reconstructing Network Topologies.” Proceedings of the 3rd international Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. ISSN 1609-395X.

[Munich] Munich Information Center for Protein Sequences. http://mips.gsf.de/genre/proj/yeast/index.jsp

[Neapolitan 2003] Neapolitan R. E., “Learning Bayesian Networks”, Prenctice Hall Series in Artificial Intelligence, ISBN10: 0130125342, ISBN13: 9780130125347

[Osterman et al 2003] Andrei Osterman and Ross Overbeek. “Missing genes in metabolic pathways: a comparative genomics approach”. Current Opinion in Chemical Biology 2003, 7:238-251.

[Pe’er et al 2001] Pe’er,D., Regev,A., Elidan,G., and Friedman,N. (2001) “Inferring subnetworks from perturbed expression profiles.” Bioinformatics, 17, S215–S224.

[Pearl 1988] Pearl, J. “Probabilistic Reasoning in Intelligent Systems: networks of plausible inference”. 1988

[Phillip et al. 2004] Phillip P. Le,a, Amit Bahl,a and Lyle H. Ungar "Using prior knowledge to improve genetic network reconstruction from microarray data" In Silico Biology 4, 0027 (2004); ©2004, Bioinformation Systems

[Robinson 1977] Robinson, R.W. “Counting Unlabeled Acyclic Digraphs” Lecture notes in mathematic, 622: Combinatorial Mathematics V, Springer-Verlag, New-York, 1977

[Smith et al, 2002] Smith, V., Jarvis, E., & Hartemink, A. (2002) “Evaluating Functional Network Inference Using Simulations of Complex Biological Systems.” Intelligent Systems in Molecular Biology 2002 (ISMB02), *Bioinformatics*, 18:S1. pp. S216–S224.

[Smith et al, 2003] Smith, V., Jarvis, E., & Hartemink, A. (2003) “Influence of Topology and Data Collection on Functional Network Inference.” In *Pacific Symposium on Biocomputing 2003 (PSB03)*, Altman, R., Dunker, A.K., Hunter, L., Jung, T., & Klein, T., eds. World Scientific: New Jersey. pp. 164–175.

[Wiki – Naive Bayes] Wikipedia, the free encyclopedia, information page for naive Bayes classifiers (25 march 2007), http://en.wikipedia.org/wiki/Naive_Bayes_classifier

[Wiki - Yeast] Wikipedia, the free encyclopedia, information page for S. cerevisiae (25 march 2007), http://en.wikipedia.org/wiki/Saccharomyces_cerevisiae

[Yeastgenome] The yeast genome database. http://www.yeastgenome.org/

[Yu et al, 2002] Yu, J., Smith, V., Wang, P., Hartemink, A., & Jarvis, E. (2002) “Using Bayesian

Network Inference Algorithms to Recover Molecular Genetic Regulatory Networks." International Conference on Systems Biology 2002 (ICSB02), December 2002.

[Yu et al, 2004]Yu, J., Smith, V., Wang, P., Hartemink, A., & Jarvis, E. (2004) "Advances to Bayesian Network Inference for Generating Causal Networks from Observational Biological Data." *Bioinformatics*, 20, December 2004. pp. 3594–3603.