TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK

Large-Scale Bayesian Network Structure Learning

Andreas Nägele

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Helmut Seidl
Prüfer der Dissertation:		
	1.	Hon.-Prof. Dr. Thomas Runkler
	2.	Univ.-Prof. Dr. Burkhard Rost

Die Dissertation wurde am 27.09.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.02.2013 angenommen.

Abstract

Since more than two decades, probabilistic graphical models have gained more and more attraction for analyzing data or modeling dependencies. Especially Bayesian belief networks, also called Bayesian networks (BN), allow a very intuitive representation of dependencies among a set of random variables. Each variable is represented as a node in a graph, and dependencies are modeled as directed edges among the nodes. By linking each variable with a conditional probability distribution, BNs form a joint probability distribution over the random variables. Learning the structure of Bayesian networks and thus revealing the dependency structure among variables from data is one of the most challenging tasks in the area of BNs.

This thesis is mainly concerned with methods to learn Bayesian networks in large domains. A typical approach to make structure learning feasible in large domains is the reduction of the number of variables: Variables ought to be not interesting for a specific analysis are removed and the structure between the remaining variables is learned. While this approach of learning a smaller subnetwork solves the performance issue, the learned structure might not represent the true dependency structure since many variables are invisible for the learning process. In a dedicated chapter of this thesis the impact on the quality of the learned structure is analyzed and a new method is developed that measures the robustness of edges against such missing variables. The contribution in this chapter is three-fold: First, we show that the commonly used statistically-based pre-selection of variables has a negative effect on the quality of the learned network by means of learning statistical fluctuations in the data. Second, we use subnetwork learning as a method to measure the network errors learned because of the reduction of the number of variables and show, based on several benchmark data, that the number of false positive edges is at least doubled compared to the case if the network is learned completely. Third, with dimensional bootstrap we introduce a method to estimate the confidence in edges in such situations in order to efficiently identify false edges in the network.

We used the idea of omitting many variables and learning the network only for a small subnetwork in a more systematic way to provide new efficient algorithms to learn the structure in large domains with high quality. The first structure learning algorithm developed in this thesis is based upon the idea to learn small network substructures independently from each other. After learning small substructures in such a way that all variables are covered, all the small networks are combined in a single graph representing the dependency structure. However, this graph structure comes along with one problem: Since the substructures are learned independently from each other, the resulting graph is not necessarily acyclic, but may contain cycles in the directed structure. On the other hand, one basic property of the graph of a Bayesian network is its acyclicity. As a result, this algorithm lacks the representation of a single Bayesian network structure. To overcome this problem, a second algorithm, called S-DAG, is introduced in this thesis that combines the small substructures and builds an acyclic Bayesian network structure out of the small networks. In a dedicated chapter, these two algorithms are introduced and their performance is evaluated. For the latter algorithm (S-DAG), a comparison study with several other state-of-the-art Bayesian network structure learning algorithms is carried out. We show that S-DAG outperforms other well-known BN structure learning algorithms like the very competitive algorithm MMHC.

The previously mentioned reduction of the number of variables is commonly used if the so-called genetic network is reconstructed by means of Bayesian networks. The genetic network, which controls all life processes, is mainly formed by mutual biochemical interactions between DNA, RNA, and proteins. With structure learning based on microarray data, these mutual interactions can be revealed on a higher level: All the complex interactions are modeled as abstract gene-gene interactions in the network. In another chapter of this thesis, a complete genetic netwok estimation is learned with the new efficient structure learning algorithm. An analysis of important structural features of the genetic network that were found and published based on small subnetworks is carried out on the complete network. With more than 50.000 variables in one network, we learned one of the largest unrestriced Bayesian networks with our new algorithm S-DAG.

Contents

Abstract iii

1 Introduction 1

1.1 Motivation and Goal . . . 3

1.2 Overview . . . 5

2 Bayesian Networks 7

2.1 Definition of Bayesian Networks . . . 7

2.1.1 Factorization . . . 7

2.1.2 Independence . . . 11

2.2 Equivalence and Causality . . . 15

2.3 Faithfulness . . . 21

3 Learning Bayesian Networks 23

3.1 Parameter Learning . . . 24

3.2 Structure Learning . . . 25

3.2.1 Score-based Approach . . . 26

3.2.2 Constraint-based Approach . . . 28

3.3 Structure Learning Algorithms . . . 29

3.3.1 Greedy Hill Climbing: GS / MMHC . . . 31

3.3.2 Simulated Annealing: SA and MMSA . . . 32

3.3.3 Ant Colony Optimization: ACO, MMACO and related algorithms 34

3.3.4 Constraint Hill Climbing: CHC, CHC*, iCHC, 2iCHC . . . 37

3.3.5 Recursive Autonomy Identification: RAI . . . 38

4 Robust Learning in Large Domains 39

4.1 Robustness Assessment . . . 42

4.1.1 MCMC Methods . . . 43

4.1.2 Data Bootstrap 44
4.1.3 Feature Graph 45
4.2 Sparse Data Analysis 47
4.2.1 Method 48
4.2.2 Results 51
4.2.3 Discussion and Related Work 55
4.3 Subnetwork Learning 56
4.3.1 Method 58
4.3.2 Results 59
4.4 Dimensional Bootstrap 64
4.4.1 Method 64
4.4.2 Results 68
4.5 Summary 71

5 Large-Scale Network Learning 75
5.1 Background 76
5.2 Substructure Learning 77
5.2.1 Algorithm 77
5.2.2 Time Complexity of Substructure Learning 79
5.2.3 Results 80
5.2.4 Discussion 84
5.3 Substructure DAG Learning 85
5.3.1 Algorithm 85
5.3.2 Results 90
5.3.3 Comparison to Other BN Structure Learning Algorithms 103
5.3.4 Discussion 117
5.4 Summary 118

6 Estimating Genetic Networks 121
6.1 Introduction 121
6.2 Biological Data 122
6.2.1 Rosetta Compendium 126
6.2.2 Oncology Data Set 129
6.3 Summary 129

7 Conclusions **131**

A Appendix **135**

A.1 Algorithm MMPC (Max-Min Parents and Children) 135

A.2 Benchmark Networks 136

A.3 Detailed Results of Structure Learning Algorithms 138

Bibliography **157**

1 Introduction

More than 50 years ago, Artificial Intelligence (AI) was established as a new research area during a conference on the campus of Dartmouth College (McCarthy, Minsky, Rochester & Shannon 1955, Russell & Norvig 2009). In the following years, research in that area received more and more attraction induced by astounding results. For instance, Winograd (1972) wrote a program to deal with geometric objects in a small world based on a language parser that allowed a user interaction by giving instructions in English terms. Motivated by such results, optimists even predicted that Artificial Intelligence will be capable within two decades to do everything what human beings are able to do (Simon 1965). Contrary to the enthusiasm of the early stage, the AI community failed to fulfill the expectations that were produced and maintained by themselves. In the mid 1970's and at the end of the 1980's, AI fall two times into big crisis known as the two AI winters (Russell & Norvig 2009), also triggered by the realization that reasoning cannot be solely based on traditional logic (McCarthy 1987).

Besides other proposals that were made to overcome the limitations of traditional logic, Bayesian Networks (BN) were introduced by Pearl (1988). They belong to the class of probabilistic graphical models that can deal with incomplete, uncertain and even contradicting information by using a probabilistic approach to describe dependencies.

Probabilistic graphical models (or graphical models) are a useful and widely used framework that combines uncertainty and logical structure to represent and model complex statistical relationships by describing dependencies among random variables in a graph-theoretic as well as a probabilistic way. Not least due to their flexibility and intuitive representation, graphical models have been attracted increasing attention during the last two decades (Pearl 1988, Lauritzen 1996, Jensen 2002, Cowell, Dawid, Lauritzen & Spiegelhalter 2003, Jordan 2004, Koller & Friedman 2009). The high-level goal of such models is the effective representation of a joint probability distribution p over a set of n random variables $\mathbf{X} = \{X_1, ..., X_n\}$. Even in a very simple case where each random variable can have only two different states, in total the variables $\mathbf{X}$ have 2^n different assignments

$x_1, ..., x_n$, and each assignment has to be specified separately. Without any further assumptions, multivariate models with more than only few variables would be intractable large. However, usually there is some structure in the distribution that enables the factorization into smaller components. By taking systematic advantage of conditional independencies among random variables, graphical models enable a compact representation of joint probability distributions.

As the name *probabilistic graphical models* suggests, conditional dependencies are graphically represented by means of edges (also called links or arcs) between nodes (each single variable is represented as a node in the graph), while missing edges represent conditional independencies. Then the graph captures the way in which the joint probability distribution can be decomposed into a product of smaller factors. The most prominent and widely used representatives of graphical models are Markov networks (also called Markov random fields, MRF) and Bayesian networks (belief networks). In Markov networks, the semantics of conditional dependencies are based on undirected graphs, hence Markov networks are undirected graphical models. In contrast, in Bayesian networks the semantics of conditional dependencies are based on directed graphs. Hence, BNs are directed graphical models.

BNs emerged to be a quite useful tool in many domains and were used to tackle many problems. A lot of research was done during the last years in the field of BNs (Kojima, Perrier, Imoto & Miyano 2010, Schulte, Frigo, Greiner & Khosravi 2010, Scutari & Brogini 2011, Wu, McCall & Corne 2011). Due to their intuitive graphical representation together with a sound theoretical basis, they are widely-accepted tools for both modeling knowledge and making predictions in different domains. For instance, medical diagnosis relies on an increasing amount on diagnostic tests, with the challenge to identify diseases or a high risk for a disease with high accuracy. Here, Bayesian networks can be used to model the dependency between a disease and risk factors, diagnostic tests, and other factors influencing the disease. Based on these factors, the Bayesian network can be used for diagnosis. E.g. HEPAR-II is a Bayesian network for the diagnosis of liver disorders (Kraaijeveld & Druzdzel 2005), and DIAVAL can be used to diagnose heart diseases (Díez, Mira, Iturralde & Zubillaga 1997). Recently, it has been shown that the detection of wrong blood in the tube, a common problem in blood transfusion, based on BNs outperforms medical experts (Doctor & Strylewicz 2010). But the application of BN is not restricted to the biomedical area: for instance, Davis (2003) reconstructs the cause of a traffic accident by means of Bayesian networks.

1.1 Motivation and Goal

While many networks in the aforementioned cases were build up by experts, or at least the graph structure of the network was mainly defined by experts and just the probability distributions were learned from data, a more challenging task is the structure learning of Bayesian networks. Thereby, the conditional dependencies and independencies among the variables must be extracted from data in order to build up the BN structure (Pearl & Verma 1991, Cooper & Herskovits 1991, Lam & Bacchus 1994, Heckerman 1995, Heckerman, Geiger & Chickering 1995, Friedman 1998, Darwiche 2009).

The problem of learning the structure from a data set could be tackled in the following way: With a small number of variables, it is possible to iterate over all possible network structures. Each structure is rated by a score that valuates the compliance of the network and the given data. The network that best fits the data (the network with the best score) is taken as result of the learning procedure. However, this approach has a severe drawback: The number of possible different network structures grows super-exponentially with the number of variables. Chickering, Geiger & Heckerman (1994) have proven that structure learning of BNs is generally a $\mathcal{NP}$-hard problem. The extreme amount of different network structures can be seen by an example: For a network with only 10 variables, there are almost 10^{20} different possible BN structures, which prohibits an exhaustive search over all possible network structures (Hofmann 2000). Thus, it became a common approach to apply heuristic search strategies to learn a BN network structure.

Two general types of learning algorithms have been developed over the last two decades: The first, known as *Score-based* algorithms, utilize a scoring function to guide heuristic search strategies. The goal of these algorithms is to find the network with the best score. This is typically achieved by applying local changes to the network like edge addition, edge removal or edge reversal until a high-scoring network is found. *Constraint-based* algorithms form the second class of BN structure learning algorithms and rely on the BN definition based on independence relationships. These algorithms perform statistical tests to determine dependencies and independencies among the variables in order to reconstruct the BN structure. A lot of research was done in both areas leading to a variety of different learning algorithms (Lam & Bacchus 1994, Heckerman et al. 1995, Friedman 1998, Spirtes, Glymour & Scheines 2001, Darwiche 2009). A few years ago, Max-Min Hill-Climbing (MMHC) (Tsamardinos, Brown & Aliferis 2006) as a quite competitive algorithm in terms of network quality and computational effort for learning was introduced. This algorithm combines both approaches: In a first step, an undirected network structure is created by using constraint-based techniques. In a second step, a score-based search is used to orient the undirected edges.

There are two main factors for all score-based algorithms: The quality of the scoring function and the quality of the heuristic search algorithm. There are several approaches for different scoring functions (Cooper & Herskovits 1991, Lam & Bacchus 1994, Heckerman et al. 1995, Roos, Silander, Kontkanen & Myllymaki 2008). In this thesis, we use the *B*ayesian *D*irichlet score with likelihood *e*quivalence and *u*niform prior (BDeu score) (Heckerman et al. 1995) and focus on the second factor to improve structure learning. Since the BDeu score is a widely used score, structure learning algorithms that base upon this score can be often directly compared to other methods. For the structure search, a greedy hill climbing algorithm, sometimes enhanced by a TABU search, is often used (Heckerman et al. 1995, Tsamardinos, Brown & Aliferis 2006).

For most structure learning algorithms, learning the structure of networks with reasonable size is feasible. However, in some domains there are not only hundreds, but even thousands or tens of thousands of variables. For instance, one of the applications of Bayesian network structure learning is the estimation of biomolecular processes in cells. Here, BNs are used to learn abstract gene-gene interactions in the so called genetic regulatory network from microarray data (Friedman, Linial, Nachman & Pe'er 2000). With around 30.000 human genes, the genetic network is much larger than the artificial networks that are typically used to benchmark the learning algorithms. To get a feasible set of variables for structure learning, one normally reduces the size of the network by means of a feature-selection method (Friedman et al. 2000) and learns the structure between the remaining variables. Although this became a normal approach, there is no information about the influence of this approach on the network quality. So the first problem that is tackled in this thesis can be summarized in the following way: Does the reduction of dimensionality affect the quality of the learned network structure?

Most score-based algorithms are not feasible to learn in high-dimensional domains. In the year 2000, Silverstein, Brin, Motwani & Ullman (2000) even stated that "In our view, inferring complete causal models (i.e., causal Bayesian networks) is essentially impossible in large-scale data mining applications with thousands of variables" (Silverstein et al. 2000). An exception is MMHC (introduced in 2006) which is performant enough to learn in such domains. The second problem tackled in this thesis is to find algorithms that learn high dimensional network structures at least with the same runtime characteristics as

MMHC, but produce networks with higher quality.

1.2 Overview

The work in this thesis concerns about learning the structure of Bayesian networks with high accuracy and presents new efficient algorithms for this goal.

Chapter 2 provides a brief introduction to Bayesian networks. After the definition, the most important features of BNs are presented.

State-of-the-art and commonly used structure learning algorithms are introduced in chapter 3. These two chapters introducing Bayesian networks are partially based upon Nägele (2005), Nägele, Arndt & Dejori (2008) and Pinto, Nägele, Dejori, Runkler & Sousa (2009).

In chapter 4 the robustness of Bayesian network structure learning is studied if applied in high-dimensional spaces. The contribution is three-fold: One existing and one novel method are used to analyze the common approach of reducing the size of the network by means of a feature-selection method. The two methods were applied to benchmark data to show the influence of feature reduction together with various sample sizes on the quality of the estimated network structure. Furthermore, Dimensional Bootstrap is introduced as a new method to detect variables that are not selected by the feature-selection method, but would increase the network structure quality if they were added to the set of variables for structure learning (Nägele, Dejori & Stetter 2009).

In chapter 5 the focus lies on learning large-scale Bayesian networks. Two new structure learning algorithms that are especially suitable for large domains are introduced. The first algorithm, substructure learning, splits the learning task into many small tasks by dividing the network into small subnetworks and learning each network separately. The second algorithm, S-DAG, utilizes the small subnetworks and combines them to build up a complete BN structure comprising all variables (Nägele, Dejori & Stetter 2007, Nägele et al. 2008).

In chapter 6 the S-DAG BN structure learning algorithm is applied to gene expression data in order to provide an estimation of the genetic network. The main contribution is the application of BN learning on two complete microarray data sets without feature reduction and the analysis of network properties that were formerly found by subnetwork analysis (Nägele et al. 2008).

Based on a cooperative work, some other algorithms were developed that are not included in this thesis, but are used as comparison algorithms for the S-DAG algorithm

(Pinto, Nägele, Dejori, Runkler & Sousa 2008, Pinto et al. 2009). The implementation of some of the algorithms presented here uses a matrix package developed by Arndt, Bundschus & Nägele (2009).

2 Bayesian Networks

Bayesian networks belong to the class of probabilistic graphical models that represent the dependency structure among random variables by means of a graph. In this thesis, the focus lies on Bayesian networks which have a directed acyclic graph (DAG) as dependency structure. The acyclicity of the graph has the drawback that cyclic dependencies can not be modeled with BNs. However, the advantage of Bayesian networks lies also exactly in the acyclic dependency structure: The graph structure can be directly mapped to the factorization of the joint probability distribution. This allows an intuitive definition of Bayesian networks which is presented in section 2.1.1. In this chapter Bayesian networks and their properties are introduced, while the next chapter gives an overview of methods to learn Bayesian networks from data.

2.1 Definition of Bayesian Networks

There are two ways to define Bayesian networks (BNs). The first utilizes the ability of BNs to represent a probability distribution p among a set of variables $\mathbf{X}$. Thereby, the factorization of p defines the graph structure of the BN. The second definition of BNs uses conditional dependencies and independencies between the random variables $\mathbf{X}$ and determines the graph structure $\mathcal{G}$ of the BN according to these dependencies and independencies. We present both definitions and start with the definition using the factorization, following the way how Hofmann (2000) introduced Bayesian networks. For a more detailed introduction we refer to Heckerman (1995).

2.1.1 Factorization

Given a set of n random variables $\mathbf{X} = \{X_1, X_2, X_3, ..., X_n\}$, the joint probability function for $\mathbf{X}$ can be written as

$$p(X_1, ..., X_n) = \prod_{i=1}^{n} p(X_i \mid X_1, ..., X_{i-1}). \tag{2.1}$$

according to the chain rule of probability. Let the *parents* $\mathbf{Pa}_i$ of X_i be the minimal subset of $\{X_1, ..., X_{i-1}\}$ ($\mathbf{Pa}_i \subseteqq \{X_1, ..., X_{i-1}\}$) such that

$$p(X_i \mid X_1, ..., X_{i-1}) = p(X_i \mid \mathbf{Pa}_i). \tag{2.2}$$

This means that there exists a set of variables for each variable X_i, called *parents* $\mathbf{Pa}_i$, such that X_i and $\{X_1, ..., X_{i-1}\} \setminus \mathbf{Pa}_i$ are conditionally independent given $\mathbf{Pa}_i$. In other words: each variable X_i is conditionally independent of its non-descendants given its parents $\mathbf{Pa}_i$. This property is often referred to as *local Markov property* (Russell & Norvig 2009). Thus, equation 2.1 can be reduced to

$$p(X_1, ..., X_n) = \prod_{i=1}^{n} p(X_i \mid \mathbf{Pa}_i). \tag{2.3}$$

In a graphical context, the variable X_i can be also denoted as node. The node X_i is also called a *child* of the nodes $\mathbf{Pa}_i$. The parents $\mathbf{Pa}_i$ and the node X_i are denoted as the *family* of X_i.

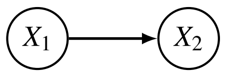
The factorization given in equation 2.3 can be represented as a directed graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, in particular as a directed acyclic graph (DAG). The nodes $\mathbf{V}$ in the graph correspond to the variables $\mathbf{X}$, and the edges $\mathbf{E}$ represent the parent-child relationships given in equation 2.2, directing from a parent node to the corresponding child. Thereby, in graphical models the terms node and variable can be used interchangeable. One important property of the graphical DAG representation is that the graph comprises the same assertions of conditional (in)dependence as given in equation 2.2. Thus, the conditional independencies can be defined either by a factorization or by a DAG, while each representation can be transformed into the other one without any change in the independence assumptions. This graphical representation (the DAG) is often referred to as *structure* of a Bayesian network (Hofmann 2000).

Besides the qualitative structure, a Bayesian network encodes the joint probability distribution as given in equation 2.3. Thus, for each variable X_i in the Bayesian network the conditional probability distribution $p(X_i \mid \mathbf{Pa}_i)$ is defined by parameters θ_i. Depending on the type of the BN, θ can be the parameters for a decision tree, a Gaussian distribution, a non-parametric density estimator, or simply a conditional probability table (CPT). In this thesis, we focus on a multinomial model and thus we use CPTs to represent the probability distribution. If a multinomial variable X_i has r_i states, and the parents $\mathbf{Pa}_i$ are in state j, the probability distribution is given by

$$p(X_i = k \mid \mathbf{Pa}_i = j) = \theta_{ijk}, k = 1, ..., r_i \tag{2.4}$$

with $\theta_{ij} = \theta_{ij1}, ..., \theta_{ijr_i}$, where θ_{ij1} is usually omitted because it is given by $1 - \sum_{k=2}^{r_i} \theta_{ijk}$. The parameters of a variable X_i are defined as $\theta_i = \{\theta_{i1}, ..., \theta_{iq_i}\}$ with q_i as the number of different states $\mathbf{Pa}_i$ can assume.

X_1	-
true	0.6
false	0.4



	X_2	
X_1	true	false
true	0.8	0.2
false	0.3	0.7

Figure 2.1: DAG and conditional probability tables for a BN representing $p(X_1, X_2) = p(X_1)p(X_2 \mid X_1)$

Figure 2.1 shows an example of a BN with two boolean variables X_1 and X_2, where X_2 depends on X_1. The probability distributions of both variables are defined by conditional probability tables. Each line of the tables contains the parameters denoted as θ_{ij}, given a state of the parents. If no parent exists (like for variable X_1), the number of different parent states is one.

Let $\Theta_{\mathcal{G}}$ be the set of the parameters for all local conditional probability distributions $\Theta_{\mathcal{G}} = \{\theta_1, ..., \theta_n\}$ of the graph structure $\mathcal{G}$. A Bayesian network is then defined as

$$BN = (\mathcal{G}, \Theta_{\mathcal{G}}) \tag{2.5}$$

and determines the probability distribution among the set of variables in $\mathcal{G}$ based upon equation 2.3.

In figure 2.2, the structure of a small Bayesian network with three variables X_1, X_2, and X_3 is shown. According to the graph structure, the joint probability distribution $p(X_1, X_2, X_3)$ factorises into $p(X_1)p(X_2)p(X_3 \mid X_1, X_2)$.

While every factorization of $p(X_1, ..., X_n)$ (equation 2.3) can be represented as a directed acyclic graph, the numbering of the variables (often called *ordering*) can have a high impact on the resulting graph structure. Different orderings usually yield to different parent sets ($\mathbf{Pa}_i$), which lead to different structures of the Bayesian network. Depending on the chosen ordering, the complexity of the network structure can also vary a lot. We follow the

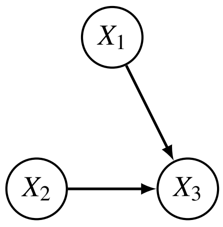


Figure 2.2: DAG for the factorization $p(X_1, X_2, X_3) = p(X_1)p(X_2)p(X_3 \mid X_1, X_2)$

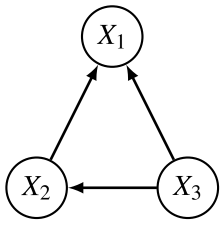


Figure 2.3: DAG for the factorization $p(X_3, X_2, X_1) = p(X_3)p(X_2 \mid X_3)p(X_1 \mid X_3, X_2)$

example of Hofmann (2000) and show the effect of the ordering on the network structure based on this example network with three nodes.

For the network shown in figure 2.2, there are two possbile orderings: X_1, X_2, X_3 or X_2, X_1, X_3. Since there is no edge between X_1 and X_2, the structure does not discriminate between both orderings (Hofmann 2000). The factorization in both cases leads to

$$p(X_1, X_2, X_3) = p(X_1)p(X_2)p(X_3 \mid X_1, X_2)$$

and

$$p(X_2, X_1, X_3) = p(X_2)p(X_1)p(X_3 \mid X_2, X_1),$$

which encodes exactly the same probability distribution and graph structure. This small example can be motivated by the following generative process: at first, values for the random variables X_1 and X_2 are chosen independently; after that, depending on both variables, the value for X_3 is selected (Hofmann 2000).

Now, let us assume the following ordering: X_3, X_2, X_1. Then the joint probability distribution factorizes to

$$p(X_3, X_2, X_1) = p(X_3)p(X_2 \mid X_3)p(X_1 \mid X_3, X_2).$$

Despite the other orderings shown before, in this case it is not possible to make a further simplification.

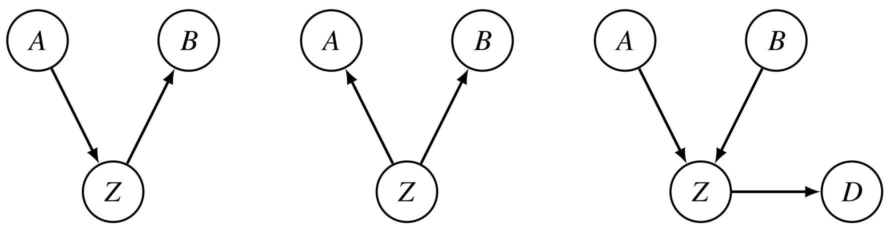
The resulting graph for this factorization is shown in figure 2.3. As it can be seen, the graph is fully connected, unable to represent the independence of X_1 and X_2.

2.1.2 Independence

This section describes the second way to define Bayesian networks. The definition of the Bayesian network given in the previous section is based upon the factorization of the joint probability distribution to a set of conditional probability distributions. As seen before, this factorization implicitly leads to a set of dependencies and independencies defined by the structure of the DAG. Not surprisingly, the meaning of the graph structure of a Bayesian network can be defined by a set of conditional (in)dependence statements, as well. Thereby, two sets of variables are independent given another set of variables if the two sets are separated in the graph after the *d-separation* criterion, where 'd' stands for directed (Geiger, Verma & Pearl 1990). For a short notation of independence, the symbol $\perp\!\!\!\perp$ is used: $A \perp\!\!\!\perp B$ means that variable A is independent of variable B, while $A \perp\!\!\!\perp B \mid Z$ means that variable A is independent of variable B if variable Z is given. In return, $\not\perp\!\!\!\perp$ is the symbol for dependence. $A \not\perp\!\!\!\perp B$ means that A and B are not independent, while $A \not\perp\!\!\!\perp B \mid Z$ means that variable A is not independent of variable B if variable Z is given.

Generally speaking, if two variables are d-separated relative to a set of variables $\mathbf{C}$ in a directed graph, then they are conditionally independent on $\mathbf{C}$ in all probability distributions such a graph can represent.

Before we define the d-separation criterion formally, we have to introduce some terms. A *path* between two variables X_i and X_j is a sequence of edges that connects both variables, independently of the direction of the edges. A variable Z is said to have *serial* edges if either the preceding or successive variable along the path is the parent of Z, and the other variable is the child ($A \rightarrow Z \rightarrow B$ or $A \leftarrow Z \leftarrow B$). A small example is given in figure 2.4(a): The edges that connect Z with A and B are serial. A variable Z has *diverging* edges if the preceding and successive variable along the path are children of Z ($A \leftarrow Z \rightarrow B$). An example is given in figure 2.4(b). Similarly, a variable Z is said to have *converging* edges if both connected variables are parents of Z ($A \rightarrow Z \leftarrow B$, see figure 2.4(c)). These examples are based upon (Hofmann 2000). With these terms, the d-separation criterion



(a) Serial edges: $A \perp\!\!\!\perp B \mid Z$

(b) Diverging edges: $A \perp\!\!\!\perp B \mid Z$

(c) Converging edges ($A \rightarrow Z \leftarrow B$): $A \perp\!\!\!\perp B \mid \emptyset$

Figure 2.4: Illustration of d-separation. The figure on the left-hand side (2.4(a)) shows edges with serial directions. A and B are d-separated given Z ($A \perp\!\!\!\perp B \mid Z$). The middle graph shows diverging edges ($A \perp\!\!\!\perp B \mid Z$). The graph on the right-hand side has converging edges ($A \rightarrow Z \leftarrow B$) and A and B are only d-separated if the state of both variables Z and D is not observed ($A \perp\!\!\!\perp B \mid \emptyset$). However, if the state of Z or D is observed, both variables are no longer rendered d-separated ($A \not\!\perp\!\!\!\perp B \mid Z$) or $A \not\!\perp\!\!\!\perp B \mid D$).

can be formally defined (Geiger et al. 1990):

Definition 2.1 (d-separation) *If $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ is a directed acyclic graph with two disjoint subsets of nodes $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{A}$ and $\mathbf{B}$ are* d-separated *by $\mathbf{C} \subseteq \mathbf{X} \setminus (\mathbf{A} \cup \mathbf{B})$ in $\mathcal{G}$ if and only if every undirected path that connects a node $A \in \mathbf{A}$ with a node $B \in \mathbf{B}$ satisfies at least one of the conditions:*

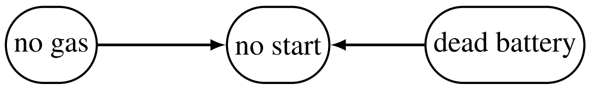
1. *The path contains an intermediate node $Z \in \mathbf{C}$ with serial or diverging edges (not converging).*
2. *The path contains an intermediate node Z with converging edges and neither Z nor the descendants of Z are in $\mathbf{C}$.*

A path is said to be *inactive* or *blocked* if at least one of the two conditions is fulfilled, otherwise the path is said to be *active*. Thus, two variables are blocked according to the d-separation criterion if every path between both variables is blocked. The criterion implies that, besides the presence of edges, also their orientation plays an important role to render variables d-separated. $\mathbf{A}$ and $\mathbf{B}$ are said to be *d-connected* by $\mathbf{C}$ if and only if they are not d-separated by $\mathbf{C}$ in $\mathcal{G}$.

The d-separation criterion, especially the second condition, can be better understood if the directed edges are interpreted as causal effects that naturally imply statistical dependencies. It can be easily seen that the statistical dependencies and independencies that are originated from the causal effects are equivalent to the d-separation criterion. While the d-separation criterion defines conditional independencies, the following examples base on dependencies since they are easier to understand.

- The first condition can be separated into two different cases: (1) the intermediate variable Z has serial edges, or (2) Z has diverging edges. The first case implies that two variables are dependent if one of the variables (A) influences an intermediate variable Z and this intermediate variable influences the other variable (B) while the state of the intermediate variable Z is not known (see figure 2.4(a) for an example) (Hofmann 2000). The second case states that two variables A and B are dependent if both variables are influenced by the same variable Z while the state of Z is not known (see figure 2.4(b) for an example) (Hofmann 2000).

- The second condition is more complicated. It states that two variables A and B are dependent if both variables influence a third variable Z and the state of this variable is known (figure 2.4(c)). This, in the first moment maybe incomprehensible condition, can be explained by the following example (Pearl 1988): Let us assume that there are two (and only two) independent cases of a car refusing to start (”no start“): having no gas (”no gas“) and having a dead battery (”dead battery“). The causal relationships can be represented as a DAG:

  Apparently, the variables ”no gas“ and ”no battery“ are independent from each other in the DAG. Knowledge about the state of one of these variables does not change the knowledge of the other one. E.g., knowing that the battery is charged does not change the knowledge about the gas. However, knowing that the battery is charged and the car does not start implies that the gas tank must be empty. Equally, if the gas tank is full and the car does not start, it is sure that the battery is not charged.

  Thus, the knowledge about a common effect renders two originally independent causes dependent. This effect is referred to as *explaining away*. With a similar consideration it can be shown that even the knowledge about a child of a common effect can render two causes dependent. This is graphically shown in figure 2.4(c).

The independent variables A and B are independent if the states of variables Z and D are unknown. However, if the state of Z or its descendant (child) D is known, A and B are no longer independent from each other.

It has been shown by Verma & Pearl (1990) that the definition of Bayesian networks based on d-separation is equivalent to the definition based on the factorization of the joint probability distribution. Based on a directed acyclic graph, the independence relations defined by the d-separation are preserved in any probability distribution that can be represented as a Bayesian network with the DAG as structure. However, a specific probability distribution may entail additional independence statements that are not encoded in the graph structure. On the other hand, dependencies encoded in the graph structure by means of the d-separation criterion must be also represented in the probability distribution.

For undirected graphical models (e.g. Markov random fields), a concept similar to d-separation as applied to directed acyclic graphs has been introduced. In an undirected graph, two sets of variables $\mathbf{A}$ and $\mathbf{B}$ are u-separated ('u' for undirected) by a third set $\mathbf{C}$ if each undirected path between a variable in $\mathbf{A}$ and a variable in $\mathbf{B}$ contains a variable in $\mathbf{C}$ (Castillo, Gutiérrez & Hadi 1996). Similarly, c-separation has been introduced for chain graphs (Bouckaert & Studený 1998). Chain graphs can contain both directed and undirected edges, and can represent directed acyclic graphs as well as undirected graphs as borderline cases (Lauritzen & Wermuth 1984, Lauritzen & Wermuth 1998).

A concept that is closely related to the concepts of conditional dependencies and independencies introduced so far is the *Markov blanket* of a variable (Pearl 1988):

Definition 2.2 *The Markov Blanket $MB(X_i)$ of a variable X_i is a set of variables of a domain $\mathbf{X}$ such that for any $X_j \in \mathbf{X} \setminus (MB(X_i) \cup \{X_i\})$: X_i is independent of X_j given $MB(X_i)$, i.e. $X_i \perp\!\!\!\perp X_j \mid MB(X_i)$.*

In other words, this definition says that a Markov Blanket $MB(X_i)$ completely shields a variable X_i from any other variable in $\mathbf{X}$. The minimal Markov Blanket is called *Markov Boundary* of X_i (Pearl 1988). Henceforth, when we refer to the Markov Blanket in this thesis, we actually always mean the Markov Boundary.

In Bayesian networks, the Markov Blanket of a variable can easily be determined according to the graph structure of the Bayesian network. A variable's Markov Blanket consists of the variable's parents, children and the parents of the children. A small example is given in figure 2.5.

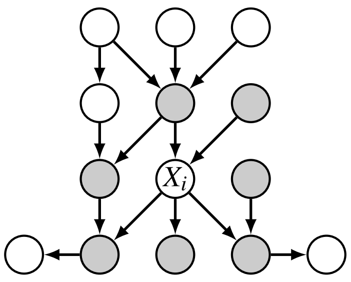


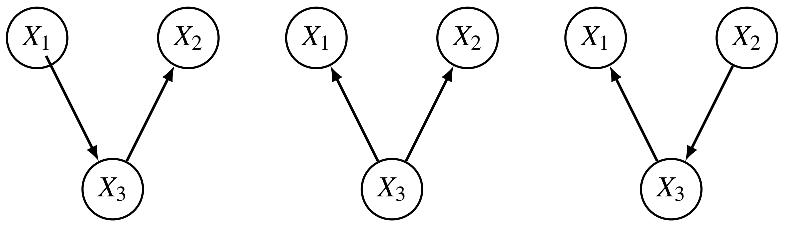
Figure 2.5: Illustration of the Markov Blanket definition. The variables that belong to the Markov Blanket of X_i are shown as shaded. These variables render X_i independent from all other variables (filled with color white).

Another independence statement, mentioned already before, can be made by the local Markov property: each variable is conditionally independent of its non-descendants given its parent variables.

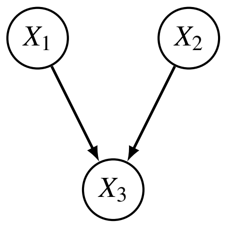
2.2 Equivalence and Causality

The structure of a Bayesian network uniquely determines a set of probability distributions that encode conditional dependencies and independencies according to the d-separation (and d-connection) criterion. However, the contrary case is not uniquely defined. This means that for any given probability distribution $p(\mathbf{X})$ there might be several different graph structures (defining the same conditional dependencies and independencies) that all encode the same probability distribution. This can be seen in a small example (Hofmann 2000): Let $p(X_1, X_2)$ be a probability distribution. This probability distribution can be factorized to $p(X_1, X_2) = p(X_1)p(X_2 \mid X_1)$. The dependencies encoded in the factorization can be represented as a graph with a directed arc from X_1 to X_2 ($X_1 \rightarrow X_2$). However, the probability distribution can be factorized to $p(X_1, X_2) = p(X_2, X_1) = p(X_2)p(X_1 \mid X_2)$, as well, and the conditional dependencies of this factorization lead to a different graphical representation with a directed arc from X_2 to X_1 ($X_1 \leftarrow X_2$). Since both graphs can represent the same set of probability distributions, the graphs are called *equivalent*.

In figure 2.6 an example with three variables X_1, X_2 and X_3 is shown. The example is based upon an example shown in Hofmann (2000). For the network shown on the left-hand



(a) Three equivalent Bayesian network structures



(b) Network that is not equivalent to those shown in figure 2.6(a)

Figure 2.6: Illustration of the equivalence of Bayesian network structures. The equivalent networks in subfigure 2.6(a) all represent the same following d-separations and d-connections: $X_1 \not\perp\!\!\!\perp X_2 \mid \emptyset$ and $X_1 \perp\!\!\!\perp X_2 \mid X_3$.

side, the joint probability distribution can be factorized:

$$p(X_1, X_2, X_3) = p(X_1)p(X_3 \mid X_1)p(X_2 \mid X_3)$$

with the rule $p(X_1)p(X_3 \mid X_1) = p(X_1, X_3) = p(X_3)p(X_1 \mid X_3)$, the factorization can be modified to:

$$p(X_1, X_2, X_3) = p(X_3)p(X_1 \mid X_3)p(X_2 \mid X_3)$$

The corresponding structure is shown in the middle subfigure of figure 2.6(a). Similarly, with the rule $p(X_3)p(X_2 \mid X_3) = p(X_2, X_3) = p(X_2)p(X_3 \mid X_2)$ the previous equation can be transformed to

$$p(X_1, X_2, X_3) = p(X_2)p(X_1 \mid X_3)p(X_3 \mid X_2)$$

The corresponding structure is shown in the right-hand side of figure 2.6(a). Apparently, all three factorizations, and thus all three directed acyclic graphs, represent exactly the same probability distribution. The equivalence of these graph structures can be seen by using the d-separation criterion, as well. According to the d-separation criterion, X_1 and X_2 are d-separated by X_3 ($X_1 \perp\!\!\!\perp X_2 \mid X_3$), as well as X_1 and X_2 are d-connected given the empty set ($X_1 \not\perp\!\!\!\perp X_2 \mid \emptyset$) (Hofmann 2000).

A graph structure that is not equivalent to these networks is shown in figure 2.6(b). Obviously, set of d-separations and d-connections is different from the other networks: According to the second condition of definition 2.1, X_1 and X_2 are d-separated by the empty set ($X_1 \perp\!\!\!\perp X_2 \mid \emptyset$), but they are d-connected given X_3 ($X_1 \not\perp\!\!\!\perp X_2 \mid X_3$). The difference between these networks that render them in-equivalent is the kind of how variable X_3 is connected to the other variables X_1 and X_2. While in the first three networks the connections are serial or divergent (first condition of d-separation), variable X_3 has converging edges in the last network (second condition of d-separation).

While the term *equivalence* of two directed acyclic graphs was introduced before rather informally, we here define equivalence in a more formal fashion. Two Bayesian networks are *equivalent* if the represented probability distribution of one network is equal to the distribution of the other network (Chickering 1995).

Definition 2.3 (Equivalence) *Two directed acyclic graphs $\mathcal{G}$ and $\mathcal{G}'$ are* equivalent *iff for every Bayesian network $BN = (\mathcal{G}, \Theta_{\mathcal{G}})$ with graph structure $\mathcal{G}$ and parameters for the conditional probability distributions $\Theta_{\mathcal{G}}$ there exists a Bayesian network $BN' = (\mathcal{G}', \Theta_{\mathcal{G}'})$ with structure $\mathcal{G}'$ and parameters $\Theta_{\mathcal{G}'}$, such that BN and BN' define the same probability* *distribution.*

Two Bayesian networks are commonly referred to as equivalent if, in fact, their DAGs are equivalent. Verma and Pearl have shown that the equivalence of two Bayesian networks can be decided by comparing their directed acyclic graph structures (Verma & Pearl 1990).

Therefore the following two definitions are needed. The *skeleton* of a DAG is the undirected graph which results if every directed edge in the DAG is transformed to an undirected edge, ignoring the edge direction. A local structure with three variables X_1, X_2 and X_3 is called *collider* or *v-structure* if two arcs connecting the variables converge in variable X_3 ($X_1 \rightarrow X_3 \leftarrow X_2$), and X_1 and X_2 are not connected.

Theorem 2.1 (Verma and Pearl, 1990) *Two directed acyclic graphs are equivalent if and only if they have the same skeleton and the same colliders.*

That means that all equivalent Bayesian networks have a unique graph structure if the direction of the edges is not taken into account. Since the equivalence relation, which is often referred to as *Markov equivalence* relation, is symmetric, reflexive and transitive, the relation defines a set of *equivalence classes* over directed acyclic graphs. One equivalence class comprises all graph structures that are equivalent. Each directed edge that appears in all DAGs of one equivalence class is called *compelled* edge. More formally, a directed edge $X_i \rightarrow X_j \in \mathbf{E}_{\mathcal{G}}$ is *compelled* in $\mathcal{G}$ if for every DAG $\mathcal{G}'$ that is equivalent to $\mathcal{G}$, this edge also appears in $\mathcal{G}'$ ($X_i \rightarrow X_j \in \mathbf{E}_{\mathcal{G}'}$). Any edge that is not compelled in $\mathcal{G}$ is called *reversible*. This means that there exists a DAG $\mathcal{G}'$ equivalent to $\mathcal{G}$ in which this edge has the opposite direction.

It has been shown in Chickering (1995) that a directed acyclic graph $\mathcal{G}$ can be transformed into any equivalent network structure $\mathcal{G}'$ by iteratively reverting *covered* edges. An edge $X_i \rightarrow X_j$ is said to be *covered* in $\mathcal{G}$ if X_i and X_j have identical parents in $\mathcal{G}$ while, of course, X_i can not be the parent of itself.

Markov equivalence based on graph properties has also been introduced for other graphical models. E.g., any DAG with latent variables can be represented by an ancestral graph that encodes the same conditional independence relations entailed by the DAG (Richardson & Spirtes 2002), providing a finite search space of latent variable models (Spirtes, Richardson & Meek 1997). Ali, Richardson & Spirtes (2004) found sufficient graphical conditions that render two ancestral graphs Markov equivalent, while in Zhang & Spirtes (2005) a transformational characterisation, comparable to the one given for DAGs by Chickering (1995), has been introduced.

For the class of chain graphs, Frydenberg (1990) introduced graph-theoretic criteria for Markov equivalence concerning a restricted class of probability distributions, and Andersson, Madigan & Perlman (1997) introduced criteria for the general case without restricting to a certain class of probability distributions.

Graphical Representation of Equivalence Classes

In the previous section we have introduced criteria for the graphical equivalence of two directed acyclic graphs. As mentioned before, the equivalence relation partitions the space of directed acyclic graphs into equivalence classes of (graphically) equivalent Bayesian network structures.

Based on theorem 2.1 a graphical representation of equivalent Bayesian network structures can be defined. All equivalent Bayesian network structures can be represented by a so called *partially directed acyclic graph* (PDAG). A PDAG can contain both directed and undirected edges. A directed edge in the PDAG represents an edge that is directed (compelled) in all Bayesian network structures contained in the equivalence class. All reversible edges are represented as undirected edges in the PDAG.

That means if there is an edge that has no direction in the PDAG, there are at least two Bayesian networks that have this edge in common, but the edge differs in its direction. On the other hand it does not mean that directions of reversible edges can be combined arbitrarily. First, the graph of a Bayesian network must be always acyclic. Second, if the direction of an edge is chosen in a way that a new collider is generated, the Bayesian network would no longer belong to the same equivalence class. Hence, inserted directions must not create new colliders.

In this work we use the term PDAG as used by Chickering (1995). However, in the literature, different terms for the graphical representation of equivalence classes have been established. The PDAG representation was first introduced by Verma & Pearl (1990) as the *(completed) pattern* associated with a DAG. Andersson, Madigan & Madigan (1997) introduced the term *essential arrows* for the term *compelled edges* used by Chickering (1995). Thus, they use the term *essential graph* instead of PDAG.

Based on a given directed acyclic graph structure, several algorithms have been developed in order to obtain the corresponding PDAG representing the equivalence class this DAG belongs to. It is worth to mention that it is sufficient to direct only those edges in the PDAG that are members of a collider structure in the underlying DAG (Chickering 1995).

However, other edges might be compelled (and not reversal) as well since a reversal of these edges would cause a new collider structure, and thus the corresponding DAG would belong to another equivalence class. If the collider edges are the only edges that are directed in the PDAG, the graph is called a *minimal PDAG representation*. On the other hand, if every compelled edge is represented as a directed edge in the PDAG and every undirected edge corresponds to a reversible edge, the PDAG is called a *completed PDAG representation*.

There are several algorithms to identify compelled edges in a directed acyclic graph. Usually, the algorithms start with a minimal PDAG and try to identify remaining undirected, but compelled edges that do not participate in any collider structure. If a compelled edge is found, it is directed in the PDAG and the search is repeated until no more compelled edges can be identified. An algorithm that is known to be sound but not complete was introduced in Verma & Pearl (1992). Since not all compelled edges can be identified with this algorithm, the resulting PDAG might not be completed. However, every edge that is directed in the resulting PDAG is compelled. Meek (1995a) and Andersson, Madigan & Madigan (1997) have introduced sound and complete polynomial-time algorithms to derive completed PDAG representations, another algorithm was introduced in Dor & Tarsi (1992). An algorithm that can take both DAG as well as a minimal PDAG as input to derive the completed PDAG with polynomial complexity has been introduced in Chickering (1995).

Causal networks

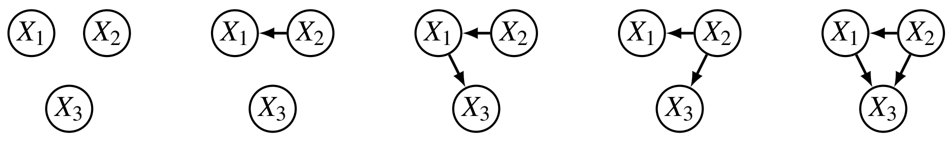
Above, for the explanation of the d-separation criterion, we have shown a small example network with three variables: ”no gas“, ”dead battery“ and ”no start“, and the corresponding network structure is: ”no gas“ $\rightarrow$ ”no start“ $\leftarrow$ ”dead battery“. In this example it is obvious that the edges represent causal relationships besides the formal conditional independencies defined by the d-separation criterion. ”no gas“ and ”dead battery“ are causes that prevents a car being started (”no start“ is the effect of both causes). Such types of Bayesian networks that describe causal dependencies among random variables are called *causal networks* (Pearl 1988, Pearl & Verma 1991). In a causal network, each directed edge represents a causal influence from the variable at the tail of the edge to the variable at the head of the edge.

Usually, Bayesian networks can not be interpreted as causal networks. Instead, the structure of a BN with its directed edges represent the factorization of the joint probability distribution (or, equivalently, a set of conditional (in)dependence statements). The (in)dependence statements result from a causal interpretation of the network, however, the opposite is usually not true: the causal interpretation of edges does not result from (in)dependence statements given by the d-separation criterion. This is apparent since d-separations (and d-connections) do not define a directed acyclic graph but only an equivalence class of equivalent Bayesian network structures. This means that at least edges that are reversable in this equivalence class can not be treated as causal influences, if the network is build upon (in)dependence statements.

2.3 Faithfulness

In section 2.1.2 we defined the meaning of the Bayesian network structure based on the d-separation criterion, while we previously have introduced Bayesian networks by means of the factorization of a probability distribution. Conditional dependencies and independencies that are defined by the d-separation criterion must be enclosed in the probability distribution. However, the probability distribution may comprise additional independencies that are not predetermined by the structure of the BN, but that are encoded in some suitably chosen parameters for the conditional probability distributions $p(X_i \mid \mathbf{Pa}_i)$. Some factors of the factorized probability distribution could be further simplified, or the joint probability could be factorized in different ways (Hofmann 2000).

If learning BNs from data, the following question arises: Is there a BN structure describing all the dependence and independence relations encoded in the data? Unfortunately, this question must be negated, which can be seen in the following example (see (Steck 2001) for more details): Assume that a probability distribution with three variables X_1, X_2, and X_3 comprises following conditional independencies: $X_1 \perp\!\!\!\perp X_3 \mid X_2$ and $X_2 \perp\!\!\!\perp X_3 \mid X_1$. In other words, X_1 and X_3 are conditionally independent if X_2 is known, and X_2 and X_3 are conditionally independent if X_1 is known. All other associations are dependencies (see figure 2.7). Obviously, there is no DAG that comprises all dependencies and independencies: Either, the DAG does not encode all dependencies (DAGs shown in figures 2.7(a) and 2.7(b)), or some independencies are missing (figures 2.7(c), 2.7(d) and 2.7(e)). A graph that encodes at least all independencies of the probability distribution is called D-map. On the other side, this means that every dependency encoded in the graph must be represented in the probability distribution. A maximal D-map is a D-map where no edge can be added



(a) D-map (b) maximal D-map (c) minimal I-map (d) minimal I-map (e) I-map

Figure 2.7: Possible DAG representations of following conditional dependencies and independencies:
Independencies: $X_1 \perp\!\!\!\perp X_3 \mid X_2$, $X_2 \perp\!\!\!\perp X_3 \mid X_1$
Dependencies: $X_1 \not\perp\!\!\!\perp X_2$, $X_1 \not\perp\!\!\!\perp X_3$, $X_2 \not\perp\!\!\!\perp X_3$, $X_1 \not\perp\!\!\!\perp X_2 \mid X_3$

without destroying the D-map property. In contrast to D-maps, in I-maps each dependency encoded in the probability distribution must be represented in the graph. This also means that every independence encoded in the graph must exist in the probability distribution, as well. A minimal I-map is an I-map from which none of the edges can be removed. P-maps are graphs that are both I-maps and D-maps. For the example in figure 2.7 exists no P-map since there exists no D-map that is also an I-map: The maximal D-map (figure 2.7(b)) has one dependency less than the minimal I-map (figure 2.7(c)).

Additional independencies that are not encoded in the structure of a Bayesian network but only in the probability distribution defined by a data set can complicate the learning of a Bayesian netwok. Many algorithms, especially constraint-based learning algorithms which are explained later in this chapter, usually assume the faithfulness condition (Pearl 1988):

Definition 2.4 (Faithfulness) *A DAG $\mathcal{G}$ and probability distribution p satisfy the faithfulness condition if the Markov condition entails all and only the conditional independencies in p.*

Meek (1995b) shows that the assumption of faithfulness is reasonable and the probability distributions that are un-faithful have measure zero.

3 Learning Bayesian Networks

Learning Bayesian networks refers to the process of extracting a Bayesian network from a data set $\mathbf{D}$ by means of unsupervised learning.

Bayesian network learning can be divided into two problems: parameter learning and structure learning. In the first case the structure is already known or assumed known and the parameters $\Theta_{\mathcal{G}}$ of the local conditional probability distributions of the Bayesian network $\mathcal{G}$ are learned from data. The second case, learning the structure, is much harder, since both, structure $\mathcal{G}$ and parameters $\Theta_{\mathcal{G}}$, must be learned.

Learning the structure and the parameters of a Bayesian network can be formulated as the following problem: Given a finite data set $\mathbf{D} = (\mathbf{d}^1, ..., \mathbf{d}^N)$ with N different independent observations, where each data point $\mathbf{d}^l = (d_1^l, ..., d_n^l)$ is an observation of all n variables, find the graph structure $\mathcal{G}$ and the parameters $\Theta_{\mathcal{G}}$ that best match data set $\mathbf{D}$ (Nägele et al. 2008). In this thesis we assume that all variables are observed and no observation d_i^l is missing, meaning that the data set $\mathbf{D}$ is complete. There are also algorithms to learn from incomplete data. For instance, the parameters can be learned by an *E*xpectation *M*aximization (EM) algorithm even if some observations are missing. Friedman (1998) introduced SEM (Structural Expectation Maximization) as an algorithm to learn the structure of a BN from incomplete data. Besides complete data we also assume that data are nominal. There are other types of data that are conceivable: For instance, variables could be continuous. Hofmann (2000) shows how BN can be learned in domains where nominal and continuous variables appear together.

Generally there are two methods how data can be interpreted, the Bayesian and the frequentist method. In a frequentist's point of view, data are generated from one model. That means that the model is real, but unknown. A common learning approach of frequentist would be the maximum likelihood (ML) estimation. In contrast, the Bayesian point of view is that the data is real, but not the model. All models are generally possible, each model has an own degree of belief. To calculate the prediction for an event, all models have to be considered. There are several advantages of the Bayesian point of view: First, the interpretation of the Bayesian approach is more intuitive: Having a fixed data set and a distribution for all models is easier to understand and closer to the learning problem than having just one fixed model and a standard error. Second, Bayesian statistics always have a prior and a posterior: The prior distribution forms together with the observed data the posterior distribution. This enables the incorporation of prior knowledge in a quite intuitive way. However, there are also some drawbacks: To calculate the prediction for an event, all models have to be considered. This can be computationally intractable, or at least very expensive. For some prior distributions, the posterior distribution can be solved in closed form which renders the calculation computationally tractable. The second problem also arises from the prior distribution: The parameters of the prior distribution must be set even if no prior knowledge is available. In the literature it is quite common to use the Bayesian point of view for learning Bayesian networks. Thus, we focus on this view in this thesis.

3.1 Parameter Learning

At first it is shown how to learn the local probability distributions of a Bayesian network given a data set **D**. The methods are capable to combine data and prior knowledge (for example expert knowledge) to produce an optimal estimate.

Let k be the state of variable X_i and j the state of the parents $\mathbf{Pa}_i$ of variable X_i. N_{ijk} is the number of cases in the data set **D** in which the parents of X_i are in state j and X_i itself is in state k, and r_i is the number of states the variable X_i can assume. The incorporation of prior knowledge can be efficiently done by setting the prior variables α_{ijk}. One can imagine that α_{ijk} is defined like N_{ijk}, but not on the data set **D** but on an imaginary data set containing the prior knowledge. The size of this imaginary data set is called *equivalent sample size*. This size is used to weight the proportion between prior knowledge and data. If no prior knowledge is available and the parameters should be learned only from data a noninformative prior can be used, and α_{ijk} is reduced to a uniformly distributed pseudo count.

With $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$ and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$, the probability of a variable in state k given the parent state j, the data set **D** and the graph structure $\mathcal{G}$ can be easily assessed by

$$p(X_i = k \mid \mathbf{Pa}_i = j, \mathbf{D}, \mathcal{G}) = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}}. \tag{3.1}$$

Since the parameter θ_{ijk} and $p(X_i = k \mid \mathbf{Pa}_i = j, \mathbf{D}, \mathcal{G})$ are equivalent (see 2.4), 3.1 can be used to learn the parameters $\Theta_{\mathcal{G}}$ of the BN with structure $\mathcal{G}$. For a more detailed deduction of 3.1, we refer the interested reader to Heckerman et al. (1995).

3.2 Structure Learning

In many domains, the structure of a Bayesian network is given and the learning process focuses on the parameter estimation. For instance, a Naive Bayes classifier is a simple Bayesian network with fixed structure: The class variable representing the variable to be estimated (e.g. Cancer: yes / no) is the parent of all observed variables, representing the observed conditions (e.g. blood test: positive / negative). The task here is to learn the parameters of the local probability distributions in order to calculate the probability of the class given the observed conditions.

In many other cases, however, the structure of a BN is unknown and must be reconstructed from data. But learning the structure of a Bayesian network is a much more challenging task than learning the parameters: The number of possible different network structures growths super-exponentially with the number of variables. Chickering et al. (1994) has proven that structure learning of BNs is generally an $\mathcal{NP}$-hard problem. The extreme amount of different networks can be seen by an example: For a network with only 10 variables, there are almost 10^{20} different possible BN structures, which prohibits an exhaustive search over all possible network structures (Hofmann 2000). Thus, it became a common approach to apply heuristic strategies to create a BN network structure.

Two general types of learning algorithms have been developed over the last two decades: The first, known as *score-based* algorithms, utilize a scoring function to guide heuristic search strategies. The goal of these algorithms is to find the network with the best score. This is typically achieved by applying local changes to the network like edge addition, edge removal or edge reversal until a high-scoring network is found. The underlying concept of these algorithms is based on the factorization of the probability distribution (see equation 2.3) which is utilized by the scoring functions. Constraint-based algorithms form the second class of BN structure learning algorithms and rely on the BN definition based on independence relationships (see section 2.1.2). These algorithms perform statistical tests to determine dependencies and independencies among the variables in order to reconstruct the BN structure.

In general, both approaches have advantages and disadvantages (Dash & Druzdzel 1999). Both have the ability to incorporate prior knowledge about the network structure. However, with an appropriate scoring function one can even integrate the knowledge about the structure and parameters in form of prior probabilities. Usually, constraint-based methods are relatively quick and outperform score-based approaches in terms of runtime. However, constraint-based approaches identify dependencies based on independence tests with a specific significance level. A wrongly detected dependency can lead to cascading effects leading to a corrupted network structure. Score-based approaches can avoid such cascading effects since they usually allow to remove an arc later on if it was added in a previous phase of the learning.

The work presented in this thesis is based on both approaches (score-based and constraint-based), thus a short overview of both techniques is given in the following sections.

3.2.1 Score-based Approach

The learning algorithms used in this work are restricted to the case of fully observed data. For data with missing values the algorithms for fully observed data can be used if missing values are filled as described in Heckerman (1995). Another method, introduced by Friedman (1997) and Friedman (1998), uses an advanced EM, called *structural EM* (SEM), to determine the structure $\mathcal{G}$ and the parameter set $\Theta_{\mathcal{G}}$. From now on we assume fully observed data. The introduction of the score-based approach closely follows the introduction published in Nägele (2005), Nägele et al. (2008) and Pinto et al. (2009).

Scoring function

To rate a structure according to the data set $\mathbf{D}$, a scoring function $S(\mathcal{G} \mid \mathbf{D})$ is used. This function rates the structure $\mathcal{G}$ by assigning a score $S(\mathcal{G} \mid \mathbf{D})$ to it. In this thesis a scoring function is used that is based upon Bayesian statistics. The score of a graph $\mathcal{G}$ given data set $\mathbf{D}$ is (Dejori 2005):

$$S(\mathcal{G} \mid \mathbf{D}) = \frac{p(\mathbf{D} \mid \mathcal{G})p(\mathcal{G})}{p(\mathbf{D})}, \tag{3.2}$$

where $p(\mathcal{G})$ is the prior probability for the graph, $p(\mathbf{D})$ a normalization constant and $p(\mathbf{D} \mid \mathcal{G})$ the marginal likelihood of the data $\mathbf{D}$. The scoring function $S(\mathcal{G} \mid \mathbf{D})$ is a probability distribution and is also called the posterior distribution for the network structures, while $p(\mathcal{G})$ is denoted as the prior.

To determine the posterior distribution, the marginal likelihood has to be calculated:

$$p(\mathbf{D} \mid \mathcal{G}) = \int p(\mathbf{D} \mid \Theta_{\mathcal{G}}, \mathcal{G}) p(\Theta_{\mathcal{G}} \mid \mathcal{G}) d\Theta_{\mathcal{G}}, \tag{3.3}$$

where $p(\mathbf{D} \mid \Theta_{\mathcal{G}}, \mathcal{G})$ is the likelihood of data $\mathbf{D}$ for the Bayesian network $(\mathcal{G}, \Theta_{\mathcal{G}})$, and $p(\Theta_{\mathcal{G}} \mid \mathcal{G})$ denotes the prior for the local probability distributions $\Theta_{\mathcal{G}}$ of the Bayesian network with structure $\mathcal{G}$.

Equation 3.3 can be solved in closed form if some assumptions that are proposed by Cooper & Herskovits (1991) are fulfilled. These are *parameter independence*, *parameter modularity*, which means that the conditional probability distribution of X_i depends only on its parent $\mathbf{Pa}_i$, and *complete data* in combination with *Dirichlet priors*. The scoring function, based upon these assumptions, is called BD (Bayesian Dirichlet) score (Heckerman et al. 1995):

$$p(\mathbf{D} \mid \mathcal{G}) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(a_{ijk})}, \tag{3.4}$$

where q_i is the number of possible parent configurations of X_i and r_i is the number of states of X_i. Γ denotes the gamma function. As mentioned in section 2.2, different Bayesian networks that belong to one equivalence class represent the same underlying probability distribution. Therefore the scoring function should not distinguish between equivalent Bayesian networks. Given a noninformative prior $\alpha_{ijk} = \frac{1}{q_i r_i}$ as proposed by Heckerman et al. (1995) this *score equivalence* is achieved. Equation 3.4 together with the noninformative prior builds the *B*ayesian *D*irichlet score with likelihood *e*quivalence and *u*niform prior (BDeu score).

Since this score is decomposable (Heckerman et al. 1995), it is possible to calculate the score contribution of each variable independently from the score contribution of other variables. With this, the score difference of simple network changes like edge addition or edge removal can be efficiently calculated.

If using the logarithmic value of the score instead of 3.4 directly, the score becomes

$$f_{BDeu}(\mathbf{D} \mid \mathcal{G}) = \log(p(\mathbf{D} \mid \mathcal{G})) = \sum_{i=1}^{n} f_{BDeu}(X_i \mid \mathbf{Pa}_i) \tag{3.5}$$

with

$$f_{BDeu}(X_i \mid \mathbf{Pa}_i) = \log\left(\prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(a_{ijk})}\right). \tag{3.6}$$

Throughout this thesis, we use the BDeu score (to be more precise: its logarithmic value $f_{BDeu}(\mathbf{D} \mid \mathcal{G})$) for our BN learning algorithms since this score is most widely used in the literature. Whenever we refer to the BDeu score, we actually refer to its logarithmic value.

However, there are several other scoring functions to evaluate the fitness of a BN regarding a data set. Another score based on the Bayesian approach is the K2 score (Cooper & Herskovits 1991). A second class is formed by information-theoretic scores like the log-likelihood or the *Bayesian information criterion* which is an approximation score derived by Schwarz (1978). Exactly the same except the algebraic sign is the MDL score based on minimum description length. MIT (mutual information tests) (de Campos 2006) and NML (Roos et al. 2008) do also belong to the class of information-theoretic scores. We do not consider these scoring functions in this thesis since we focus on algorithms for structure learning and not on scoring functions. BDeu is the most widely used score, thus using this score enables a direct comparison to results reported by others in the literature.

Heuristic Search Strategies

The target of a score-based structure learning algorithm is to find the Bayesian network with the best score. Unfortunately, finding the highest-scoring structure is $\mathcal{NP}$-hard (Chickering et al. 1994), and the number of possible network structures grows super-exponential with the number of nodes. Thus, an exhaustive search over all possible network structures is intractable for networks of reasonable size. Some methods were developed to find the optimal network structure even for networks with tens of variables (Tamada, Imoto & Miyano 2011, Yuan, Malone & Wu 2011), but for larger networks one needs heuristic search strategies to find high-scoring networks. The drawback of heuristic search strategies is that the learned network structure might not be the optimal network structure. Some existing algorithms are explained in section 3.3 in more detail.

3.2.2 Constraint-based Approach

Constraint-based algorithms form the second class of BN structure learning algorithms and rely on the BN definition based on independence relationships (see section 2.1.2) (Spirtes et al. 2001, Steck 2001, Cheng 2002). The basic concept behind constraint-based algorithms is the following: For a small set of variables, conditional independence tests are performed in order to obtain a set of conditional dependency and conditional independence relationships among the variables. These tests are repeated until a feasible set of such relationships are extracted from data. Based on the faithfulness assumption (see section 2.3), the structure of the BN including the direction of the edges is built upon the set of conditional dependencies and independencies.

The local independence tests are a big advantage and a disadvantage at the same time: Since it is not necessary to evaluate the complete network structure, but only local dependencies and independencies, local evaluations are sufficient to create the BN network structure. However, putting locally good structures together to form a BN structure might lead to worse network reconstructions compared to the global network analysis of the score-based approach.

Another problem arises with the sparseness of data: The quality of the reconstructed network heavily depends on the quality of the conditional independence tests. The conditional independence tests need a sufficient amount of data to make statements about statistical independence.

Tsamardinos, Brown & Aliferis (2006) introduced a new method to combine both approaches: In the first step, a constraint-based algorithm is used to detect dependencies among the variables to build a skeleton with undirected edges between dependent variables, and a score-based Greedy Hill Climbing search algorithm is used afterwards to learn a structure, but restricted on the edges apparent in the skeleton.

3.3 Structure Learning Algorithms

There are many Bayesian network structure learning algorithms that are published in the literature. These algorithms have typically in common that they either use a scoring function to determine the quality of a candidate network, or use constraint-based techniques to identify the structure by conditional independence tests. A third fraction of algorithms combines both approaches.

In this section we give an overview about existing structure learning algorithms. Whereas the number of existing algorithms is high, we restrict on algorithms that are well-known, commonly used or very competitive regarding the quality and the runtime performance.

The most prominent representative of the class of algorithms using a scoring function is

Greedy Hill Climbing (Heckerman et al. 1995) which searches in the space of DAGs. An algorithm that learns a network in a greedy way, but with fixed ordering of the nodes, is the K2 algorithm (Cooper & Herskovits 1991). Sparse Candidate (SC) (Friedman, Nachman & Pe'er 1999) restricts the number of possible parents in such a way that it is much more efficient than a pure Greedy Hill Climbing. Optimal Reinsertion is an any-time algorithm (Moore & Wong 2003): Iteratively, a node is chosen and the best neighbourhood of this node is determined.

Bio-inspired methods like genetic algorithms were also applied to the problem of Bayesian network structure learning (Larrañaga, Poza, Yurramendi, Murga & Kuijpers 1996, Lee, Chung, Kim & Kim 2010, Ko, Kim & Kan 2011), also Ant Colony Optimization was used to learn the structure (de Campos, Fernandez-Luna, Gamez & Puerta 2002, Pinto et al. 2009). There is even an approach that combines a genetic algorithm with Ant Colony Optimization (Li 2009). Greedy Equivalent Search (Chickering 2003) is a score-based algorithm that does not search in the space of DAGs, but in the space of PDAGs. Others search in the space of orderings (de Campos & Puerta 2001, Chen, Anantha & Lin 2008).

The second class of algorithms uses conditional independence tests to determine the structure of a Bayesian network. The PC algorithm (Spirtes et al. 2001) is the most well-known algorithm that returns a PDAG. Afterwards, the remaining undirected edges must be directed. Three Phase Dependency Analysis (TPDA) (Cheng 2002) is another constraint-based algorithm.

Hybrid methods form the third class of algorithms: Essential Graph Search (EGS) (Dash & Druzdzel 1999) searches in the space of PDAGs by means of constraint-based techniques and uses a score (in one variation combined with Greedy Hill Climbing) to rate the network. An algorithm that builds a network by using constraint-based techniques as well as a score-based search, given an ordering for the variables, is BENEDICT (Acid & Campos 2001). An algorithm that is quite competitive regarding runtime performance and quality is Max-Min-Hill-Climbing (MMHC) (Tsamardinos, Brown & Aliferis 2006). This algorithm uses Max-Min Parents and Children (MMPC) to detect an undirected skeleton with constraint-based methods and applies Greedy Hill Climbing on the set of undirected edges to direct them.

In the following sections we present some Bayesian network structure learning algorithms in more detail. Some of them were already mentioned in this thesis, some of them were not yet presented. These algorithms are either commonly used or shown to be quite competitive regarding performance and/or quality. We show them in more detail since later

Algorithm 1: GS/MMHC Algorithm

```
   Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$, set of all/candidate edges $\mathbf{E}$
   Output: Bayesian network
1  if edges $\mathbf{E}$ not given: $\mathbf{E}$ contains all edges between variables $\mathbf{X}$;
   // initialize with empty graph
2  $\mathcal{G} = (\mathbf{X}, \emptyset)$;
3  $\mathcal{G}_{total}$ = copy of $\mathcal{G}$;
4  $TL$ : tabu list (FIFO) with last 100 structures;
5  repeat
       // actions can be add-edge, remove-edge, revert-edge; only
          try add-edge if edge exists in E; action may not cause
          cycles
6      choose best action for $\mathcal{G}$ with resulting graph not in $TL$;
7      apply best action to $\mathcal{G}$;
8      add $\mathcal{G}$ to $TL$;
9      $\Delta S$: score $\mathcal{G}$ - score $\mathcal{G}_{total}$;
10     if ($\Delta S > 0$) then
11         $\mathcal{G}_{total}$ = copy of $\mathcal{G}$;
12     end
13 until $\mathcal{G}_{total}$ has not changed last 20 times ;
14 return Bayesian network with structure $\mathcal{G}_{total}$
```

we compare our methods directly to these structure learning algorithms.

3.3.1 Greedy Hill Climbing: GS / MMHC

One of the most used score-based algorithms to learn the structure of Bayesian networks is Greedy Hill Climbing (GS). This algorithm is quite simple compared to other algorithms which explains the popularity of this algorithm. The algorithm itself learns a Bayesian network in iterations: At every iteration, the action (add, remove, or revert edge) which leads to the highest score-improvement (but not leading to a cycle in the graph structure) is chosen (line 6, algorithm 1). The benefit of an action is calculated by the score difference between the resulting graph and the graph before applying this action (see equation 3.4). Since the BDeu score is decomposable, the score difference can be efficiently calculated. For each of the local changes *add-edge*, *remove-edge* and *revert-edge*, only the score difference of the variables that have a changed set of parents must be calculated. Usually, the algorithm ends if there is no operation that improves the score. The algorithm is shown in detail in algorithm 1 which is taken from Pinto et al. (2009).

In our implementation of the Greedy Hill Climbing algorithm which is used to benchmark our new methods, we extended the pure Greedy search as suggested in Tsamardinos, Brown & Aliferis (2006) with a TABU search. The algorithm remembers the graph structures that were already evaluated and disallows every action in line 6 that leads to a graph structure which was already evaluated. In our implementation, the last 100 graph structures are remembered. If a new graph structure is chosen in line 7, it is added to the tabu list TL (line 8) and the oldest structure is removed from the list if the size of the tabu list would be larger than 100. In order to escape local maxima, score decreases are allowed if the best action leads to a score decrease. Only after several changes without improving the score, the overall highest-scoring network is returned. We follow Pinto et al. (2009) and use 20 as a number for the number of changes. The authors of MMHC repeated the search for a better structure only 15 times without score improvement (Tsamardinos, Brown & Aliferis 2006).

Tsamardinos, Brown & Aliferis (2006) introduced Max-Min-Hill-Climbing (MMHC) as a combination of constraint-based techniques with a Greedy Hill Climbing search: A so called skeleton of possible edges is constructed with a method called *M*ax-*M*in *P*arents and *C*hildren (MMPC) (see appendix A.1). The edges of the skeleton are given as parameter **E** to the Greedy Hill Climbing algorithm (compare algorithm 1). If Greedy Hill Climbing is applied, but the edges are restriced on those contained in the skeleton produced by MMPC, the algorithm is called MMHC (Tsamardinos, Brown & Aliferis 2006).

3.3.2 Simulated Annealing: SA and MMSA

Simulated Annealing mimics the behaviour of metal that builds large crystals if it is cooled in a controlled way. It was introduced by Kirkpatrick, Gelatt & Vecchi (1983) as a general optimization technique, a more detailed description of SA can be found in Laarhoven & Aarts (1987). The algorithm for learning the structure of a BN learns a network in iterations (Dejori 2005). In every iteration a candidate edge from set of candidate edges **E** is randomly chosen (line 10, algorithm 2). The best local action (add, remove or revert edge) according to the score is selected (line 12) and applied if the score difference ΔS is positive (meaning that the new structure has a better score than the network without the action being applied). If there is no score improvement or even a decrease of the score, the action is applied depending on a value called temperature T. As higher the temperature,

Algorithm 2: SA/MMSA Algorithm

```
   Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$, set of all/candidate edges $\mathbf{E}$
   Output: Bayesian network
1  if edges $\mathbf{E}$ not given: $\mathbf{E}$ contains all edges between variables $\mathbf{X}$;
2  $T_0 = 30$; // start temperature
3  $T_{end} = 0.000005$; // end temperature
4  $\gamma_{SA} = 0.99$; // temperature decrease factor
5  $\beta_{SA} = 20$; // iteration factor
   // initialize with empty graph
6  $\mathcal{G} = (\mathbf{X}, \emptyset)$;
7  $T = T_0$;
8  repeat
9      for $\beta_{SA} \cdot |\mathbf{X}|$ times do
10         choose edge $E_{ij}$ randomly from candidate edges $\mathbf{E}$ ;
           // actions can be add-edge, remove-edge, revert-edge; only
              try add-edge if edge exists in E; action may not cause
              cycles
11         choose best local action for edge;
12         $\Delta S$: score difference for best action (score $\mathcal{G}$ with best action - score $\mathcal{G}$);
13         if ($\Delta S > 0$) or ($exp(\frac{\Delta S}{T}) > rand(0,1)$) then
14             apply best action to $\mathcal{G}$;
15         end
16     end
17     $T = T \cdot \gamma_{SA}$;
18 until $T < T_{end}$ ;
19 return Bayesian network with structure $\mathcal{G}$
```

as higher the probability that score-decreasing actions are applied. The temperature is decreased from time to time (line 17), until it reaches a minimum value. As a result, score decreasing actions become more unlikely as longer the algorithm runs. The algorithm is sketched in algorithm 2 which is taken from Pinto et al. (2009).

To perform comparisons to our new methods, we implemented the algorithm in our software tool. In our implementation, we set the start temperature to 30, decrease the temperature after every iteration by the factor 0.99, and iterate until a minimum temperature of 0.000005 is reached. In one iteration, 20 times number of nodes local changes are tested. These are the same values as used in Pinto et al. (2009).

The algorithm returns a Bayesian network with structure $\mathcal{G}$. Similarly to MMHC, SA applied on the skeleton is called MMSA (see (Pinto et al. 2009) for more details).

3.3.3 Ant Colony Optimization: ACO, MMACO and related algorithms

Ant Colony Optimization is a meta-heuristic that mimics the way how ant colonies search for food sources (Dorigo & Stützle 2004). The ACO algorithm presented here (see algorithm 3) was first published in Pinto et al. (2008) and further investigated in Pinto et al. (2009). The description of the algorithm is taken from Pinto et al. (2009) with minor modifications. We refer to this publication for more information about the ACO algorithm. At the end of this section, some other ACO-based BN structure learning algorithms are introduced. But first we introduce the algorithm of Pinto et al. (2009).

At each iteration of the algorithm, m ants collaboratively try to find a good network structure. When building one network, each of the m ants take a randomly chosen node pair X_i and X_j and assign the edge between the two nodes the state $k = Z(X_i, X_j)$, representing one of the three states $\{i \rightarrow j, i \leftarrow j, i \nleftrightarrow j\}$ (meaning: edge from X_i to X_j, edge from X_j to X_i, no edge between X_i and X_j). These edge states are restricted if the skeleton with candidate edges $\mathbf{E}$ is given: In this case, edge states that are not included in the skeleton have zero probability. The ant with the highest score improvement is taken and the edge state produced by this ant is applied to the network. The network is build until no better network structure is found, meaning that all ants are unable to find an edge improving the score. This procedure is repeated N_{max} iterations and the network with the highest score is returned as result of the ACO algorithm.

To guide the ants to the direction of the best network, each ant chooses the edge state according to the score and so called pheromones. The pheromones represent the knowledge of the ants about previously learned networks. To be more precise, the pheromones assign a value to each state of each possible edge E_{ij}. The pheromones are stored in the pheromone matrix τ.

At the end of each iteration, the pheromones are updated according to the following steps (see (Pinto et al. 2009)):

- Application of evaporation (with ρ as evaporation coefficient)

$$\tau \leftarrow (1 - \rho) \cdot \tau \tag{3.7}$$

- Preserving information about currently learned network $\mathcal{G}$:

Algorithm 3: ACO/MMACO Algorithm

```
Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$, set of all/candidate edges $\mathbf{E}$
Output: Bayesian network
   // Initialization
1  if pheromones $\tau$ not given: initialize each entry of $\tau$ with $\tau_0$;
2  if edges $\mathbf{E}$ not given: $\mathbf{E}$ contains all edges between variables $\mathbf{X}$;
3  define $N_{max}$ as max number of iterations;
4  $N_{iter} = 0$;
5  $\mathcal{G}_{total} = (\mathbf{X}, \emptyset)$;
   // Optimization
6  repeat
       // initialize with empty graph
7      $\mathcal{G} = (\mathbf{X}, \emptyset)$;
8      $\mathcal{G}_{best}$ = copy of $\mathcal{G}$;
       // Edge assignment and orientation
9      while better solutions are found do
           // go over all $m$ ants
10         for $ant = 1$ to $m$ do
11             choose edge $E_{ij}$ from $\mathbf{E}$ randomly;
12             choose edge assignment $k_{ant}$ for $E_{ij}$ according to equation (3.11);
13             $\Delta S_{ant}$: score difference for the assignment;
14         end
15         Find ant $BestAnt$ with highest score benefit $\Delta S_{BestAnt}$;
16         Assign edge value $k_{BestAnt}$ chosen by ant $BestAnt$ to $\mathcal{G}$;
17     end
18     Perform greedy hill climbing on $\mathcal{G}$ (Algorithm 1);
       // Pheromone update
19     if score $\mathcal{G}$ > score $\mathcal{G}_{best}$ then
20         $\mathcal{G}_{best}$ = copy of $\mathcal{G}$;
21     end
22     Update $\tau$ according to equation (3.7);
23     Update $\tau$ according to equation (3.8) using $\mathcal{G}$;
24     Update $\tau$ according to equation (3.9) using $\mathcal{G}_{best}$;
25     $N_{iter}$++;
       // update total graph
26     if (score $\mathcal{G}_{best}$ > score $\mathcal{G}_{total}$) then
27         $\mathcal{G}_{total}$ = copy of $\mathcal{G}_{best}$;
28     end
29 until $N_{iter}=N_{max}$ ;
30 return Bayesian network with structure $\mathcal{G}_{total}$
```

$$\tau_{i,j,k} \leftarrow \tau_{i,j,k} + 1/|f_{BDeu}(\mathcal{G})| \tag{3.8}$$

- Preserving information about best learned network $\mathcal{G}_{best}$:

$$\tau_{i,j,k} \leftarrow \tau_{i,j,k} + 1/ \mid f_{BDeu}(\mathcal{G}_{best}) \mid \tag{3.9}$$

Additionally to the pheromones the search is also guided by the score of a candidate network. In Ant Colony Optimization this is referred to as heuristics. For the ACO algorithm the heuristic information is defined as:

$$\eta_{i,j,k} = f_{ij}(k) \tag{3.10}$$

with $f_{ij}(k)$ as the local score $f_{BDeu}(X_i \mid \mathbf{Pa}_i) + f_{BDeu}(X_j \mid \mathbf{Pa}_j)$ with edge E_{ij} assigned to value k.

Both the pheromones and the heuristics are used to determine the edge assignment in line 12 of algorithm 3. The probability that state k is chosen for the edge E_{ij} is defined as

$$p_k = \frac{\tau^{\alpha}_{i,j,k} \cdot \eta^{\beta}_{i,j,k}}{\sum_{e=1}^{3} \tau^{\alpha}_{i,j,e} \cdot \eta^{\beta}_{i,j,e}} \tag{3.11}$$

with α and β as balance factors to weight pheromones and heuristics.

The algorithm exists in two variants. The ACO algorithm that has a restricted set of possible edges is called MMACO, while the algorithm without any edge restriction is called ACO. MMACO uses MMPC to gain a set of candidate edges in order to restrict the search space.

As supposed by Pinto et al. (2009), the parameters in our implementation were set to:

- Factor for pheromones: $\alpha = 1$
- Factor for heuristics: $\beta = 1$
- Evaporation coefficient: $\rho = 0.05$
- Number of ants: $m = 20$
- Number of iterations: $N_{max} = 400$

A couple of other Ant Colony Optimization algorithms to learn Bayesian networks are reported in the literature. Since we have no implementation available for these algorithms, and thus we do not compare results of our methods with results of these methods, we just shortly review the main other algorithms in that area. The first algorithm called *ACO-B* was published in de Campos et al. (2002). There are some differences between ACO/MMACO and ACO-B. While ACO/MMACO uses BDeu as scoring function, ACO-B uses the K2 metric. Instead of detecting the next edge to add to the network, in ACO-B every ant builds its own network. To build this network, the algorithm B is used which is a simple greedy algorithm: At every step, the edge with the best score improvement is added to the network. In contrast to GS, any edge added to the network can never be changed again. To introduce non-determinism to the B algorithm, ACO-B refines the rule which edge is added to the network: Instead of using the best-scoring edge, the edge to add is detected by using a probabilistic rule.

I-ACO-B improves the performance of ACO-B by applying two major modifications (Ji, Zhang, Hu & Liu 2009): First, similarly to MMACO, edges are restricted to node-pairs that are not statistically independent according to a independence test. Second, the heuristics which are used to select the next edge are not only the score of two nodes, but the score is weighted with the dependency according to mutual information of the two variables. It is shown in Ji et al. (2009) that these modifications improve the runtime significantly (Ji et al. 2009). In Ji, Hu, Zhang & Liu (2011), I-ACO-B was further developed and Simulated Annealing was additionally employed to optimize the stochastic search process.

In Wu, McCall & Corne (2010), two novel algorithms were introduced: Both algorithms (*ChainACO* and *K2ACO*) detect good node orderings for all variables with ACO and then apply the K2 algorithm to learn a network structure.

3.3.4 Constraint Hill Climbing: CHC, CHC*, iCHC, 2iCHC

A quite interesting approach is constraint hill climbing (CHC) that was first published in Gámez & Puerta (2005). This approach follows the idea of restricting the search space in order to improve the performance of the structure learning. Instead of splitting the structure learning into two phases as it is done for the MM algorithms (MMHC, MMSA, MMACO), CHC identifies edges that should not be evaluated again during a greedy hill climbing run. So the restriction of the search space and the structure learning are performed in one step. Every time during the greedy hill climbing phase when an edge operation is evaluated, the score change is evaluated. If for an edge addition the score is not improved, this edge is removed from the search space. If for an edge removal the score is improved, this edge is removed from the search space. Theoretical considerations have shown that this approach has the drawback of returning structures that might not be a minimal I-map of the distribution represented by data (Gámez, Mateo & Puerta 2007). In (Gámez, Mateo & Puerta 2011) the authors presented new algorithms that are based on the original CHC algorithm that do not have this drawback. Since greedy hill climbing does not have this drawback, they introduced a new method CHC* that uses the output of CHC as a start network for a greedy hill climbing search. Unfortunately, CHC* needs a lot of statistical computations more than CHC and shoots the performance improvement down. iCHC repeats single CHC steps (the output of one CHC step is used as start network for the next step) until the output network does not improve. 2iCHC does exactly two CHC steps, where the output network of the first CHC run is used as a start network for the second CHC run. Both algorithms return minimal I-maps of the underlying distribution.

3.3.5 Recursive Autonomy Identification: RAI

Another approach is Recursive Autonomy Identification (RAI) (Yehezkel & Lerner 2009). This method employs purely constraint based techniques and uses independence tests to obtain the Bayesian network structure.

RAI starts with a fully-connected graph. Then, the following steps are performed: (1) Test of conditional independencies and removal of edges if conditional independencies are detected. (2) Edge orientation based on conditional independence tests. (3) Detection of so-called autonomous substructures. On each autonomous substructure, the steps are applied recursively. During the whole process, the order of the conditional independence tests are increased (beginning with 0). This means that the size of the conditioning set for the conditional independence tests is increased. At the end, the final Bayesian network structure is used as a result of RAI.

4 Robust Learning in Large Domains

Due to the $\mathcal{NP}$-completeness of structure learning (Chickering et al. 1994), many interesting domains for Bayesian network learning face the problem of high dimensionality. For instance, one of todays applications of structure learning is the estimation of biomolecular processes in cells. Here, BNs are used to learn abstract gene-gene interactions in the so called genetic regulatory network from microarray data (Friedman et al. 2000, Dejori & Stetter 2003, Dejori, Schürmann & Stetter 2004). However, with about 30,000 human genes, learning the full network as a whole is a challenging task for current learning methods given the available computational power.

The computational complexity of learning Bayesian networks can be reduced by applying heuristic assumptions about possible network structures. E.g. there is an approach to deal with very large networks (up to hundreds of thousands of variables) (Goldenberg & Moore 2004), but it is restricted to binary variables and a very sparse network structure. To mention another example: in Friedman, Nachman & Pe'er (1999), an algorithm with polynomial computational complexity was introduced. The basic idea behind the so called "Sparse Candidate" algorithm is following heuristic argument: If variables X and Y are almost independent in the data, they are unlikely to be connected in a Bayesian network, and thus, the search can be constrained by allowing edges only between dependent variables. On the other hand, the assumption might be wrong: X and Y can be marginally independent, but conditionally strongly dependent on another variable Z (e.g., X is the XOR of Y and Z). However, it is reasonable to assume that in many domains this dependency structure does not appear. The restriction on network structures that can have links only between dependent variables enables structure learning up to several hundreds or thousands of variables.

If learning in even larger domains, one typically goes one step further and restricts the feature dimensions on a feasible subset of relevant variables that are of high interest (Friedman et al. 2000, Pe'er, Regev, Elidan & Friedman 2001a, Hartemink, Gifford, Jaakkola & Young 2001, Imoto, Goto & Miyano 2002, Dejori et al. 2004, Stetter, Nägele

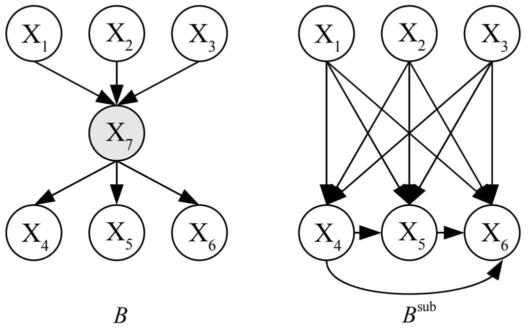


Figure 4.1: The Bayesian network to the left is an example for a complete Bayesian network, the network to the right is the simplest Bayesian network that encodes the same probability distribution, but without the missing variable X_7. Note that the variables X_4, X_5 and X_6 are no longer independent given their parents.

& Dejori 2007). Following the idea that two variables that are marginally almost independent in the data are not linked to each other in the network structure, one can abandon both variables to be present in one network. On the other hand, strongly dependent variables are likely to be connected in the network structure and thus should be put together in one subnetwork. For this, the variables in one single subnetwork can be determined according to their statistical dependence given the data. As a result, a typical pipeline for learning Bayesian networks from high-dimensional data could be stated as a two-step process (Stetter et al. 2007):

1. Based on a statistical method, choose a number of highly-relevant variables.

2. Learn a Bayesian network with the set of variables selected in step 1.

Even though the structure learning becomes feasible, the restriction on a small set of variables for learning is a potentially problematic step which can lead to a strongly corrupted estimation of the true structure. First of all, edges incident to missing variables cannot be learned by definition. Edges that are missing in the learned network, but contained in the true underlying network structure, are called false negatives. Second, additional false positive edges might be learned to explain statistical dependencies that can not be represented by a dependence on the missing variables (see figure 4.1 for an example (Binder, Koller, Russell & Kanazawa 1997)). False positives are those edges that exist in the learned network, but do not exist in the true underlying network structure. While there are many publications about the quality of network reconstructions, the impact of the reduction of variables is not well evaluated. To rate the impact of such a search space restriction, we introduce new methods in the chapter.

To evaluate the extent of the subnetwork-based structural shifts in the learned networks, we simulate the reduction of variables with a method called subnetwork learning. This approach was originally published in Nägele (2005) and systematically estimates the network structure of the whole network step by step using smaller subnetworks. According to the method that is used to obtain smaller networks, we put variables together in one subnetwork that are statistically dependent on each other. In this chapter we extend the analysis of Nägele (2005) and analyze the dimensionality reduction in a systematic way. Based on known benchmark networks, we can answer how many network errors appear caused by the reduction of dimensionality.

In benchmark cases with known network structure, subnetwork learning can be used to estimate the influence of learning partial networks with moderate size on the features of the network structure. If learning the structure from data in real-world scenarios, the true structure is never known and subnetwork learning can not help to identify errors in the learned network. For this a method would be suitable that assigns confidence levels to structural features in order to distinguish between true network structures and false positives. To minimize the error induced by subnetworks in real-world cases with unknown true network structure, we develop and propose a new robustness assessment algorithm, called dimensional bootstrap. For that, we iteratively learn the structure of a small subnetwork. However, with each iteration, a somewhat different set of variables is added to the subnetwork in such a way that even weakly dependent variables can evolve their influence on the estimated network structure. With dimensional bootstrap, each edge gets a confidence level that helps to separate between true edges and false positive edges in the learned network.

Large and high-dimensional domains commonly imply the additional problem of sparse data. A common approach to deal with sparse data and the fluctuations contained therein is the non-parametric data bootstrap (Efron & Tibshirani 1994). This method allows to assess those dependencies that are caused by the underlying true dependency structure between the variables and not by statistical fluctuations in the data (Friedman, Goldszmidt & Wyner 1999, Friedman et al. 2000). However, in very large but sparse domains there might be several variables that are highly dependent on each other only because of such fluctuations. By restricting the learning on a subnetwork with highly related variables, the presence of fluctuations in the small sub data set can be higher-than-average. To estimate the influence of such fluctuations, we propose a surrogate data analysis. For that, we perform the same steps used for structure learning with artificially generated data that contain no dependencies between the variables at all. If applying this proposed new method one can answer which fraction of the number of edges is learned because of true dependencies in the data and which fraction is just based upon fluctuations in the data.

Before we introduce our new methods, in the next section we shortly describe common methods to measure the robustness of features in the area of Bayesian networks.

4.1 Robustness Assessment

Primarily, Bayesian networks represent the joint probability distribution among random variables. There are many applications that use this distribution directly. For instance, the Naive Bayes classifier utilizes the probability distribution to classify samples. Another example is the work of Dejori & Stetter (2004) who train Bayesian networks from microarray data in order to simulate *in-silico what-if* scenarios.

However, there are several applications of Bayesian network structure learning that have the primary goal of structural feature estimation, i.e. they make use of the link structure of Bayesian networks, neglecting the quantitative part (Friedman & Koller 2003). A prominent example is the previously mentioned reconstruction of abstract gene-gene interactions (Friedman et al. 2000, Dejori & Stetter 2003, Dejori et al. 2004). The most common approach to discover BN structure is to learn a single high-scoring model. This model (or its Markov equivalence class) is used as the structural estimation of the domain. It has been shown that the likeliness of the highest scoring network is order of magnitudes higher than any other network structure in small domains with a number of samples that is much higher than the number of random variables (Heckerman, Meek & Cooper 1997). Unfortunately, in many interesting domains, the number of observations are much smaller than the number of variables. Especially in the case of genetic network reconstruction, there are usually only few hundred samples in a microarray data set, but there are commonly thousands of genes measured. In cases where the number of samples is relatively small in comparison to the number of variables, it is likely that there are many models (graph structures) that describe the dependency structure reasonably well.

A possible solution to the problem of model uncertainty is to use the standard Bayesian way. There, one sums over all possible models, weighted by each models likelihood, to compute the posterior of a quantity of interest. Let be f this quantity of interest, such as a structural feature in the graph, the posterior distribution given data **D** is then given by

$$p(f \mid \mathbf{D}) = \sum_{k=1}^{K} p(f \mid \mathcal{G}_k, \mathbf{D}) p(\mathcal{G}_k \mid \mathbf{D}), \tag{4.1}$$

where K is the number of possible graph structures $\mathcal{G}_k$. The posterior $p(\mathcal{G}_k \mid \mathbf{D})$ for each model $\mathcal{G}_k$ in the light of data **D** is given by equation 3.4. It has been shown that averaging over all possible models provides better results than using only the highest-scoring model (Madigan, Raftery, Wermuth, York & Zucchini 1994).

However, a direct implementation of the Bayesian approach is difficult. Except for domains with only few variables, there is no practical way to calculate the sum in equation 4.1 since the number of possible structures grows superexponential with the number of variables, leading to an enormous amount of different structures.

Mainly, two different approaches have been developed in order to calculate the confidence of a feature f in an approximate way. The first approach, introduced by Madigan, York & Allard (1995), approximates the sum by considering only a subset of possible structures. To obtain this subset, they use a Markov Chain Monte Carlo Method with a Markov Chain over structures. While this approach approximates the Bayesian approach (summing over all possible models), the second approach, called Bootstrap approach, estimates the confidence indirectly by manipulating the data set **D** (Friedman, Goldszmidt & Wyner 1999). Here, we shortly introduce both ways of estimating a feature's confidence.

4.1.1 MCMC Methods

The Markov chain Monte Carlo approach, as introduced in Metropolis, Rosenbluth, Rosenbluth, Teller & Teller (1953) and Hastings (1970), is a common framework for approximating probability distributions that can not be solved in closed form or are hard to calculate because of their complexity. In order to approximate 4.1, one can build a Markov chain that has the posterior distribution $p(\mathcal{G}_k \mid \mathbf{D})$ as equilibrium distribution. Samples generated from these chain are used to estimate 4.1 by summing over the restricted space of sampled models rather than on all possible graph structures. The usage of the MCMC approach to estimate the confidence of features in Bayesian networks was first proposed by Madigan et al. (1995). They build a Markov chain over the space of directed acyclic graphs and use basic edge operations as proposed movements between the states of the Markov chain.

Given a graph structure $\mathcal{G}_{old}$, the probability A of moving to another structure $\mathcal{G}_{new}$ is given by (Grzegorczyk & Husmeier 2008):

$$A = min\left\{\frac{p(\mathbf{D} \mid \mathcal{G}_{new})\, p(\mathcal{G}_{new})}{p(\mathbf{D} \mid \mathcal{G}_{old})\, p(\mathcal{G}_{old})} \cdot \frac{q(\mathcal{G}_{old} \mid \mathcal{G}_{new})}{q(\mathcal{G}_{new} \mid \mathcal{G}_{old})}, 1\right\}, \tag{4.2}$$

where the proposal distribution $q(\mathcal{G}_{new} \mid \mathcal{G}_{old})$ captures the proposed movements (here: basic edge operations). Thereby, the Hastings factor $\frac{q(\mathcal{G}_{old}|\mathcal{G}_{new})}{q(\mathcal{G}_{new}|\mathcal{G}_{old})}$ has to be chosen in such a way that the process has the correct stationary distribution.

Due to convergence problems if basic edge operations are used, Friedman & Koller (2003) suggest to build a chain over the orders of network variables rather than directly on DAG structures. They show that this approach has substantial benefits in terms of convergence of the Markov chain. Recently, this method was further improved by Grzegorczyk & Husmeier (2008).

A major drawback of MCMC approaches to estimate the confidence of structural features is the vast computational effort to calculate 4.1, in particular for large networks with hundreds of variables. Even if we sum only over a small subset of networks generated by the Markov chain rather than calculating the weighted sum over all possible models, the computation remains challenging.

4.1.2 Data Bootstrap

While the MCMC method belongs to the Bayesian paradigm, there is also a frequentist approach of dealing with sparse data. In the frequentist view, data are generated by a process with true but unknown parameters. The parameters are estimated from the data with maximum likelihood, leading to a single model representing the data. Model uncertainty can be handled by repeating the data generation process several times, leading to a set of independent and identical distributed data sets. The ensemble of models, where each model is the estimator of one of these data sets, gives an estimation of this uncertainty. In fact, the procedure of iterated data generation is unrealistic in most cases. Usually, there is not even a small set of independent and identical distributed data sets available, but only one single data set. For instance, genomic or proteomic data are very sparse in general due to two main reasons: (1) the costs for one measurement are high and (2) each measurement represents a single patient, and the amount of patients can not be increased arbitrarily. Thus, it is even hard to get a data set of a suitable number of samples, not talking about a large set of data sets. Also in many other domains, the data generation process can not be iteratively processed (Husmeier, Dybowski & Roberts 2004).

In terms of computational costs, it would be preferential to base the confidence assessment on the frequentist paradigm, i.e. to learn a single maximum likelihood or maximum a posteriori (MAP) BN network structure for each of the data sets. Luckily, the method of iterated sampling can be approximated by a Q-fold non-parametric bootstrap procedure (Zoubir 1993, Efron & Tibshirani 1994). Thereby, at each fold q, a data set $\mathbf{D}_q$ is generated from the original data set by re-sampling with replacement. Thus, some of the original samples may occur in the bootstrap data set repeatedly, while other samples may be missing. The bootstrap data set $\mathbf{D}_q$ contains the same number of samples as the original data set $\mathbf{D}$ and reflects the same underlying probability distribution. The Q Bayesian networks trained from the Q bootstrap replicas result in an assessment of the structure variability caused by finite sample fluctuations. The use of a boostrap approach to estimate the confidence of structural features in Bayesian networks was first proposed by (Friedman, Goldszmidt & Wyner 1999).

4.1.3 Feature Graph

To describe the variations in the structure of several Bayesian networks, the framework of "feature partially directed graphs" (fPDAG) (Dejori 2005) that is based upon the work in Friedman, Goldszmidt & Wyner (1999) is used. An fPDAG can deal with structural uncertainties by assigning a value for the belief in the features "edge presence" and "edge direction". We shortly introduce the framework of fPDAGs in this section.

As explained in a previous chapter, the direction of an edge in a Bayesian network can be ambiguous, and, thus, these edges should be counted as undirected edges. Before creating an fPDAG, all Bayesian networks forming the estimation of the underlying network are transformed into their PDAG representation (Chickering 1995).

The confidence in a feature can be written as

$$p(F) = \frac{1}{Q}\sum_{q=1}^{Q} f(BN_q), \tag{4.3}$$

where $f(BN_q)$ has the value one if the feature feature exists in the q-th network, otherwise it is zero. For the fPDAG, we use four features: $f_{i\rightarrow j}$ is the feature describing a directed edge from X_i to X_j, $f_{i\leftarrow j}$ a directed edge from X_j to X_i, f_{i-j} an undirected edge and $f_{i\not\leftrightarrow j}$ no edge between both variables.

The feature of an edge between two variables X_i and X_j can be described by a probability distribution with four states, that is

$$p_{i\leftrightarrow j} = \{p_{i\rightarrow j}, p_{i\leftarrow j}, p_{i-j}, p_{i\nleftrightarrow j}\}. \tag{4.4}$$

$p_{i\rightarrow j}$ denotes the probability of a directed edge from X_i to X_j, $p_{i\leftarrow j}$ the probability of a directed edge from X_j to X_i, p_{i-j} the probability of an undirected edge and $p_{i\nleftrightarrow j}$ the probability that there is no edge between the two variables. Each feature is calculated according to 4.3. Sometimes in the forthcoming sections we neglect the direction of the edges and we are just interested in the presence of an edge. The confidence in an edge regardless its direction is $p_{i\rightarrow j} + p_{i\leftarrow j} + p_{i-j}$.

The fPDAG of Q Bayesian networks with corresponding Bayesian network structures $BN_q,\ q \in \{1, ..., Q\}$ is a graph which contains all variables that are present in the Bayesian networks. The edge between each pair of variables X_i and X_j is weighted with its feature distribution $p_{i\leftrightarrow j}$. Thus, unlike Bayesian networks or PDAGs, the structure of fPDAGs is neither an acyclic directed nor a partially directed acyclic graph. Instead, it has undirected edges between related variables, and these edges are labeled with $p_{i\leftrightarrow j}$ (Dejori 2005).

This work considers partially overlapping Bayesian networks, i.e., there are Bayesian networks that do not have all variables in common. As a result, an edge between two variables X_i and X_j can only be learned in networks that contain both variables. All other networks cannot contain the edge, thus these networks also cannot be directly used to calculate the feature probability with equation 4.3. Hence, we extend the definition of a feature graph given by Dejori (2005) with the ability to deal with partially overlapping networks. We define the probability of a feature solely based on these networks that can contain the feature. If $f_{i\leftrightarrow j}(BN_q)$ is zero for all networks BN_q that cannot contain the feature, the feature probability can be estimated by replacing the normalization factor Q in 4.3:

$$p_{i\leftrightarrow j} = \frac{1}{Q_{X_i,X_j}} \sum_{q=1}^{Q} f_{i\leftrightarrow j}(BN_q), \tag{4.5}$$

where Q_{X_i,X_j} is the number of Bayesian networks that contain both variables X_i and X_j. It is necessary to point out that this calculation of a feature confidence can lead to a poor estimation of the confidence since an improbable edge could be assigned a confidence of one if only one subnetwork contains both variables X_i and X_j. An edge between two variables that appears in 90 of 100 networks that contain both incident variables would have less confidence. It seems apparent that this edge is more likely than the edge between X_i and X_j. However, the good estimation of the network structure with the fPDAG framework as it is presented later in section 4.3.2 shows that the approximation of the confidence given in equation 4.5 seems to be quite reasonable.

4.2 Sparse Data Analysis

As stated before, Bayesian network structure learning is an $\mathcal{NP}$-hard problem. Thus, if applied to high-dimensional domains, the large number of variables is usually restricted to a small set of variables of interest, and the Bayesian network structure is learned for this small set of interest. Generally, this process can be separated into two steps:

1. *Selection of highly relevant variables (set of interest).* At this step, the variables are separated into two sets: a set of interest (containing all variables used for BN structure learning) and a set of less relevant variables that should not occur in the BN structure and, thus, are omitted for the structure learning procedure. For instance, the set of interest can be selected by a human with expert knowledge. In practice, all variables are often ranked according to a specific criterion and the two sets are defined according to the ranked list of the variables. E.g., Dejori et al. (2004) apply a statistical test to detect genes that play an important role in acute lymphoblastic leukaemia and use the 271 most relevant genes for BN structure learning.

2. *Bayesian network structure learning.* Now, the set of highly relevant variables detected in step 1 are used for Bayesian network structure learning. Thereby, the structure learning algorithm is applied only to the variables that belong to the set of interest, all other variables are omitted.

In this section we do not focus on the problems arising if structure learning is applied to a small set of variables, while most variables remain latent in the network. These problems are discussed in more detail in the following sections. Here we focus in particular on the problem of sparse data, i.e. a small number of samples compared to the number of variables observed, paired with the dimension reduction usually applied to large data sets. Thereby, an insufficient number of observation can strongly impact the quality of the structure learning procedure leading to a network structure not representing the true network structure underlying the data.

Besides, statistical test methods to narrow down the amount of variables might be affected by fluctuations in sparse data sets. There might be variables that have no relevance for the tested condition, but instead they are selected because of fluctuations which in turn might influence the quality of structure learning. Typically, for each of the variables in the domain, one statistical test is performed, leading to the actual need of multiple test correction. Common approaches to control the false positives are controlling the familywise error rate or the false discovery rate (Benjamini & Hochberg 1995). However, these methods are typically used to obtain significant variables for the tested condition, and thus to determine the variables that belong to the set of interest. In fact, we are interested in the influence of sparse data on the quality of the learned network structure.

As described before, a common approach to deal with sparse data and the therein contained fluctuations is the non-parametric data bootstrap (Efron & Tibshirani 1994) using a "perturbed" version of the available data in order to enable an confidence estimation of network features. Edges that have a high confidence in the fPDAG comprised by the perturbed networks are shown to be rarely false positives (Friedman, Goldszmidt & Wyner 1999, Friedman et al. 2000). This result is based on observations on small-scale networks like the Alarm network consisting of 37 variables and 46 edges. Instead, we have domains with thousands of variables.

4.2.1 Method

To estimate the influence of sparse data, we simulate the previously described two-step process (reduction in dimensionality and structure learning) by using artificial benchmark data. First, we apply this process to data that was generated from a network with known structure. Second, we apply the same procedure to surrogate data. Figure 4.2 illustrates the generation of surrogate data by scrambling the original data.

Based on an original data set (left hand side in the figure), the observations are randomly interchanged variable-wise. Thus, all higher-order statistics are destroyed in the surrogate data (right hand side) and all variables are independent from each other, while the variable-wise marginal probabilities are kept stable. Since the variables are independent from each other, all edges that are learned from surrogate data are false positives.

By comparing the number and confidences of edges in both cases (normal and surrogate data), it is possible to quantify the fraction of edges that are caused by fluctuations in sparse data sets. As a measure we define the "predictive value" as the number of edges

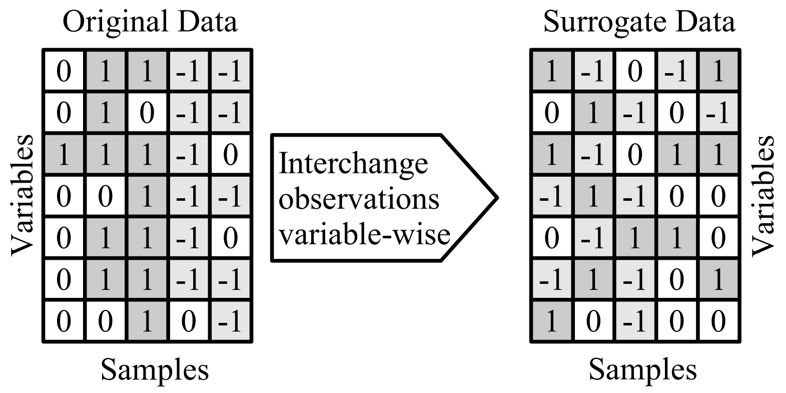


Figure 4.2: Procedure of surrogate data generation (scrambling): The observations for each variable in the original data set (left hand side) are interchanged randomly. The resulting data set without any higher-order statistics is referred to as "surrogate data set" (right hand side).

in the normal case divided by the number of all edges (normal and surrogate). A value of one implies that all learned edges are originated by dependencies that are not caused by the sparseness of the data. A high predictive value, however, does not imply all edges to be true positives. In fact, a high value only indicates that there are more dependencies in the data than in a randomly chosen data set. A differentiation between true positives and false positives is not considered here.

A lower predictive value of 0.5 means that the learned edges have no meaning at all since the same amount occurs in the surrogate case. For this reason, the original data set contains relationships that are probably contained in any artificial (surrogate) data set by chance. Hence, the predictive value quantifies the structural shift that can be caused by data fluctuations.

To identify the influence of the commonly used preselection of variables on the fluctuations in the network, we apply three different learning scenarios. The three scenarios are sketched in figure 4.3.

- The "Normal" scenario imitates the method to preselect variables in order to reduce the size of the network: Based on a high-dimensional data set, a couple of variables are selected. For this set of variables BNs are learned by means of a non-parametric bootstrap procedure.

- The "Scrambling before Variable Selection" scenario imitates the "Normal" case,

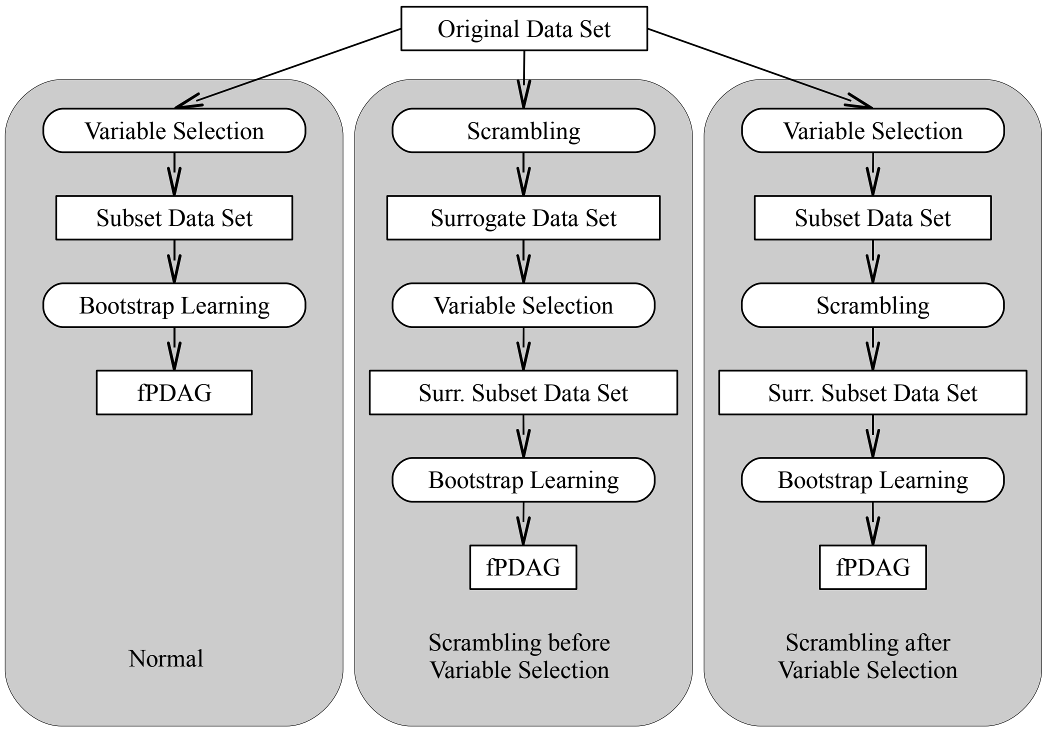


Figure 4.3: Workflows for sparse data analysis: The general workflow of learning small networks in large domains is shown on the left hand side ("Normal" case). The "Scrambling before Variable Selection" method imitates the "Normal" case, but with surrogate data. The learning from a small data set is imitated by the "Scrambling after Variable Selection" method.

however uses surrogate data in order to perform the selection of variables and learning the BNs. Since all dependencies in the data are just caused by fluctuations, the variable selection method selects variables that are assumed to be dependent just because of fluctuations in the data. So this approach enables to estimate the influence of the variable selection on fluctuations in the learned BNs.

- The third method, called "Scrambling after Variable Selection", does the scrambling after the variable selection. This method allows to estimate the number of edges that are learned by chance in a network having the size of the number of the selected variables, neglecting the variable selection procedure.

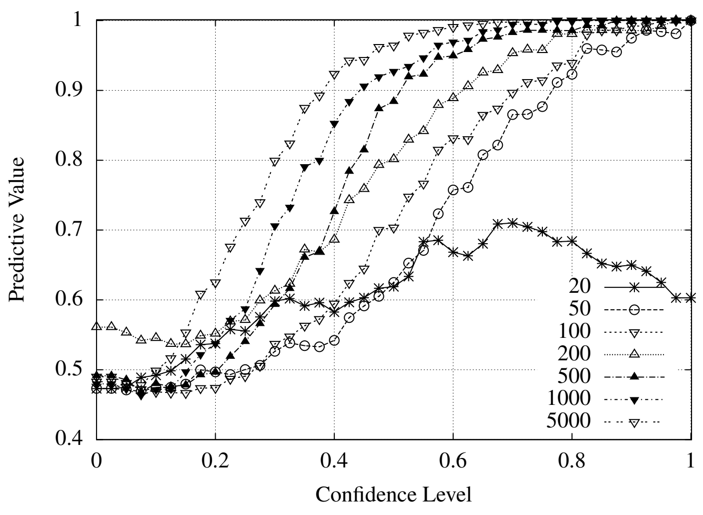
4.2.2 Results

The influence of fluctuations in sparse data on the structural robustness of learned networks is simulated by using data sets of different sizes that are drawn from the Alarm_270 benchmark network (see appendix A.2 for an overview of the network). In particular, we sampled five different data sets for each of the following sample sizes: 20, 50, 100, 200, 500, 1000, 5000.

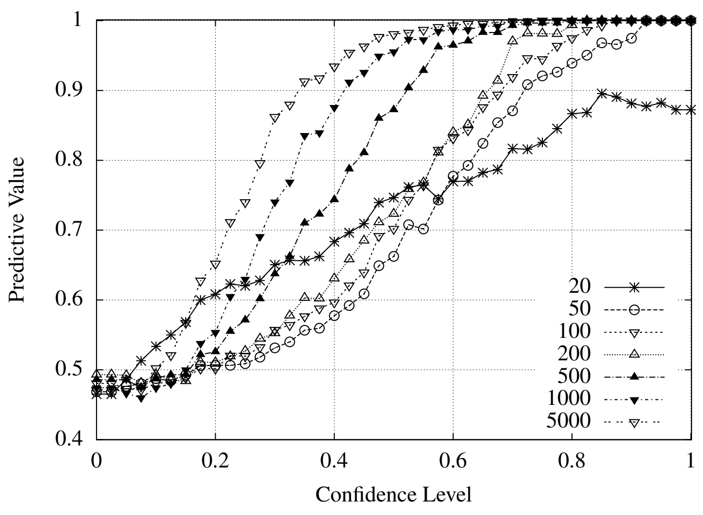
To simulate the procedure of feature reduction as it is applied for the preselection of a subset of genes from a large microarray data set, one of the 9990 variables in the benchmark network is chosen randomly and the 199 most related variables according to the mutual information measure are determined. The resulting data set ("Subset Data Set") with 200 dimensions is used to learn 40 Bayesian networks with a non-parametric bootstrap approach ("Bootstrap Learning"). An overview over the workflow is given in figure 4.3 on the left hand side ("Normal" case). In addition, the same steps are performed similarly, but using surrogate data instead of the data sets that are drawn from the Alarm_270 network (see figure 4.3, "Scrambling before Variable Selection" case). It is obvious that this procedure does not exactly reflect the procedure of gene preselection from microarray data as it was described before. There, the genes are usually detected by comprising some external information such as disease state or type of treatment. Here, however, we want to show results based on benchmark networks since it enables a comparison between learned networks and the underlying true benchmark network.

Figure 4.4(a) illustrates the predictive value for different data sizes based on the Alarm_270 benchmark network. The predictive values are calculated for several thresholds for the edge confidence, averaged over all five different data sets and plotted separately for each sample size. All edges with a confidence higher than the threshold are treated as learned edges, whereas all others with lower confidence are neglected. It can be seen that the bootstrap networks learned from a sufficiently large data set (at least 1000 samples) have high predictive values even for a very low threshold for the confidence. The figure clearly shows the impact of the sample size on the informative value of learned networks. In the case of 5000 samples a confidence threshold of 0.4 leads to an estimation of the network structure that is mainly based on the dependencies in the data, having a predictive value of greater than 0.9.

By contrast, the results get worse if data gets sparse. With a sample size of 50 or smaller the confidence must be at least 0.8 to assure a predictive value of 0.9 or higher, while a



(a) Scrambling before variable selection



(b) Scrambling after variable selection

Figure 4.4: Predictive values (fraction of edges in "normal" case and sum of edges of "normal" and surrogate cases) against different thresholds for the confidence level based on the Alarm_270 benchmark network. Results are shown for several different sample sizes.

robust identification of dependencies in the data is nearly impossible with a very sparse data set with only 20 samples: This can be seen by the fact that the predictive value barely

Table 4.1: Number of learned edges for some of the configurations shown in Figures 4.4(a) and 4.4(b). Numbers are shown for the normal (N), the scrambling before variable selection (S1) and the scrambling after variable selection (S2).

	50 samples			**100 samples**			**500 samples**			**5000 samples**		
Conf	**N**	**S1**	**S2**	**N**	**S1**	**S2**	**N**	**S1**	**S2**	**N**	**S1**	**S2**
0.2	320.0	335.0	318.2	315.8	350.4	313.6	217.8	221.0	196.4	159.4	95.6	85.0
0.4	93.4	75.6	68.2	76.6	52.2	51.8	75.4	28.4	26.0	97.0	8.0	6.8
0.6	44.2	14.4	13.8	44.4	9.0	9.0	60.0	3.2	2.2	86.0	0.8	0.6
0.8	26.4	2.2	2.0	31.0	2.0	0.8	54.6	0.8	0.0	80.8	0.0	0.0
1.0	10.8	0.0	0.2	16.8	0.0	0.0	40.8	0.0	0.0	72.4	0.0	0.0

Table 4.2: Standard deviation for the values shown in table 4.1.

	50 samples			**100 samples**			**500 samples**			**5000 samples**		
Conf	**N**	**S1**	**S2**	**N**	**S1**	**S2**	**N**	**S1**	**S2**	**N**	**S1**	**S2**
0.2	23.0	25.2	35.2	12.4	9.1	10.4	9.4	7.8	11.1	11.4	5.7	12.9
0.4	8.2	7.6	7.4	9.6	6.9	4.8	4.3	6.8	2.9	10.9	1.8	1.3
0.6	4.7	5.0	3.5	7.9	1.7	1.4	5.2	1.2	1.7	12.5	0.4	0.5
0.8	5.0	1.5	0.9	7.1	1.7	1.0	5.9	0.7	0.0	13.1	0.0	0.0
1.0	4.2	0.0	0.4	6.4	0.0	0.0	3.7	0.0	0.0	11.6	0.0	0.0

exceeds 0.7, showing the missing significance if learning networks from data with such few data points: A predictive value of 0.7 means that only 70% of the edges are present in the learned bootstrap networks, while 30% occur in a network that contains only false positives by definition. Surprisingly, the best predictive value for 20 samples is not observed for a confidence threshold of one like it would be expected, but for a lower confidence of around 0.7. With a threshold of one, the predictive value with about 0.6 is even worse. From this observation one can reason that conclusions about learned network structures (at least in this example) for a sample size of about 20 lack of any kind of significance.

While the figure shows only the predictive value which measures the relative frequency of edges, table 4.1 shows the absolute values of learned edges for "Normal" (column **N**), "Scrambling before Variable Selection" (column **S1**) and "Scrambling after Variable Selection" (column **S2**) cases with respect to some selected sample sizes. The edge numbers are reported for five different values for the confidence threshold. For a small threshold for the edge confidence (e.g. 0.2) and a small data set, the amount of edges learned in the **S1** case is even higher compared to the normal learning procedure (**N**). Also the standard deviation (see table 4.2) implies that there is no significant difference between the three

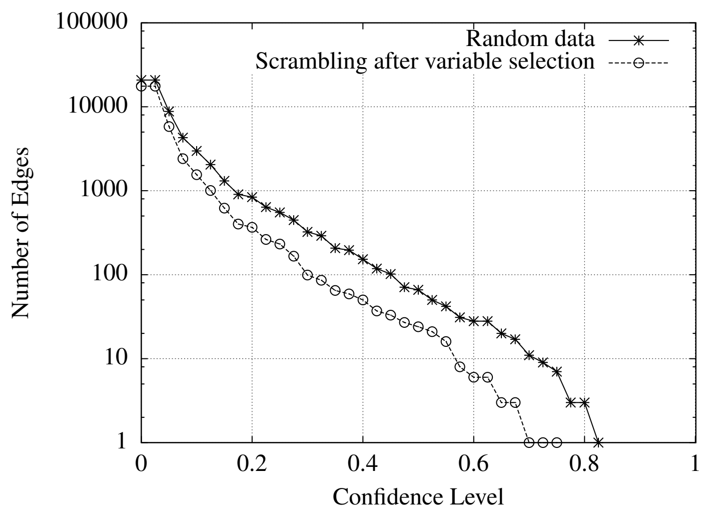


Figure 4.5: Number of edges learned from a 200 variables data set. The variables are selected from a randomly generated data set with 100.000 variables, while there are no dependencies among these variables.

cases. On the other hand, for thresholds of the edge confidence (about 0.8), as well as for large data sets, the predictive values are highly significant.

One fact is quite interesting: As expected, the number of edges learned in the "Scrambling before Variable Selection" (**S1**) case is most higher than in the "Scrambling after Variable Selection" (**S2**) case, however the difference is barely significant. This implies that the feature reduction procedure leads to a higher number of edges, however the increase compared to networks learned from a data set without any dependency does not reach a significant level. The results indicate that the commonly used method to preselect variables for network learning leads to a little biased estimation of the network with occurrence of false positive edges, which can be neglected at least with the size of network (9990 variables) used in this example.

With a network with about 10.000 variables, the pre-selection shows no massive influence on the false positive rate, however with a higher number of variables the picture could be different. To show the influence of the number of variables, we applied the variable-preselection on a data set with 100.000 variables and 200 samples. The data set is generated randomly and contains no true dependencies among the variables. The number of edges learned in a network with 200 pre-selected variables is shown in figure 4.5, labeled as "Random data" (simulating the "Scrambling before Variable Selection (S1)" case). While for a low number for the edge confidence, there are quite a lot of edges in the network, with a high confidence of 0.8 there remain only 4 false positive edges. To show the difference between a network that was learned from a pre-selected data set and a data set that contains no dependencies at all, we performed a scrambling after the variable selection (S2) to destroy the dependencies. We learned bootstrap networks from this scrambled data set ("Scrambling after variable selection"). The figure shows that the number of learned edges is up to a factor of ten lower than in the case before. On the other hand, this means that by applying the feature reduction the number of false positives is also up to a factor of ten higher. This has a negative impact on the predictive value: For instance, in our benchmark case with true dependencies in the data (see table 4.1, columns "S1" and "S2"), up to only one edge more is learned in the "S1" case compared to the "S2" case for a confidence of 0.6, while here we have a factor up to ten. In table 4.1, columns "N", one can also see that for an edge confidence of 0.6 around 40 to 60 edges are learned for sample sizes of 500 and lower. This number of edges is in the same order of magnitude as the number of edges learned in this case (around 40 edges are learned from random data, see figure 4.5).

In conclusion, the number of variables in the original data set has a big influence on the number of of edges learned because of fluctuations: In the aforementioned example, the amount of false positive edges was up to ten times higher for a data set with 100.000 variables compared to the case if no feature reduction is applied and just 200 variables are contained in the data set.

4.2.3 Discussion and Related Work

Results from the last section have shown that the commonly used approach to pre-select some variables from a data set with many variables can have the drawback that edges are just learned because of fluctuations in the data and are not based upon true dependencies. One can rate the amount of edges caused by fluctuations in the data by calculating the "predictive value" which was introduced in this chapter.

If the amount of variables is reduced and the BN is learned for a subset of the variables, the predictive value can be used to calculate the fraction of edges caused by fluctuations. If data-bootstrapping is used in addition, one can control the fraction of edges caused by fluctuations by setting the predictive value. The predefined predictive value can then be used to distinguish between "significant" edges and "not significant" edges by means of the edge confidences obtained by the bootstap procedure. Thus on can directly control the fraction of edges caused by fluctuations. In case of reduction of dimensionality, we propose to always calculate the predictive value to determine the significance of the edges in the learned BN.

Listgarten & Heckerman (2007) have published an approach to calculate the false discovery rate (FDR) of learned edges that is very similar to our approach. They use a way to calculate the FDR which is similar to the calculation of the predictive value: The predictive value is the inverse of one plus the false discovery rate: $(1 + FDR)^{-1}$. While we additionally applied a bootstrap procedure to calculate edge confidences, they just learn one network from real data. Afterwards, they get the false discovery rate as the fraction of the expected number of false edges and the number of edges learned from real data. The expected number of edges is calculated as the mean number of edges over networks learned from data sets drawn from a null-model.

Later, an approach to control the false discovery rate was published in Tsamardinos & Brown (2008). Based on statistical independence tests, they restrict the neighbours of each variable in such a way that the given value for the false discovery rate is not exceeded. This approach is not suitable for pure score-based BN structure learning algorithms. However, it can be used to define a possible set of parents and children for each variable to restrict any score-based search algorithm on edges that fulfil the parent-child relationships defined by these sets.

4.3 Subnetwork Learning

In the last section, we investigated the influence of sparse data in combination with the reduction of the network size by omitting "uninteresting" variables. We were interested in the fraction of learned edges that do not represent underlying dependencies among the variables. In this section, we show the influence on the learned network structure if there are missing variables in the network. In contrast to the last section, we focus on edges that may represent dependencies among variables, but are false positives since the dependencies are indirect and can be explained by direct dependencies using a missing variable.

By removing a variable, the dependency structure between the remaining variables may change dramatically. The potential influence of one missing variable on the structure is shown in figure 4.1 (Binder et al. 1997). The removal of one important variable (here X_7) can disrupt the structure of the Bayesian network. The direct relationships that pass originally through X_7 in the left network (B) must be represented by indirect relationships

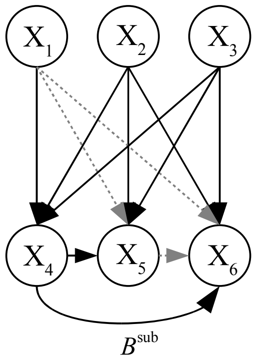


Figure 4.6: This Bayesian network was learned from a data set with 10000 samples. Out of the 12 truly existing edges in the underlying dependency structure B^{sub} (presented in figure 4.1), nine edges are learned correctly. Only three edges are missing in the learned network (missing edges are displayed in grey and dotted).

between the remaining variables in the subnetwork to the right (B^{sub}), which leads to a massive appearance of false positive edges.

While Binder et al. (1997) have presented this result theoretically, most of the false positives that appear in B^{sub} could represent dependencies among variables that are too small to be learned from data. Thus we created a benchmark network to show the impact of removing one node on the learned network. As structure for the benchmark network we used network B (figure 4.1). The local probability distributions where chosen randomly, which is the same procedure as used e.g. in Nielsen & Nielsen (2008) to generate probability distributions. From this benchmark network, 10.000 samples were generated, and the values for variable X_7 were removed to obtain a data set to learn a network without node X_7. From this data set, we learned a network structure by using Greedy Hill Climbing with BDeu as scoring function. The structure of this learned network is shown in figure 4.6. Out of the twelve truly existing edges in the underlying dependency structure B^{sub}, nine edges are learned correctly. Only three edges are missing in the learned network. As exemplary shown with this benchmark network, the appearance of false positives if removing a variable is more than a theoretical problem: There is a massive appearance of false positives in this small example.

4.3.1 Method

To estimate the influence of learning partial networks on the estimated network structure, we model this constraint by learning a set of partially overlapping subnetworks with data drawn from benchmark networks. All the learned subnetworks are afterwards compared to the structure of the "original" benchmark network and to a learned network structure that covers the whole domain without missing variables. By comparing these networks it is possible to draw conclusions to what extent relying on subnetworks affects the estimated structure and therewith the appearance of false negative and false positive edges.

Starting from one single variable, we select the variables that are statistically most dependent on this variable. As a measure for the dependency we use mutual information. The dependent variables are henceforth referred to as a "neighborhood". The variable and its neighborhood together form one subnetwork. By creating the subnetwork with one central variable and the variables that are statistically most dependent on this variable, the approach emulates the real-world case where the variables of one network are selected by their statistical dependencies. To measure the mean influence of learning subnetworks, the approach does not choose only one central variable, but instead creates one subnetwork for each variable in the large-scale data set.

The algorithm itself is based upon an idea that was initially published in Nägele (2005), however here we present a different analysis of the subnetwork learning algorithm. The algorithm can be outlined as follows: In the first step, for each variable $X_i \in \mathbf{X}$, where $\mathbf{X}$ is the set of all variables in the original network, a Bayesian subnetwork with variables $\mathbf{M}_i$ is put together, where $\mathbf{M}_i$ is the neighborhood of X_i and X_i itself. Mutual information is used as a measure for the neighborhood between two variables. The most strongly dependent variable which is not contained in the neighborhood $\mathbf{M}_i$ of X_i is added to that neighborhood. This procedure (adding of a variable) is iterated until $\mathbf{M}_i$ contains a predefined number of variables. The second step involves the learning of the BN on the small local subnetwork $\mathbf{M}_i$.

In order to test the structural robustness of learning subnetworks, a comparison is undertaken between the structure of each single subnetwork and the corresponding part of the original network by applying different distance measures (see section 4.3.2). The difference in performance between the subnetwork learning and the learning of the complete network allows to estimate the structural shifts introduced by learning subnetworks. This analysis was not carried out in Nägele (2005). In a second step an fPDAG formed by the set of all

Algorithm 4: Subnetwork Algorithm

```
  Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$
  Output: set of subnetworks
  // Learn network for each variable's neighbourhood
1 foreach $X_i \in \mathbf{X}$ do
      // 1: Create neighbourhood
2     $\mathbf{M}_i := \left\{X_j \mid X_j \text{ neighbour of } X_i\right\} \cup \{X_i\}$;
      // 2: Learn network
3     $B_i :=$ LEARN_BN($\mathbf{M}_i, \mathbf{D}_{\mathbf{M}_i}$);
4 end
5 return set $\mathbf{B}$ containing all $B_i$
```

subnetworks is created. This enables an assessment of robust features that are preserved with subnetwork learning, in particular the edge confidence can be used to discriminate between robust edges and those edges that are only caused by learning subnetworks. However, this implies that the subnetworks cover the whole domain (one subnetwork for each variable), which is definitely not the case if one single subnetwork is used for dependency estimation. Thus, the assessment of robustness with the fPDAG cannot be utilized with only one single subnetwork. An analysis based upon the fPDAG representation was already carried out in Nägele (2005), however in this chapter we do an ROC-like analysis which was not done in Nägele (2005).

4.3.2 Results

Based on subnetwork learning, we investigate the influence of learning small subnetworks that are composed by a statistical method on the quality of the learned network structure. For the investigation we use the Alarm_50 and ALL_benchmark_1000 networks with 500 samples for learning (see appendix A.2). The subnetwork learning was performed for subnetwork sizes of 20, 50 and 100. The learning was not carried out for larger sizes because of the computational time that would be needed for such configurations. Data bootstrap as it was used for the surrogate data analysis before is also not applied due to its high computational costs. As BN learning algorithm we used simulated annealing as used in Dejori & Stetter (2003) and described in section 3.3.2.

Each learned subnetwork is compared to the original, true network structure on the basis of several distance measures. To avoid the penalization of structural differences that cannot be statistically distinguished, the measures are based on the PDAG representation of the Bayesian networks. The following distance measures are used:

- **SHD** (Structural Hamming Distance) is defined as the number of the following operations to make the PDAGs match (Tsamardinos, Brown & Aliferis 2006): (1) insert or remove an undirected edge, (2) insert, reverse or remove a directed edge or (3) direct an undirected edge or make a directed edge undirected.
- **USHD** (undirected Structural Hamming Distance) is introduced in this work and is the Structural Hamming Distance without considering mismatches in the direction of edges.
- **TP** (true positives): Number of edges that appear in the true network structure as well as in the learned network structure.
- **FP** (false positives): Number of edges that appear in the learned network structure, but not in the true network structure.
- **FN** (false negatives): Number of edges that appear in the true network structure, but not in the learned one.
- **Sens (sensitivity)**: $\frac{TP}{TP+FN}$.
- **PPV** (positive predictive value): $\frac{TP}{TP+FP}$.

Usually, one uses specificity to identify the fraction of correctly classified negatives. Specificity is defined as $\frac{\sharp\ true\ neg.}{\sharp\ true\ neg.\ +\ \sharp\ false\ pos.}$. However, since the number of true negatives is much greater than the number of false positives in sparse domains, the specificity is almost one. Thus we use in this work PPV instead of specificity. All of the aforementioned measures are calculated for each single subnetwork and are afterwards averaged over all subnetworks, i.e. the measures denote the mean performance of learning a single subnetwork.

In addition, 40 Bayesian networks were learned with all variables included and the same measures calculated. To obtain comparable values for the globally learned networks as well as for the subnetworks, we calculated the measures for the global network in the same way as for the subnetworks. Thus, for each previously learned subnetwork, we selected the corresponding variables in the global network, calculated the measures for each of the

Table 4.3: Performance of subnetwork learning for the Alarm_50 (1850 vars) network with different sizes of the subnetworks (20, 50 and 100). The values for the subnetwork learning are denoted as **S**, while the result of learning networks completely are denoted as **C**.

	20		**50**		**100**	
Measure	**S**	**C**	**S**	**C**	**S**	**C**
SHD	6.4	3.1	13.0	5.7	29.0	11.8
FP	1.2	0.3	3.5	0.5	9.6	0.9
FN	0.7	0.6	1.3	1.4	3.2	3.3
TP	9.3	9.4	16.5	16.4	32.0	31.9
USHD	1.9	1.0	4.9	1.9	12.7	4.2
Sens	.93	.94	.92	.92	.91	.91
PPV	.88	.97	.82	.97	.77	.97

Table 4.4: Same configuration as used in table 4.3, but for the ALL_benchmark_1000 (1000 vars) network

	20		**50**		**100**	
Measure	**S**	**C**	**S**	**C**	**S**	**C**
SHD	8.0	4.0	24.6	13.9	52.6	30.6
FP	3.5	1.1	12.0	4.4	27.5	10.4
FN	1.6	1.8	6.2	6.7	13.9	14.8
TP	12.5	12.3	29.3	28.8	57.8	56.9
USHD	5.1	2.9	18.2	11.1	41.4	25.2
Sens	.89	.87	.83	.81	.81	.79
PPV	.78	.92	.71	.87	.68	.85

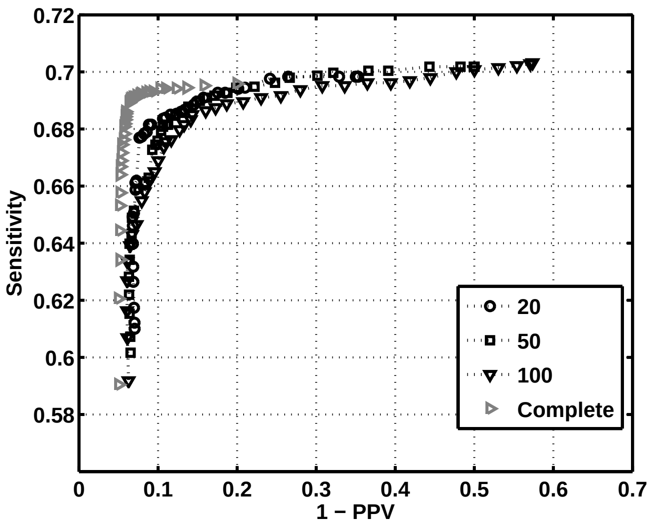
selected regions, and averaged the results. Thus, these measures indicate the performance if networks are learned as a whole, but their values are transformed to be comparable to the subnetwork measures. Tables 4.3 and 4.4 summarize the performance of the subnetwork method by means of the previously mentioned evaluation measures.

One can clearly see that the subnetwork method produces significantly more false positive edges, often two to three times higher compared to the complete network case. The structural hamming distance is much higher, as well. For example, a subnetwork size of 100 for the Alarm_50 network leads to a distance of 29 while there are only 32 true positive edges. This high value results from false positives edges, but also from a considerable number of edges directed in the wrong direction. On the other hand, the number of true positives (TP), false negatives (FN) and the sensitivity values are almost equal in the subnetwork and in the complete case for both benchmark networks. This means, for a set of selected variables, that all edges that can be detected if learning the complete network can also be detected if only a small subnetwork is learned. However, they are sometimes learned with the wrong direction.

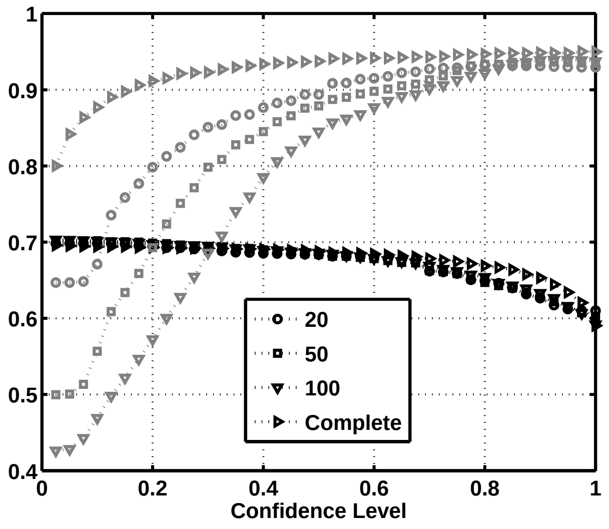
For a further robustness assessment all Alarm_50 subnetworks of equal size are combined to one fPDAG, as are the 40 complete learned networks. Based on these fPDAGs, we calculated the sensitivity values and the positive predictive values (PPV) for several confidence thresholds. In Figure 4.7 (a), the sensitivity values are plotted against the PPV for different subnetwork sizes and the complete network, with better performance indicated by more points in the top left of the graph.

The figure clearly shows that learning subnetworks performs worse than learning the network on a whole. However, if one restricts on those edges that reach a high confidence level, namely a level of 0.8 or above, the difference almost vanishes (see figure 4.7 (b)). Many of the false positive edges that are learned in the subnetwork case have a low confidence in the fPDAG and therefore can be eliminated by introducing a suitable threshold for the confidence. It is remarkable that, for a small confidence threshold, the subnetwork learning with a small subnetwork size (20 in this example) performs much better than with larger network sizes (50 or 100). A reason for this could be that the selection method for one subnetwork—taking variables that are related to one single central variable—is suitable for detecting the structure "around" the central variable, however, edges between other variables very often seem to be false positives since these variables lack their neighborhood required for learning.

The main result of this section is two-fold: First, we have shown that learning a subnetwork leads to around twice as many network errors compared to the case if learning the network completely. This must be considered if results about network structures are based upon subnetworks. Second, if we learn several different subnetworks, the edge confidences can be used as a measure to distinguish between wrong edges and edges that are also learned in the complete case.



(a) Sensitivity against 1-PPV



(b) Sensitivity/PPV against confidence

Figure 4.7: Benchmark results for the Alarm_50 network based on the fPDAG representation of all subnetworks. In the figure, (a) shows the sensitivity against the positive predictive value for several sizes of the subnetworks and the complete case. In (b) the sensitivity values (black) and positive predictive values (grey) are plotted for different levels of the edge confidence.

4.4 Dimensional Bootstrap

In order to assess the quality of a subnetwork, we introduce a new bootstrap approach, called dimensional bootstrap. With dimensional bootstrap one can assess the robustness of edges against missing variables that do not occur in the subnetwork. Especially in very large domains, this bootstrap can support a robust estimation of the dependency structure between a manageable subset of variables.

4.4.1 Method

Based on a set of interest (SOI) of variables the dimensional bootstrap tries to detect true direct relationships between variables of the SOI while false positive edges (as shown in figure 4.1) should be avoided. The SOI, thereby, can be created by a user, by using a statistical criterion as described above or by any other suitable method. Based on the assumption that variables outside the SOI that are highly dependent on variables inside the SOI might have a high impact on the learned structure, dimensional bootstrap dynamically adds missing variables that are highly dependent on variables in the SOI. These variables are added in such a way that, on the one hand, false positives are avoided, and, on the other hand, the size of the network is kept moderately small. The set of variables that is added to the SOI for learning is henceforth referred to as the set of additionals (SOA). After structure learning, only the edges between the variables of the SOI remain in the network, all other edges and the SOA are removed. Thus, all direct relationships inside the SOI remain as edges in the network. However, those indirect relationships between variables of the SOI that can be explained by a cascade of direct relationships passing over the SOA are removed, leading to a lower amount of false positive edges and, thus, to a better estimation of the network structure.

In general, there might be a large number of variables highly related to the SOI and, thus, also a large number of variables that might have an impact on the learned network structure. Adding of all those variables again turns into a computational problem due to the large size of the network. To avoid this problem, we propose a method whose general idea is based upon the bootstrap method (Efron & Tibshirani 1994). However, the approach does not re-sample data points, but re-samples the variables that should be included in the network for structure learning, and with it re-samples the dimensions of the data set. In practice this means a random selection of a few variables as SOA and the combination of these variables with the SOI to learn the structure. This procedure is applied several times, each time with a different SOA. The general algorithm is outlined in algorithm 5.

Algorithm 5: Dimensional Bootstrap Algorithm

```
   Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$, set of interests $\mathbf{SOI}$, number of bootstrap
          replicas $k$
   Output: fPDAG
1  $\mathbf{R}$ := variables that are related to the $\mathbf{SOI}$;
   // Start bootstrapping
2  for $i = 1...k$ do
       // Create set of additionals
3      $\mathbf{SOA}_i$ := choose subset of $\mathbf{R}$ ;
       // Learn network
4      learn Bayesian network $B_i$ with $\mathbf{SOI} \cup \mathbf{SOA}_i$ as variables;
5  end
6  create fPDAG from all $B_i, i = 1...k$;
7  return $fPDAG$
```

Based on the k bootstrap networks B_i, the confidence of an edge is calculated as the fraction of networks in which this edge occurs. Edges that represent direct relationships between variables of the SOI should appear in almost every network independently of the variables in the SOA. Thus, they should be rated with a high confidence level in the fPDAG. On the other hand, false positive edges that are caused by indirect relationships should not appear if the variables that allow the indirect relationships to be explained by direct relationships are integrated into the SOA. If these variables are contained repeatedly in the SOA by means of the bootstrap procedure, the confidence of false positive edges can be decreased. Thus, using an appropriate threshold for the edge confidence should allow the true positive and false positive edges to be separated.

However, if the SOA is sampled from a fairly large set of variables by using an equal probability for each variable to be selected, the dimensional bootstrap approach might fail, because variables, highly related to variables in the SOI, could appear only in a few bootstrap networks. In contrast, the majority of the variables in the SOA might be weakly related to the SOI and, hence, have only marginal influence on the learned network structure. For that reason we present two methods of choosing the SOA in such a way that these negative effects are avoided. Both methods rank all variables that are not part of the SOI according to their dependency on the SOI ($dep(X_i, \mathbf{SOI})$). We define the dependency of a variable X_i on the SOI in the following way: The dependency between this variable X_i

and each variable of the SOI is calculated, and the maximum of all these values is taken as dependency:

$$dep(X_i, \mathbf{SOI}) = \max_{X_j \in \mathbf{SOI}} dep(X_i, X_j) \tag{4.6}$$

All variables that are not included in the SOI (but in **X**) are sorted according to the dependency on the SOI in descending order and stored in **R** (line 1, algorithm 5). We use mutual information as measure for the dependency between two variables.

We investigate two different approaches to select the SOA out of **R** (line 3 in algorithm 5): SOA selection by using a fixed probability and by using a dependency distribution.

SOA selection by using a fixed probability

The first approach goes through the list **R** of variables in descending order. Each variable is selected with a given probability p_{sel} and included in the SOA, and omitted with the probability $1 - p_{sel}$. Variables are added to the SOA until the size of the SOA reaches a given maximum. Let $max\ size\ of\ SOA$ be this maximum size. Since the approach starts with the most dependent variable in the list, each of the $max\ size\ of\ SOA$ most dependent variables is added to the SOA with a probability of p_{sel}. However, the probability of variables at the end of the list **R** to be included in the SOA is very low, if it is assumed that the total number of variables is large compared to the maximum size of the SOA. All the variables in the list that can contribute to the variables in the SOA, ranging from the most dependent variable in the list to the least dependent variable that is added to the SOA, are henceforth denoted as base quantity. The size of this base quantity has an expectation of $\frac{max\ size\ of\ SOA}{p_{sel}}$. That means that mainly the first $\frac{max\ size\ of\ SOA}{p_{sel}}$ variables contribute to the SOA, whereas the probability for the subsequent variables decreases rapidly. Thus, with the parameter p_{sel} one can indirectly control the size of the set of variables that should be used as base quantity for the bootstrap selection of the SOA. Using this approach to define the variability of the SOA raises a question: How should the parameter p_{sel} be set to reach an optimal estimation of the network? Of course, this value should be smaller than 1, since a selection probability of 1 would lead to a relegated dimensional bootstrap and would imply that the bootstrap procedure serves no purpose. The value, however, should be large enough to restrict the base quantity on the relevant variables, avoiding the inclusion of a large amount of variables that have no impact on the structure. Here, a value of 0.5 was found to be useful, motivated by the following simplifying argument: A structural shift caused by one single latent variable is avoided in about 50% of all networks, since this variable is present in about 50% of the learned bootstrap networks. Similarly, a shift caused by two missing variables affects 75% of all networks, because only 25% of all bootstrap networks contain both variables, and so on. Thus, only high confident edges (confidence of 80% or above) are supposed to be robust against hidden variables.

Henceforth we refer to the dimensional bootstrap with this selection method and $p_{sel} = 0.5$ as *DB 0.5*. With another value for p_{sel}, the number is changed respectively.

SOA selection by using the dependency distribution

The second approach for SOA selection makes a direct use of the distribution of the dependency values. This dependency distribution therefore is treated as an unnormalized probability distribution to sample the variables of the SOA. Consequently, the probability for a variable to be selected depends directly on its dependency on the SOI and represents exactly the normalized dependency value of the variable. The normalization factor is the sum of the dependency values of all variables not already contained in the SOA or SOI. As a result, highly dependent variables, which are supposed to be important for the network structure, occur much more often in the SOA compared to only weakly dependent variables. Experimental investigations lead to the finding that having a large amount of variables that are only weakly dependent on the SOI can lead to an SOA with many variables that show no strong dependency on the SOI. Thus, in general, it is practical to restrict the base quantity for the bootstrap selection depending on the maximal size of the SOI. This is evident as the bootstrap procedure has to cover the whole space of dependent variables several times to be able to make a good estimation of the network structure. For example, consider a variable that is not contained in the SOI, but causes one false positive edge. This variable has to occur several times in the SOA to lead to a decreased confidence in this false positive edge. Otherwise the difference between the confidence of the edge in this example and the confidence of true positive edges might be too small to separate between those edges. Hence, in the remainder of this chapter a value for the size of the base quantity that is four times higher than the maximal size of the SOI is chosen. Other values might be suitable as well, but they are not considered in this chapter.

The dimensional bootstrap with this selection method is referred to as *DB D* later on.

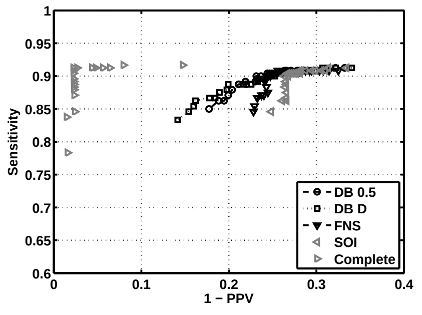
4.4.2 Results

For a performance test of the dimensional bootstrap, the Alarm_50 benchmark network is used. The data set for learning contains 500 observations sampled from the network. Out of the variables in this network we selected three different sets of interest (SOI) with 200 variables each. To generate a SOI, one variable was chosen by chance and its 199 most related variables were added, determined by the value of the mutual information. The results shown in this section are the averaged results of the three cases.

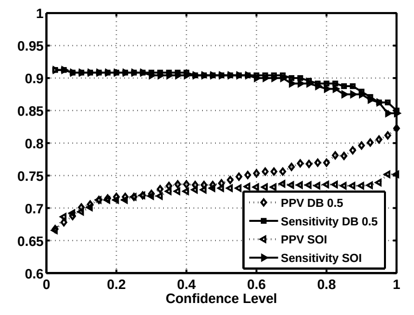
To benchmark the dimensional bootstrap method different settings of learning were chosen. At first, the dimensional bootstrap with a selection probability of 0.5 was used ("DB 0.5"). The alternative method based upon the distribution of the neighborhood is denoted by "DB D". In addition, to enable a suitable comparison to the learning without dimensional bootstrap, the SOA was kept fixed (i.e. the selection probability was set to 1.0). We refer to this setting as "FNS". For the sake of completeness we also show how the learning without SOA performs, including only the set of interest (denoted by "SOI") and the case if the complete network was learned without missing variables ("Complete"). 40 networks were learned by using simulated annealing for each method. Based on the 40 networks, a fPDAG was generated to calculate the edge confidences.

As performance criteria, the sensitivity and the positive predictive values (PPV) of the network dependent on the confidence threshold are calculated. In figure 4.8(a), the resulting sensitivity values are plotted against the PPV for a size of 20 for the SOA. One can clearly see that the method only learning the set of interest ("SOI") is outperformed by any other method, i.e. the PPV is markedly smaller for a given sensitivity level. With a fixed SOA ("FNS" method), the PPV increases, however it is markedly outperformed by both bootstrap approaches ("DB 0.5" and "DB D"). With the best method, in this case the "DB D", one can achieve a sensitivity of 0.85 with a specificity up to 0.84. This means the approach is able to detect 85% of all edges with only about 15% false positives, compared to 25% false positives in the "SOI" case. With an increasing size of the SOA (figure 4.8(c) and figure 4.8(e)), all three methods that depend on the size of the SOA perform better. However, both dimensional bootstrap methods outperform the "FNS" method in all cases.

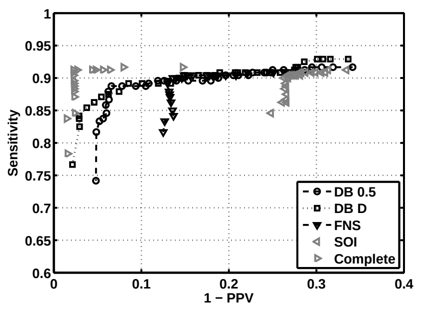
On the right hand side of figure 4.8 the sensitivity (squares) and the PPV (diamonds) of the "DB 0.5" method are plotted against several confidence thresholds. For comparison, the sensitivity (right-pointing triangles) and PPV (left-pointing triangles) of the "SOI" case are plotted in addition. One can clearly see that the dimensional bootstrap method substantially



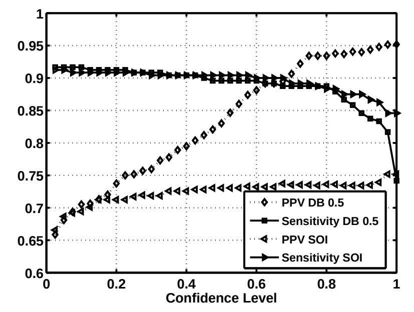
(a) SOA size: 20



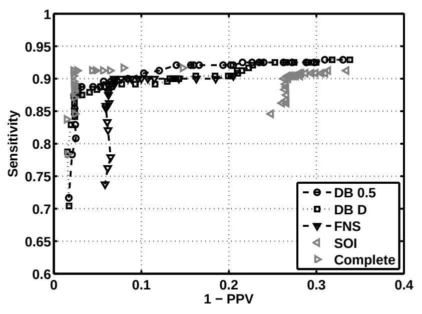
(b) SOA size: 20



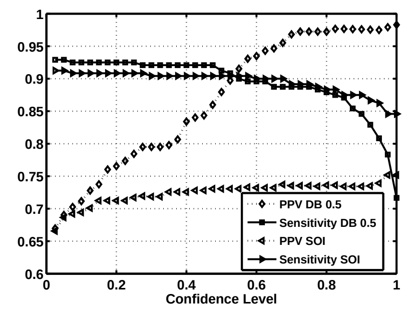
(c) SOA size: 100



(d) SOA size: 100



(e) SOA size: 200



(f) SOA size: 200

Figure 4.8: Benchmark results for the dimensional bootstrap method for different sizes of the SOA. For the dimensional bootstrap (DB) a selection probability of 0.5 (“DB 0.5”) is used, the method based on the dependency distribution is denoted by “DB D”. The left hand side shows the sensitivity against the positive predictive value (PPV). The right hand side illustrates the sensitivity and PPV plotted against the confidence threshold, and for comparison, the corresponding results for the simple subnetwork case (“SOI”).

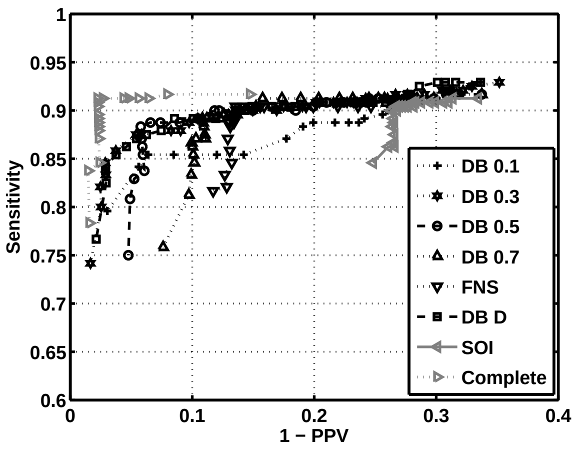


Figure 4.9: Same plot as shown in figure 4.8(c), additionally results for different values of the selection probability are included. The size of the SOA amounts to 100.

outperforms the simple case of learning a subnetwork especially for high values (0.8 and above) for the edge confidence. While the sensitivity changes only marginally, the PPV increases dramatically in the bootstrap case, particularly for large sizes of the SOA (see figure 4.8(d) and figure 4.8(f)).

For all tests considered here we used a probability of 0.5 to select the SOA. We tested the dimensional bootstrap method with different values for the selection probability, as well. Figure 4.9 shows the results for a SOA size of 100. It can be seen that the results with highest positive predictive values are achieved with a value for the selection probability of 0.3. With lower values, the sensitivity decreases noticeable, however with higher values the positive predictive value is reduced markedly. Results with a value of 0.5 show a good tradeoff between sensitivity and PPV, however, the maximum PPV is not as good as with a value of 0.3.

To see the plausibility of this result, we show in figure 4.10 the dependency value of all variables that are not contained in the SOI, ranked by their dependency on the SOI. Since we have three different SOIs, we show the dependency for each SOI separately as different curve in the plot. One can see that only few variables are highly dependent on the SOI. The dependency values decrease nearly exponentially. A set of about 100 variables covers already all highly related variables (marked by the left vertical line), exactly this set was used to learn the networks for the “FNS” benchmark before. But the previous experiments

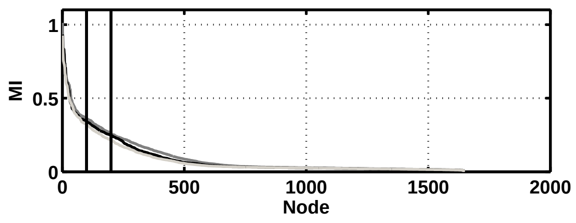


Figure 4.10: Dependency of the SOA on the SOI. Each single variable in **R** is represented on the x-axis (using the index of the variable in **R**). On the y-axis, the dependency value is shown (calculated according to equation 4.6). Since three different SOIs were selected for the tests, each of the dependency distributions is plotted in a separate curve.

on the Alarm_50 network have shown that with a larger base ground for the dimensional bootstrap one can achieve better results. For example, if the selection probability is set to 0.5, the expectation for the size of the base ground is 200 (marked by the right vertical line), with a value of 0.3 we reach a size of 333. It seems that this set is sufficient for a good performance for the dimensional bootstrap. It is small enough for an exhaustive inclusion of all variables in the bootstrap, on the other hand it is large enough to contain mostly all important and relevant variables. With further increasing sizes of the base ground, the performance decreases. This stands to reason that the base ground is too large to be covered sufficiently by the bootstrap.

Since there is no perfect borderline that can be seen in figure 4.10 for a determination of the size of the base ground and therewith of the selection probability, we propose to use the "DB D" method that adopts automatically to the neighbourhood distribution of the SOI.

4.5 Summary

Graphical models, and in particular Bayesian networks, are very useful models for inferring genetic networks by learning the dependency structure from microarray data. However, when learning BN structures in large-scale domains such as the genetic network one faces the problem of $\mathcal{NP}$-hardness. Thus, microarray data sets are usually reduced in their dimensionality by ranking the genes according to their statistical dependency, and network learning is applied on a small subset of potentially dependent genes (Friedman et al. 2000, Dejori & Stetter 2003, Dejori et al. 2004). We evaluated the robustness of

Bayesian network learning if such a reduction in dimensionality is applied by means of artificial benchmark data.

Initially, we investigated the additional influence of sparse data on the robustness of edge features. A common approach to deal with the fluctuations in sparse data is the non-parametric data bootstrap (Efron & Tibshirani 1994), which assesses those dependencies that are caused by the underlying true dependency structure between the variables and not by statistical fluctuations in the data (Friedman, Goldszmidt & Wyner 1999, Friedman et al. 2000). To estimate the influence of the variable-preselection on this assessment, data bootstrap with a surrogate data analysis were combined. In the surrogate data set, all true dependencies between the variables are destroyed. Thus, dependencies that do occur in data are caused by statistical fluctuations. We defined a measure, called predictive value, indicating the fraction of edges caused by true patterns in the data but not by fluctuations. Results from benchmark cases show that these fluctuations can have a significant influence on the learned network structure. While edges with a low confidence might not represent true dependencies but also fluctuations, edges with a high confidence are likely to represent true dependencies in the data. For a tradeoff between the number of detected relationships and the fraction of false dependencies, one can choose a predictive value leading to a threshold for the edge confidence that fulfills the given predictive value. Based on benchmark networks, we have shown that the reduction of dimensionality influences the number of false positive edges and thus the quality of the learned network in a negative manner. We suppose to apply the here presented surrogate data analysis every time if a reduction of dimensionality is applied. This allows to measure the influence of fluctuations in the data if the dimensionality is reduced.

We also used subnetwork learning as an approach to benchmark the learning of subnetworks in a structured way. Therefore, for each variable in the complete network one learns a subnetwork, where the variables in the subnetwork are determined by their dependency on the central variable. Results from a benchmark case show that the fPDAG formed by all subnetworks is a reasonable representation of the true underlying network structure, if only edges with high confidence are considered. However, each single subnetwork contains several false positive edges and many edges with a wrong direction, on average. While such false positive edges can be neglected in the fPDAG because of their low confidence, they can disrupt the network of a single subnetwork. Although these results can probably be transferred to networks that have a similar topology as the Alarm benchmark network used here, the robustness of learning subnetworks should be benchmarked on other networks with the subnetwork algorithm too, because networks with different topological features can also have a different performance. For example, the subnetwork method is known for not being applicable to fully or almost fully connected networks, because the variables cannot be reasonably divided into subnetworks. However, most highly dimensional network structures are sparse. For instance, genetic networks are supposed to be scale-free (Jeong, Tombor, Albert, Oltvai & Barabási 2000, Barabási & Bonabeau 2003) and are therefore only sparsely linked.

To avoid on the one hand disruptions arising from learning subnetworks, but on the other hand to enable networks with moderate size without learning the complete network, the dimensional bootstrap approach was introduced. Thereby the network structure between a set of variables (set of interest: SOI) is estimated by learning several networks containing the SOI and randomly chosen variables (SOA) that are highly dependent on variables contained in the SOI. At each iteration, a different set of dependent variables (SOA) is combined with the SOI for learning. It has been shown by means of a benchmark network that networks estimated by the dimensional bootstrap procedure are much closer to the original network structure than networks learned only based on the SOI. The bootstrap networks also have a better structural estimation than networks of same size with a fixed set of dependent variables. Particularly with regard to very large domains where the reduction in dimensionality is commonly used, the dimensional bootstrap approach can boost a more robust estimation of subparts in the complete dependency structure.

5 Large-Scale Network Learning

For the task to learn in large domains, reducing the size of the network is a common approach in order to minimize the computational time for the learning procedure. As stated before, this approach is often used in the literature. In the previous chapter we investigated the shortcomings of this technique regarding the quality of the learned network. As we have shown, the reduction of dimensionality comes along with a set of problems that lead to a decrease of the quality of the learned network. Except the dimensional bootstrap method, we focused more on measuring the network disruptions introduced by the downscaling instead of investigating new learning methods to obtain high-quality network structures in high-dimensional spaces.

In this chapter we go one step further and present two new learning algorithms that tackle the problem of the network quality paired with high-dimensional feature spaces: Based on a more profound feature selection algorithm introduced in Tsamardinos, Aliferis & Statnikov (2003), we follow our idea of subnetwork learning and estimate the total network by learning small subnetworks. The first algorithm, called *substructure algorithm* as introduced in Nägele et al. (2007), utilizes subnetwork learning to estimate edges and their direction on the basis of a feature graph (fPDAG). Substructure learning can be used in high-dimensional spaces to estimate selective parts of a network without learning the network as a whole. Since one can restrict the learning procedure on a small part of the network, this method allows to estimate network areas that are of major interest with a guaranteed computational complexity.

While this method leads to a pure structural estimation of the network on the basis of small substructures, our second approach, called S-DAG as introduced in Nägele et al. (2008), builds a complete Bayesian network on the basis of all substructures. We show that this divide-and-conquer approach to learn DAG structures for large networks leads to performant and high-quality BN reconstructions, outperforming state-of-the-art BN network learning algorithms.

5.1 Background

A quite competitive algorithm to learn BNs that combines constraint based and score based approaches was developed (Tsamardinos, Brown & Aliferis 2006). The so called MMHC (max-min hill-climbing) algorithm performs Bayesian network learning in two steps: first, an undirected network skeleton $\mathcal{S}$ is estimated with MMPC (max-min parents and children) that employs constraint-based techniques (Tsamardinos et al. 2003). Afterwards, a score-based greedy search is performed to orient the edges.

MMPC is a local discovery algorithm to assess the set of parents and children $\mathbf{PC}_i$ of a variable X_i. The assessment is done in two phases: First, variables conditionally dependent on X_i are added to a candidate set of parents and children. Just these variables are added which are conditionally dependent on X_i, given the current set of candidate parents and children. Thereby, the so-called Max-Min heuristic is used (Tsamardinos et al. 2003): this variable is taken that maximizes the minimum association with X_i relative to candidate set. The basis of this heuristic is the idea to add this variable to the candidate set that is most unlikely conditionally independent from X_i. But due to this heuristic it is possible that false positives enter the candidate set. Thus, in the second phase, these false positives are removed. False positive variables are those variables that are conditionally independent of X_i relative to any subset of the candidate parents and children.

A skeleton is constructed by performing MMPC on all variables. Undirected edges between all variables of the set of parents and children $\mathbf{PC}_i$ and X_i are added to the skeleton. This procedure is repeated for every variable X_i in the network. To avoid false positives, the relationship between X_i and $\mathbf{PC}_i$ must be symmetric. If not, variables are removed from $\mathbf{PC}_i$ until the relation is symmetric. If faithfulness is assumed, the skeleton contains also no false negative edges. For more details we direct the interested reader to appendix A.1 or Tsamardinos, Brown & Aliferis (2006).

It has been shown that the MMHC algorithm has a good performance in terms of quality as well as runtime and outperforms many state-of-the-art BN learning algorithms such as the sparse-candidate algorithm (Friedman, Nachman & Pe'er 1999, Tsamardinos, Brown & Aliferis 2006). The MMHC algorithm is described in more detail in section 3.3.1 (algorithm 1).

Algorithm 6: Substructure Algorithm

```
Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$
Output: set of Bayesian subnetworks $\mathbf{B}$
// Skeleton reconstruction
1 create skeleton $\mathcal{S}$ by MMPC ;
// structure learning
2 $\mathbf{B}$ := SUBSTRUCTURE_LOC($\mathbf{D}, \mathbf{X}, \mathcal{S}$) ;
3 return set $\mathbf{B}$
```

5.2 Substructure Learning

In this section we introduce substructure learning as an efficient and scalable method for estimating the structural dependencies in large and sparse domains.

5.2.1 Algorithm

Based on the idea of reducing the dimensionality and instead of learning the whole network, we learn a set of small subnetworks that together resemble the original global structure with high accuracy. The algorithm itself is a two-step process (see algorithm 6): First, the skeleton $\mathcal{S}$ of the complete network structure is reconstructed. Second, small subnetworks are learned independently of each other for an estimation of the complete network structure (see algorithm 7). The new approach of substructure learning consists in estimating the complete network structure by learning several subnetworks, one for each variable in the complete network.

The first phase of our algorithm (line 1, algorithm 6) is identical to the first phase of MMHC and determines the set of parents and children $\mathbf{PC}_i$ of each variable X_i to reconstruct the skeleton $\mathcal{S}$ of the complete network by means of MMPC. In the second phase (line 2), the subroutine SUBSTRUCTURE_LOC to calculate the substructures is called. This routine (algorithm 7) calculates one subnetwork (also called substructure) for each variable in the complete network. Therefore, a set of related variables $\mathbf{M}_i$ is determined by NEIGHBOURHOOD(X_i,$\mathcal{S}$) for each variable X_i: To detect related variables within the NEIGHBOURHOOD routine, we introduce a variable selection method that determines variables centered "around" the variable X_i in the resulting graph. Manual tests have shown that at least all variables that belong to the Markov blanket are important to reconstruct the network structure around a variable. As defined in chapter 2.1.2, the Markov blanket of a

```
Algorithm 7: SUBSTRUCTURE_LOC
  Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$, skeleton $\mathcal{S}$
  Output: set of Bayesian subnetworks $\mathbf{B}$
1 foreach variable $X_i \in \mathbf{X}$ do
      // Create Neighbourhood of $X_i$
2     $\mathbf{M}_i$ := NEIGHBOURHOOD($X_i$,$\mathcal{S}$) ;
3     $\mathbf{D}_{\mathbf{M}_i}$ := restrict data $\mathbf{D}$ on variables in $\mathbf{M}_i$ ;
      // Create subnetwork around $X_i$
4     $B_i$ := LEARN_BN($\mathbf{M}_i, \mathbf{D}_{\mathbf{M}_i}, \mathcal{S}$) ;
5     $B_i$ := restrict $B_i$ on $X_i$ and the Markov blanket of $X_i$ ;
6     $\mathbf{B}$ := $\mathbf{B} \cup B_i$ ;
7 end
8 return $\mathbf{B}$
```

variable is a subset of variables that render this variable independent from all others. In a BN, the Markov blanket of a variable consists of its parents, its children and the parents of its children.

To detect variables that belong to the Markov blanket, we utilize the skeleton $\mathcal{S}$ and the neighbourhood of variables regarding the undirected network structure of $\mathcal{S}$. To include all variables of the Markov blanket, one has to include the neighbours of X_i in $\mathcal{S}$ and their neighbours. Regarding $\mathcal{S}$, this is the minimum set of variables that contain all variables of the Markov blanket. Thus, the central variable X_i of the local structure, the parents and children of X_i and their parents and children are all put together for learning one single BN. This variable selection is the first crucial step since a suboptimal selection with missing variables which are structurally important can lead to false positives as well as false negatives, as it has been shown in chapter 4.

The second crucial step is the learning of the local Bayesian subnetworks. As done for MMHC, we restrict edges in the subnetwork to edges that also appear (as undirected edges) in the skeleton, this means an edge between two variables can only be added during structure search if the variables are also connected in the skeleton. To increase the quality of the network estimation we afterwards restrict the learned subnetwork to the Markov blanket of X_i by removing all variables and edges that do not belong to the Markov blanket or X_i itself. The result of the substructure algorithm is the set $\mathbf{B}$, containing all local Bayesian subnetworks B_i, one for each variable. All the local subnetworks allow a structural estimation of the complete DAG and, as well, build a quantitative model for each single variable given its Markov blanket, encoded as a BN.

The set of partially overlapping subnetworks **B** lacks of a unifying representation of the network structure. Thus we use the fPDAG representation for the set of subnetworks **B** as introduced in section 4.1.3.

5.2.2 Time Complexity of Substructure Learning

In the first phase of the substructure algorithm, the skeleton of the underlying dependency structure is reconstructed using MMPC. The complexity of MMPC is $O(|\mathbf{X}|2^{|\mathbf{PC}_i|})$ according to Tsamardinos, Brown & Aliferis (2006). Since a minimum amount of samples for conditional independence tests is needed, the algorithm tests the independence not for all possible subsets of $\mathbf{PC}_i$ (algorithm 11, lines 4 and 12), but only for some of the possible conditioning subsets. Then, each single call on MMPC has a computational complexity of $O(|\mathbf{X}||\mathbf{PC}|^{l+1})$ with l as the maximum size of all conditioning subsets. Thus, the overall cost for reconstructing the whole skeleton is $O(|\mathbf{X}|^2|\mathbf{PC}|^{l+1})$, where $|\mathbf{PC}|$ is $\max_{X_i \in \mathbf{X}} \mid \mathbf{PC}_i \mid$. If l is restricted to a maximum (and fixed) number, MMPC performs polynomial in $|\mathbf{X}|$, while MMPC has worst-case complexity $\mathcal{NP}$ (we refer to (Tsamardinos, Brown & Aliferis 2006) for more details). So far, the substructure algorithm does not differ from MMHC. However, in the edge orientation phase substructure learning splits the structure search problem into several small subproblems.

We now estimate the influence of this splitting on the number of possible network structures. The learning problem with a given skeleton where each variable has at least two neighbours is similar to the problem of having a minimum number of parents of 2. Finding the best DAG in this setting is $\mathcal{NP}$-hard in the number of variables (Chickering et al. 1994, Friedman, Nachman & Pe'er 1999). Thus, learning one subnetwork is $\mathcal{NP}$-hard in $|\mathbf{PC}|^2$, since $|\mathbf{PC}|^2$ is an upper bound for the number of variables in one subnetwork. This means, if $|\mathbf{PC}|$ is much smaller than the number of all variables, the substructure approach dramatically reduces the number of possible network structures. This affects the performance of heuristic search strategies like hill climbing as well. For an estimation of the impact, we define the cost of a search strategy, depending on the maximum number of parents and children and the size of the domain, as $f(|\mathbf{PC}|, |\mathbf{X}|)$. For one subnetwork, the cost becomes $f(|\mathbf{PC}|, |\mathbf{PC}|^2)$. Thus, the overall cost for the second phase of substructure learning is $|\mathbf{X}|f(|\mathbf{PC}|, |\mathbf{PC}|^2)$. If we restrict $|\mathbf{PC}|$ to a fixed value, the second phase performs even linearly in the number of variables. For small networks however, we expect substructure learning to be significantly slower than MMHC. For instance, the edge orientation phase for the network presented in figure 4.1 must be learned 7 times with the substructure algorithm, but only once with MMHC.

5.2.3 Results

In this section, we empirically benchmark the substructure algorithm by a comparison to MMHC. We use only MMHC for comparison as it has been shown in Tsamardinos, Brown & Aliferis (2006) that MMHC outperforms many other structure learning algorithms in terms of accuracy and time efficiency. The study of Tsamardinos, Brown & Aliferis (2006) included the algorithms optimal reinsertion (OR) (Moore & Wong 2003), sparse candidate (Friedman, Nachman & Pe'er 1999), greedy hill climbing (Heckerman et al. 1995), PC (Spirtes et al. 2001), three phase dependency analysis (Cheng 2002) and greedy equivalent search (Chickering 2002). For all of them they have shown that MMHC outperforms these algorithms in terms of time efficiency and quality, on average (Tsamardinos, Brown & Aliferis 2006). We do not compare substructure learning to other algorithms, for instance to the other algorithms presented in section 3.3. We do this comparison in more detail with our new method S-DAG which is based upon substructure learning. This method is introduced in the next section.

For the benchmark, we sample training data from known benchmark networks and request both algorithms to reconstruct the original network structures. These reconstructed networks are then compared to the original network to assess the quality of the learned structures. As benchmark networks we have chosen the networks Alarm_10, Alarm_20, Alarm_30, Insurance_10, Insurance_20 and Insurance_30 (or abbreviated: A._10, ..., I._30). From each of the benchmark networks we sampled data sets of different sizes (100, 200, 500, 1000 and 5000 samples). More details about the networks can be found in appendix A.2.

For the structure learning part of the substructure algorithm we use random hill climbing as heuristic search method. This means we select randomly two variables, calculate the scores for arc addition, arc removal and arc reversal and apply the local change with the highest score until no action can improve the total score. As scoring function that solves 3.3, we use the BDeu score (Heckerman et al. 1995) with an equivalent sample size of ten. For the MMHC algorithm we use the implementation of the original authors from the Causal Explorer software package (Aliferis, Tsamardinos, Statnikov & Brown 2003). For the DAG search, they implemented greedy hill climbing and used the BDeu score with an equivalent sample size of ten, as well. The approach presented here is focused on optimally reconstructing the original structure. Thus, we assess the accuracy by using evaluation measures that are based on structural features only. Other quality measures that take the density distribution into account are not considered here. As first evaluation measure we use the Structural Hamming Distance (SHD) (Tsamardinos, Brown & Aliferis 2006). For feature graphs (fPDAGs), we extend the definition that was given in section 4.3.2 in such a way that each operation counts not as one but as the confidence of the corresponding feature. Additionally, we use the number of false positives (FP) and false negatives (FN) to benchmark the algorithm. The false negatives and false positives are also counted by the confidence of the corresponding feature. For runtime comparisons we use the real-time of both algorithms in seconds on a computer with an Intel Pentium M processor, 2 GHz, and two GB working memory.

We have theoretically shown that under some circumstances substructure learning has a better computational complexity for the edge orientation phase than MMHC, but it is not proven that substructure learning performs faster than MMHC in reasonable large domains. Thus we compare the runtime of the MMHC algorithm with the substructure algorithm for several network sizes. Since we base on the MMHC implementation of Aliferis, Tsamardinos, Statnikov & Brown (2003) (in the Causal Explorer software package) for the edge orientation phase, but use our own implementation for the substructure algorithm, the difference between the runtime might be caused by different implementations of the algorithms.

As a result, the runtimes presented in this section can not be used as a hard criterion for comparing both algorithms. But they show quite reasonable the scalability of substructure learning for large networks. With a better implementation of the MMHC algorithm, we expect the border for the network size where the substructure algorithm performs faster than MMHC to be markedly shifted towards larger network sizes, even in such a way that there can not be seen any performance gain for network sizes used in this section.

Table 5.1 shows the runtime performance of substructure learning compared to MMHC for different sample sizes (500, 1000 and 5000 samples) and networks. We provide the absolute values in seconds (column ’Absolute’) as well as the normalized performance (column ’Norm.’). This means that we divided each measure for substructure learning by the corresponding measure for MMHC. Thus, a value smaller than 1 denotes that substructure learning performs better than MMHC. The ’Absolute’ numbers denote the original,

Table 5.1: Runtime results for different networks and sample sizes

	500		1000		5000	
	Norm.	Absolute	Norm.	Absolute	Norm.	Absolute
Alarm	1.22	5.43	1.23	7.31	1.30	26.4
Alarm_10	**0.64**	162.8	**0.73**	228.3	**0.85**	862.1
Alarm_20	**0.40**	582.9	**0.45**	802.4	–	–
Alarm_30	**0.24**	1265	**0.33**	1741	–	–
Insurance	1.18	5.1	1.09	7.66	1.11	57.2
Insurance_10	**0.78**	129.3	**0.88**	199.6	**0.98**	1348
Insurance_20	**0.54**	398.5	**0.65**	598.0	–	–
Insurance_30	**0.39**	815.7	**0.45**	1202	–	–

Table 5.2: Structural Hamming Distance results for different networks and sample sizes

	500		1000		5000	
	Norm.	Absolute	Norm.	Absolute	Norm.	Absolute
Alarm	1.25	26.2	1.03	16.4	1.85	18.5
Alarm_10	1.01	382.4	**0.99**	314.5	1.10	253.9
Alarm_20	**0.91**	742.8	**0.89**	620.9	–	–
Alarm_30	**0.88**	1066	**0.85**	867.0	–	–
Insurance	1.00	42.1	**0.90**	36.1	**0.92**	34.1
Insurance_10	1.11	405.0	1.04	327.1	1.08	201.6
Insurance_20	1.03	757.0	1.01	592.3	–	–
Insurance_30	1.02	1137	**0.97**	885.2	–	–

Table 5.3: Average normalized performance results

Network	Size	Edges	Runtime	SHD	FP	FN
Alarm	37	46	1.32	1.37	**0.98**	1.31
Alarm_10	370	570	**0.74**	1.21	**0.81**	1.06
Alarm_20*	740	1101	**0.43**	**0.90**	**0.32**	1.09
Alarm_30*	1110	1580	**0.29**	**0.86**	**0.31**	1.08
Insurance	27	52	1.31	1.00	1.04	1.09
Insurance_10	270	556	1.27	1.15	1.40	1.04
Insurance_20*	540	1074	**0.59**	1.02	**0.86**	1.03
Insurance_30*	810	1619	**0.42**	**0.99**	**0.87**	1.01

not normalized measures for substructure learning. For small networks the substructure algorithm shows a performance that is worse than the performance of MMHC. E.g. the substructure algorithm takes around 1.2 times the runtime of MMHC for the Alarm network with 500 samples. As MMHC shows better runtime results for small networks, the reduced complexity of substructure learning shows its advantage for larger networks: For the Insurance_30 benchmark case, substructure learning needs only about 40 % of MMHC's runtime, while for the largest Alarm network only about 30 % of the runtime is needed. We also tried to learn larger networks, thus we created a tiled Alarm network with 1850 variables and 2853 edges. MMHC failed to learn the complete network within one day (we interrupted the algorithm because of time issues), while substructure learning reconstructed the whole network within 255 minutes with a hamming distance of 1378, 88 false positives, 661 false negatives and 1564 correctly identified edges.

While the pure runtime might show a wrong picture, the normalized runtime clearly shows that the substructure algorithm has a better scalability than MMHC in terms of runtime.

A fact that is more reliable than the runtime is the performance in terms of the quality of the reconstructed network. In table 5.2, the Structural Hamming Distance between the original network and the reconstructed network is presented. As for the runtime, we show the absolute values as well as the normalized values. A normalized value smaller than 1 means that substructure learning has a better network reconstruction quality (less false edges and/or less missing edges) than MMHC. For small networks, MMHC has better results than substructure learning. The relatively small Alarm network is reconstructed poorly for 5000 samples with a normalized hamming distance of 1.85 by substructure learning. For all other cases, however, the Structural Hamming Distances are comparable for both approaches, in some cases substructure learning even outperforms MMHC. An important fact is that the relative performance of the substructure algorithm increases with the network size. This is apparent when one compares the average performance over all sample sizes for one network size.

Thus, we report in table 5.3 the network-wise averaged values over all sample sizes (100, 200, 500, 1000 and 5000 samples). Some of the networks (denoted by an asterisk in the table) are only learned with 500 and 1000 samples due to the large amount of time needed for one network reconstruction. As we can see, the substructure algorithm generally shows a good performance in terms of runtime and network quality compared to MMHC, especially for large networks. As larger the network is, as better the quality of the reconstructed network. For the large Alarm networks, substructure learning clearly outperforms MMHC. Besides SHD, we also show the normalized number of false positives (FP) and false negative edges (FN). While there are slightly more false negatives (see table 5.3), the number of false positives are even less in most substructure networks. This indicates that the way of shrinking the subnetworks to the Markov blanket of the central variable favors a higher number of false negatives than leading to false positive edges.

To draw a conclusion, the substructure algorithm has a higher runtime and produces networks with less quality if the networks are small. However, we have exemplary shown with two different types of benchmark data (Alarm and Insurance) that substructure learning outperforms MMHC in terms of runtime performance and quality if applied on larger networks with around 500 variables.

5.2.4 Discussion

Many other approaches for efficient network learning optimize the search procedure to find a good DAG by utilizing the sparseness of the structure. An algorithm that deals with domains up to hundreds of thousands of variables was introduced by Goldenberg & Moore (2004). However, it restricts BNs on binary variables paired with very sparsely linked graphs. Another approach that is closely related to MMHC was introduced in Brown, Tsamardinos & Aliferis (2005). While, in the worst case, the skeleton reconstruction phase using MMPC can have an exponential cost, they developed an polynomial algorithm (called PMMS) for learning the skeleton. While we did not use PMMS for the substructure algorithm, we included this algorithm for the S-DAG algorithm presented in section 5.3.

Since substructure learning detects the Markov blanket for each variable and thus renders this variable independent from all other variables given the Markov blanket, it can also be seen as a feature selection algorithm. In Tsamardinos et al. (2003) a variation of MMPC is developed that estimates the Markov blanket using conditional independence tests. A comparison of different other approaches can be found in Aliferis, Tsamardinos & Statnikov (2003). However, these methods return only the set of variables that belong to the Markov blanket, without discovering the probability distribution and its underlying network structure.

Another approach that is somehow related to our work is the framework of dependency networks (Heckerman, Chickering, Meek, Rounthwaite & Kadie 2001). There, the joint probability distribution is defined by a set of conditional probabilities. Unlike BNs where the conditional probability of a variable is defined given its parents, the conditional probability for each variable is determined by the complete Markov blanket. Subnetworks, resulting from substructure learning, can be easily transformed into a dependency network: The conditional probability of variable X_i is given by the joint distribution of subnetwork B_i, conditioned on the Markov blanket of X_i. For inference in a dependency network, the original authors have introduced a Gibbs sampling method. Since subnetworks can be transformed into dependency networks, this inference method can also be applied to subnetworks.

5.3 Substructure DAG Learning

We have shown in the previous section that substructure learning is a scalable method to learn high-dimensional network structures with high quality, for some networks even better than MMHC which on its part outperforms most others BN learning algorithms (Tsamardinos, Brown & Aliferis 2006). However, there is a main drawback of substructure learning: The result of the method is a set of independent Bayesian networks and the structural result can only be presented using the fPDAG framework. Learning a single DAG would be preferential because many useful tools, such as exact inference, require standard Bayesian networks. The problem of merging the subnetworks to a single Bayesian network was avoided in the last section. In this section however, we present a new algorithm, called S-DAG (Substructure-DAG), as an extension to the substructure algorithm (Nägele et al. 2008). While substructure learning produces a set of independent BNs, S-DAG uses these BNs, combines them to a large graph and removes edges until the structure is a directed acyclic graph (DAG). The acyclicity of the network structure is important since the structure of a Bayesian network must be always a DAG. S-DAG scales polynomially (under the assumption that the maximum number of parents and children are fixed), thus being able to learn networks with tens of thousands of variables.

5.3.1 Algorithm

The substructure algorithm introduced in section 5.2 performs Bayesian network learning in two steps: first, an undirected network skeleton is estimated with MMPC (max-min parents and children) that employs constraint-based techniques. Afterwards, a score-based greedy search is performed to orient the edges. The first step with the MMPC algorithm

Algorithm 8: S-DAG

```
Input: data set $\mathbf{D}$, set of variables $\mathbf{X}$
Output: DAG $\mathcal{G}$
// skeleton reconstruction
1 create skeleton $\mathcal{S}$;
// learn substructures
2 $\mathbf{B}$ := SUBSTRUCTURE_LOC($\mathbf{D}, \mathbf{X}, \mathcal{S}$) ;
3 $\mathcal{G}$ := graph with nodes $\mathbf{X}$ and all edges in $\mathbf{B}$ ;
// remove cycles
4 $\mathcal{G}$ := REMOVE_CYCLES($\mathcal{G}, \mathbf{D}$) ;
5 $\mathcal{G}$ := LEARN_BN_SKELETON_GREEDY($\mathbf{X}, \mathbf{D}, \mathcal{S}, \mathcal{G}$) ;
6 return $\mathcal{G}$
```

has a worst-case complexity of $\mathcal{NP}$ (Tsamardinos, Brown & Aliferis 2006), which is a serious drawback in very large domains. To overcome this issue, besides MMPC we use a slightly different version of the MMPC algorithm, called PMMS (Polynomial Max-Min Skeleton), which has a polynomial complexity (Brown et al. 2005). This improvement in performance is achieved by replacing the exhaustive search of the minimal conditional dependency of two variables by a greedy search (see Brown et al. (2005) for more details). However, PMMS does not guarantee to return no false positives, since some true conditional independencies may not be detected by the greedy search.

Our S-DAG algorithm which performs the BN structure (DAG) search in three phases is outlined in algorithm 8. The first two phases are similar to substructure learning (see algorithm 6): First, the undirected skeleton is reconstructed (line 1). Based on the skeleton, we learn the directed structure around each variable with SUBSTRUCTURE_LOC (algorithm 7) in the second phase (line 2) and merge all networks together (line 3). Finally, in the third phase, all cycles that might have been introduced in the second phase are removed (line 4), followed by a greedy optimization at the end (line 5).

In the following, the single steps of S-DAG are explained in more detail:

Skeleton reconstruction (line 1): We have two variants of the S-DAG algorithm: S-DAG PMMS uses the polynomial variant PMMS for the skeleton reconstruction, while S-DAG MMPC uses MMPC and is therefore directly comparable to MMHC since both base on the same skeleton reconstruction method. We refer to S-DAG if the general algorithm is meant and use the terms S-DAG PMMS and S-DAG MMPC if the algorithm including a specific skeleton reconstruction method is meant.

Algorithm 9: LEARN_BN_SKELETON_GREEDY

```
Input: set of variables $\mathbf{X}$, data set $\mathbf{D}$, set of possible edges (Skeleton) $\mathcal{S}$, initial
       structure $\mathcal{G}$
Output: DAG $\mathcal{G}$

1  if $\mathcal{G}$ not given then
       // initialize with empty graph
2      $\mathcal{G} = (\mathbf{X}, \emptyset)$;
3  end
4  $\mathcal{G}_{tmp}$ = copy of $\mathcal{G}$;
5  $TL$ : tabu list (FIFO) with last 100 structures;
6  repeat
       // actions can be add-edge, remove-edge, revert-edge; only
          try add-edge if edge exists in $\mathcal{S}$; action may not cause
          cycles
7      choose best action for $\mathcal{G}_{tmp}$ with resulting graph not in $TL$;
8      apply best action to $\mathcal{G}_{tmp}$;
9      add $\mathcal{G}_{tmp}$ to $TL$;
10     $\Delta S$: score $\mathcal{G}_{tmp}$ - score $\mathcal{G}$;
11     if ($\Delta S > 0$) then
12         $\mathcal{G}$ = copy of $\mathcal{G}_{tmp}$;
13     end
14 until $\mathcal{G}$ has not changed last 5 times ;
15 return $\mathcal{G}$
```

SUBSTRUCTURE_LOC (line 2): Based on the skeleton, a Bayesian network is learned around each variable with the SUBSTRUCTURE_LOC algorithm that was introduced in section 5.2 (algorithm 7). This algorithm returns a set **B** of Bayesian networks, one for each of the variables in **X**. Instead of using the original SUBSTRUCTURE_LOC algorithm, we replace the structure search algorithm LEARN_BN (line 4, algorithm 7) with the method LEARN_BN_SKELETON_GREEDY (algorithm 9). This method implements a greedy search and is almost identical to the standard Greedy Search (algorithm 1) described in section 3.3.1. The difference is that SUBSTRUCTURE_LOC accepts an initial structure $\mathcal{G}$ for the Bayesian network structure. If $\mathcal{G}$ is given, this algorithm uses $\mathcal{G}$ as start structure and performs the actions based upon this initial structure. The parameter $\mathcal{G}$ is not given if LEARN_BN_SKELETON_GREEDY is called from S-DAG. In this case LEARN_BN_SKELETON_GREEDY is identical to algorithm 1 (except one small detail:

Algorithm 10: REMOVE_CYCLES

```
   Input: graph $\mathcal{G}$, data set $\mathbf{D}$
   Output: DAG $\mathcal{G}$
1  $\mathbf{E}$ := edges in $\mathcal{G}$ ;
   // remove length two cycles
2  foreach edge $E_{ij} \in \mathbf{E}$ do
3      if $E_{ji} \in \mathbf{E}$ then
4          $\mathcal{G}$ := Remove edge ($E_{ij}$ or $E_{ji}$) with lower score from $\mathcal{G}$;
5      end
6  end
   // remove cycle edges
7  while cycles exist in $\mathcal{G}$ do
8      $E_{ij}$ := detect edge in cycles with smallest score ;
9      remove edge $E_{ij}$ ;
10 end
11 return $\mathcal{G}$
```

The search for better structure is stopped after 5 times instead of 20 times if no structure with improved score is found).

Subnetwork merging (line 3): After the calculation of all Bayesian subnetworks **B**, all learned edges are added to the network structure $\mathcal{G}$ that contains all variables **X**. If an edge already exists in the target structure $\mathcal{G}$, the edge is not added. However, the causing of a cycle in $\mathcal{G}$ is no reason to refuse the adding of the edge. Of course, only the directed edges contained in the subnetworks **B** are added. In particular no undirected edge that might appear the the PDAG representation of the BNs is used to build the structure $\mathcal{G}$.

REMOVE_CYCLES (line 4): Since every subnetwork is learned independently from each other, there might exist cycles in the complete network structure $\mathcal{G}$. This can happen, for instance, if one subnetwork contains an edge from X_i to X_j, and in another subnetwork this edge has exactly the opposite direction, resulting in a cycle in $\mathcal{G}$ containing both variables X_i and X_j. Since the structure of a BN is a directed acyclic graph that must not contain any cycle, we have to transform the potentially cyclic graph into an acyclic graph structure forming the structure of a single Bayesian network. Generally, there are many ways how such a structure can be created. Here, we restrict the DAG structure in such a way that only edges which are in the cyclic graph $\mathcal{G}$ can be considered. Other edges that do not exist in any of the substructures are not considered. Despite this constraint, finding the highest-scoring BN is an $\mathcal{NP}$-hard problem (Hulten, Chickering & Heckerman 2003) which makes heuristic search strategies appropriate. There are two obvious search strategies: Starting from an empty graph, repeatedly add edges from graph $\mathcal{G}$ until no cycle occurs, or to start from the network $\mathcal{G}$ and remove edges until there are no cycles. The first approach is very similar to a normal greedy hill climbing structure learning approach. This means that all edges that are in the final BN must be added, and for all edges, an acyclicity check must be performed. Since we assume that most edges remain in the final BN, and only a few edges cause cycles, we choose the latter approach and employ a greedy search strategy to remove low-scoring edges in $\mathcal{G}$ until there are no cycles. Removing only a few edges leads to the qualified assumption that this approach is more efficient than constructing a complete network from an empty graph. Based on benchmark networks, we will show later on that the number of edges that must be removed to get an acyclic network is much smaller than the number of total edges. This observation justifies our approach to start with the full network and remove edges until there are no cycles left.

The cost of removing an edge is calculated in the following way: Let be $S(\mathcal{G}, X_i \rightarrow X_j)$ the score of the network with edge $X_i \rightarrow X_j$ and $S(\mathcal{G}, X_i \nrightarrow X_j)$ the score without this edge. Then, the cost is defined as $S(\mathcal{G}, X_i \rightarrow X_j) - S(\mathcal{G}, X_i \nrightarrow X_j)$. Since we use the decomposable BDeu score, the cost calculation can be reduced by determining solely the local score $f_{BDeu}(X_j \mid \mathbf{Pa}_j)$ (equation 3.6) of variable X_j with and without the edge. The use of the BDeu score to calculate the cost of removing an edge arises a question: Is it possible to use the BDeu score even if there are cycles in the network? The whole deduction of the BDeu score is based upon the fact that the graph is acyclic. However, we can justify the usage of the BDeu score by the following argument: At the point in time when we remove an edge, we are not interested in the global score of the complete network, indeed we are only interested in the cost of removing a single edge. To calculate this cost, only the variable X_j and its parents are needed. This local structure has no cycles. Thus, for this local structure the score calculation is sound, and since this local score calculation is sufficient to determine the cost, it is possible to calculate the cost of removing an edge in this way.

Algorithm 10 outlines the general procedure of removing the cycles. Instead of applying a simple greedy algorithm that iteratively removes the edge with the lowest cost, we restrict the search on edges that are members of cycles. The simplest cycles are those with a length of two, i.e. cycles that are caused by duplicate edges between two variables X_i and X_j. This means that X_i is a parent of X_j and X_j a parent of X_i, and the variables X_i and X_j

are connected by the two edges $X_i \rightarrow X_j$ and $X_i \leftarrow X_j$. A length-two cycle can be easily broken by removing either edge $X_i \rightarrow X_j$ or $X_i \leftarrow X_j$, in fact we remove the edge with lower cost. Before we remove any other edges in the graph $\mathcal{G}$, we break all length-two cycles to avoid the removal of edges that have a lower cost and are a member of a cycle, but do not cause cycles. After removing all length-two cycles, we perform a standard greedy search and iteratively remove the edge with lowest cost until the graph is acyclic. Thereby, only edges that are members of cycles are considered for the removal. These edges can be determined by finding strongly connected components (SCC) in the graph structure $\mathcal{G}$. Using the algorithm of Tarjan, this operation can be done in $O(n + e)$ (linear time) with n as the number of variables and e the number of edges (Tarjan 1972). As a result, REMOVE_CYCLES returns the acyclic counterpart of the initially cyclic input graph $\mathcal{G}$.

LEARN_BN_SKELETON_GREEDY (line 5): During the process of DAG structure learning with substructures, it can happen that some edges are included in the substructures, but reduce the score of the complete network structure. Others might be added in the wrong direction, i.e an edge in the opposite direction would improve the score in a stronger way. Some edges that would improve the score might be removed during REMOVE_CYCLES because of the greedy search strategy, however they would not cause any cycle in the network structure. Thus, at the end of algorithm S-DAG, we perform a greedy search with LEARN_BN_SKELETON_GREEDY to improve the structure $\mathcal{G}$ returned from REMOVE_CYCLES.

5.3.2 Results

To benchmark our method, we investigate its performance on artificial data with real world characteristics focusing on the well-known Alarm (Beinlich, Suermondt, Chavez & Cooper 1989) and Insurance (Binder et al. 1997) networks. In addition to the results presented before, we also applied the algorithm on the HailFinder (Jensen & Jensen 1996) network. As before, we used the tiling-method (Tsamardinos, Statnikov, Brown & Aliferis 2006) to enlarge the networks in size. We used the Alarm, Alarm_10, Alarm_20, Alarm_30, Alarm_50, Alarm_270, Insurance, Insurance_10, Insurance_20, Insurance_30, Insurance_200, HailFinder and HailFinder_10 as benchmark networks and applied the algorithms on data sets with size 200, 500, 1000, and 5000. We sampled five different data sets for each data set size and averaged the result over the five different data sets. In total, we applied the algorithms to 260 different data sets. For more information about the benchmark networks we refer to chapter A.2. For a short-hand notation, we sometimes refer to the benchmark networks with the following abbreviations: A., A._10, A._20, A._30, A._50, A._270, I., I._10, I._20, I._30, I._200, H. and H._10.

For the BDeu score calculation, we used an equivalent sample size of 10 for all networks. The number of iterations to find a better graph was set to 20 for the MMHC comparison algorithm (see line 13 of algorithm 1), while we used 5 for the S-DAG algorithm (line 14 of algorithm 9). The MMPC and PMMS algorithms were used with the same parameters as described by the original authors. We used the same settings for the MMHC algorithm as in Tsamardinos, Brown & Aliferis (2006), thus the results of the authors of MMHC are directly comparable to our results presented in this chapter. To allow a proper performance comparison of all algorithms, we reimplemented the MMHC, MMPC and PMMS algorithms. Thus we can directly compare the algorithms and can neglect differences that might be caused by different implementations and runtime environments.

In order to measure the quality of the learned network structures and the complexity to learn the structures, the values of several metrics are reported. Apparently, the BDeu score is one of the most important metrics to rate the structure learning algorithm, since this score is optimized during the structure learning process. An algorithm that leads to a better score optimizes the score in a better way. Besides the BDeu score, the Structural Hamming Distance (SHD) and the number of statistical calls (NSC) are used. SHD is a measure that counts the structural differences between two network structures and was already introduced in chapter 4.3.2.

An important measure is also the runtime performance of an algorithm. However, the pure runtime of an algorithm depends on the actual implementation, the programming language and the execution environment an is thus a measure that can heavily depend on factors that are not based on the algorithm itself. To avoid such influences on the measure, we use the number of statistical calls (NSC) as a substitute for the runtime performance. Under the reasonable assumption that the learning algorithm spends most time in calculating statistical values from data, the number of these statistical calculations can be used as an indicator for the runtime performance (Tsamardinos, Brown & Aliferis 2006). In our case, each call of $dep(...)$ or $ind(...)$ in MMPC or PMMS (see algorithm 11 in the appendix) and each call of the BDeu score calculation is counted as one statistical call. In our implementation of the algorithms we use a BDeu score cache and calculate the BDeu score for each parameter setting only once. If a requested BDeu score is found in the cache, we do not count this call for the measure NSC.

In the next section, we compare S-DAG directly to MMHC since MMHC outperforms many other BN network learning algorithms. Later, we also compare our method to algorithms that are not included in the MMHC comparison study.

5.3.2.1 Comparison to MMHC

Based on several measures, we empirically benchmark both S-DAG algorithms by a comparison to MMHC. As stated before, it has been shown that MMHC has a good performance in terms of efficiency and network quality and outperforms greedy hill climbing, optimal reinsertion, sparse candidate, PC, three phase dependency analysis and greedy equivalent search (Tsamardinos, Brown & Aliferis 2006). Thus, an algorithm that outperforms MMHC shows also better results than these algorithms. For a better comparison, we have normalized the values of all measures to the results of MMHC. Thus, a value smaller than 1 denotes a better result than MMHC, while a value greater than 1 implies a result that is worse than MMHC.

At first, we show the results regarding SHD (Structural Hamming Distance), BDeu score, NSC (number of statistical calls) and runtime for each benchmark network and sample size separately. The values presented in the following tables are averaged over the five different data sets with the same size. We also show the average over all sample sizes per method and network in the column *Avg*.

Table 5.5 contains the normalized “BDeu score” results. This score directly shows the optimization capability of the score-based BN reconstruction algorithms since the algorithms try to optimize this score. As lower the value for the normalized score, as better the optimization by the algorithm. If we compare the MMHC algorithm with its S-DAG counterpart S-DAG MMPC that uses the same skeleton reconstruction method as MMHC, it can be clearly seen that our S-DAG MMPC algorithm never performs worse than the MMHC algorithm (all normalized values are 1 or below 1). Indeed, our algorithm outperforms MMHC in almost every setting of our benchmark. The approach to use substructures and afterwards combining them to a complete DAG leads to better optimization results than a pure Greedy-Hill-Climbing approach enriched with a TABU list as used by MMHC. Our second variant of the substructure DAG algorithm called S-DAG PMMS shows a comparable optimization result as S-DAG MMPC, however in rare cases (Alarm network with 5000 samples, table 5.5) it has a worse performance in terms of BDeu score than MMHC. Ob-

Table 5.4: Normalized “SHD” results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	S-DAG PMMS	0.962	1.001	0.960	1.118	1.010
Alarm	S-DAG MMPC	0.949	1.001	0.921	1.000	0.968
Alarm_10	S-DAG PMMS	0.935	0.896	0.831	0.782	0.861
Alarm_10	S-DAG MMPC	0.926	0.883	0.831	0.782	0.856
Alarm_20	S-DAG PMMS	0.961	0.898	0.837	0.772	0.867
Alarm_20	S-DAG MMPC	0.953	0.891	0.836	0.778	0.865
Alarm_30	S-DAG PMMS	0.957	0.900	0.813	0.741	0.853
Alarm_30	S-DAG MMPC	0.947	0.895	0.817	0.744	0.851
Alarm_50	S-DAG PMMS	0.961	0.912	0.852	0.791	0.879
Alarm_50	S-DAG MMPC	0.956	0.904	0.852	0.793	0.876
Alarm_270	S-DAG PMMS	0.993	0.927	0.843	0.789	0.888
Alarm_270	S-DAG MMPC	0.981	0.915	0.845	0.790	0.883
Insurance	S-DAG PMMS	1.000	0.996	1.002	0.879	0.969
Insurance	S-DAG MMPC	1.000	0.972	0.993	0.917	0.971
Insurance_10	S-DAG PMMS	0.964	0.921	0.905	0.804	0.898
Insurance_10	S-DAG MMPC	0.964	0.890	0.907	0.786	0.887
Insurance_20	S-DAG PMMS	0.975	0.919	0.842	0.734	0.867
Insurance_20	S-DAG MMPC	0.974	0.890	0.851	0.720	0.859
Insurance_30	S-DAG PMMS	0.974	0.930	0.868	0.741	0.878
Insurance_30	S-DAG MMPC	0.973	0.894	0.854	0.720	0.860
Insurance_200	S-DAG PMMS	0.995	0.930	0.879	0.759	0.890
Insurance_200	S-DAG MMPC	0.992	0.900	0.870	0.730	0.873
HailFinder	S-DAG PMMS	1.005	0.986	1.036	1.003	1.007
HailFinder	S-DAG MMPC	1.005	1.000	0.988	0.954	0.987
HailFinder_10	S-DAG PMMS	0.998	0.989	0.985	0.966	0.985
HailFinder_10	S-DAG MMPC	0.998	0.989	0.973	0.965	0.981

Table 5.5: Normalized “BDeu score” results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	S-DAG PMMS	0.997	1.000	0.997	1.001	0.999
Alarm	S-DAG MMPC	0.999	1.000	0.997	1.000	0.999
Alarm_10	S-DAG PMMS	0.995	0.994	0.992	0.993	0.994
Alarm_10	S-DAG MMPC	0.996	0.994	0.992	0.993	0.994
Alarm_20	S-DAG PMMS	0.996	0.994	0.992	0.991	0.993
Alarm_20	S-DAG MMPC	0.996	0.994	0.992	0.992	0.994
Alarm_30	S-DAG PMMS	0.995	0.995	0.991	0.990	0.993
Alarm_30	S-DAG MMPC	0.996	0.995	0.992	0.990	0.993
Alarm_50	S-DAG PMMS	0.996	0.994	0.992	0.990	0.993
Alarm_50	S-DAG MMPC	0.996	0.994	0.992	0.990	0.993
Alarm_270	S-DAG PMMS	0.996	0.994	0.991	0.989	0.992
Alarm_270	S-DAG MMPC	0.997	0.995	0.991	0.989	0.993
Insurance	S-DAG PMMS	1.000	0.997	1.000	0.997	0.999
Insurance	S-DAG MMPC	1.000	0.996	1.000	0.997	0.998
Insurance_10	S-DAG PMMS	0.994	0.983	0.985	0.980	0.985
Insurance_10	S-DAG MMPC	0.996	0.984	0.986	0.980	0.987
Insurance_20	S-DAG PMMS	0.995	0.984	0.981	0.972	0.983
Insurance_20	S-DAG MMPC	0.996	0.985	0.984	0.972	0.984
Insurance_30	S-DAG PMMS	0.995	0.986	0.982	0.978	0.985
Insurance_30	S-DAG MMPC	0.997	0.987	0.984	0.977	0.986
Insurance_200	S-DAG PMMS	0.997	0.986	0.982	0.978	0.986
Insurance_200	S-DAG MMPC	0.999	0.988	0.984	0.977	0.987
HailFinder	S-DAG PMMS	1.000	1.000	0.999	0.999	1.000
HailFinder	S-DAG MMPC	1.000	1.000	1.000	0.999	1.000
HailFinder_10	S-DAG PMMS	1.000	0.999	0.999	0.998	0.999
HailFinder_10	S-DAG MMPC	1.000	0.999	0.999	0.999	0.999

Table 5.6: Normalized “NSC” results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	S-DAG PMMS	0.970	0.990	1.061	0.945	0.992
Alarm	S-DAG MMPC	1.050	1.054	1.048	1.044	1.049
Alarm_10	S-DAG PMMS	1.000	0.993	0.989	0.981	0.991
Alarm_10	S-DAG MMPC	1.018	1.014	1.014	1.016	1.016
Alarm_20	S-DAG PMMS	0.999	0.995	0.992	0.990	0.994
Alarm_20	S-DAG MMPC	1.010	1.008	1.008	1.009	1.009
Alarm_30	S-DAG PMMS	0.999	0.994	0.993	0.992	0.995
Alarm_30	S-DAG MMPC	1.008	1.006	1.005	1.006	1.006
Alarm_50	S-DAG PMMS	0.999	0.994	0.993	0.994	0.995
Alarm_50	S-DAG MMPC	1.005	1.003	1.003	1.004	1.004
Alarm_270	S-DAG PMMS	0.995	0.993	0.993	0.995	0.994
Alarm_270	S-DAG MMPC	1.003	1.001	1.001	1.001	1.001
Insurance	S-DAG PMMS	0.970	0.950	0.865	0.724	0.877
Insurance	S-DAG MMPC	1.063	1.063	1.041	1.032	1.050
Insurance_10	S-DAG PMMS	1.003	0.991	0.956	0.799	0.937
Insurance_10	S-DAG MMPC	1.040	1.031	1.029	1.027	1.032
Insurance_20	S-DAG PMMS	1.002	0.996	0.976	0.863	0.959
Insurance_20	S-DAG MMPC	1.029	1.019	1.019	1.021	1.022
Insurance_30	S-DAG PMMS	1.008	0.994	0.979	0.886	0.967
Insurance_30	S-DAG MMPC	1.031	1.013	1.014	1.016	1.019
Insurance_200	S-DAG PMMS	1.010	0.996	0.993	0.975	0.994
Insurance_200	S-DAG MMPC	1.024	1.002	1.002	1.003	1.008
HailFinder	S-DAG PMMS	0.951	0.941	0.833	0.325	0.762
HailFinder	S-DAG MMPC	1.054	1.010	1.014	1.009	1.022
HailFinder_10	S-DAG PMMS	1.000	0.994	0.970	0.790	0.938
HailFinder_10	S-DAG MMPC	1.052	1.015	1.008	1.010	1.021

Table 5.7: Average normalized “SHD” results

	Sample size				
Method	200	500	1000	5000	Avg
S-DAG PMMS	0.975 ± 0.019	0.939 ± 0.021	0.896 ± 0.026	0.837 ± 0.032	0.912
S-DAG MMPC	0.971 ± 0.020	0.925 ± 0.019	0.887 ± 0.028	0.822 ± 0.030	0.901
Empty	1.254 ± 0.038	1.591 ± 0.046	2.137 ± 0.117	2.927 ± 0.191	1.977

Table 5.8: Average normalized “BDeu Score” results

	Sample size				
Method	200	500	1000	5000	Avg
S-DAG PMMS	0.997 ± 0.001	0.993 ± 0.001	0.991 ± 0.001	0.989 ± 0.002	0.992
S-DAG MMPC	0.998 ± 0.001	0.993 ± 0.001	0.992 ± 0.001	0.989 ± 0.001	0.993
True	1.013 ± 0.003	0.995 ± 0.003	0.987 ± 0.003	0.978 ± 0.003	0.993
Empty	1.326 ± 0.007	1.444 ± 0.006	1.508 ± 0.007	1.583 ± 0.007	1.465

Table 5.9: Average normalized “NSC” results

	Sample size				
Method	200	500	1000	5000	Avg
S-DAG PMMS	0.993 ± 0.004	0.986 ± 0.004	0.969 ± 0.006	0.866 ± 0.006	0.953
S-DAG MMPC	1.030 ± 0.004	1.019 ± 0.003	1.016 ± 0.003	1.015 ± 0.003	1.020

viously, this is due to the poorer quality of the reconstructed skeleton when using PMMS instead of MMPC. But on average, S-DAG MMPC has almost the same BDeu score result with a normalized value of 0.992 as S-DAG PMMS with 0.993. This can be seen in table 5.8. In this table, the results are averaged over all networks. Additionally, we present the mean standard deviation over the different data sets per data set size. The low values for the standard deviation (between 0.001 and 0.002) indicate that the scores are significantly smaller for networks reconstructed with S-DAG. This means that S-DAG performs a better network optimization than MMHC. For a better comparison of the score values, the normalized BDeu score for an empty network and the true network are also shown.

While the S-DAG algorithm clearly outperforms the MMHC algorithm in terms of score optimization, the main goal is to retrieve high-quality network reconstructions. The network with the best score is not automatically the network with the best quality. This can be seen by the fact that the averaged BDeu score for the true network is (with a value of 1.013) greater than for the learned networks with 200 samples (see table 5.8). This means that S-DAG finds networks that have better scores than the true network structures. Thus we compare the algorithms also based on the Structural Hamming Distance (SHD) results presented in the tables 5.4 and 5.7. While for smaller networks the S-DAG PMMS method does six times (out of totally 52 different settings) not reach the result of MMHC (e.g. average for Alarm network: S-DAG PMMS has a normalized SHD of 1.010), S-DAG MMPC usually outperforms MMHC. Only for two of the 52 settings, the SHD value for S-DAG is worse than for MMHC. For the rest, S-DAG MMPC is equal to MMHC or better. For some of the benchmarks, the normalized SHD is even below 0.75 (e.g. Insurance_20, 5000 samples). On average, both S-DAG methods outperform MMHC with around 90% to 91% of structural difference to the true network compared to MMHC. This means that there are almost 10% less structural differences in networks learned by S-DAG compared to MMHC.

In tables 5.6 and 5.9, the normalized results for “NSC” are reported. While the S-DAG PMMS method has a better performance than MMHC with an average value of 0.949, the performance of the S-DAG MMPC algorithm is worse than MMHC with an average value of 1.021. Both values show, however, that the performance of all three algorithms is comparable.

5.3.2.2 Analysis of S-DAG

In this section we analyse the properties of our new algorithm S-DAG in more detail and have a closer look at the single steps of this algorithm. One of the crucial steps for the performance of S-DAG is the splitting process: Big subnetworks directly lead to a non-performant run of the S-DAG algorithm. Thus, each subnetwork must be as small as possible to obtain a good performance result. The size of one subnetwork is directly dependent on the number of parents and children in the skeleton. As a result, the crucial step here is to get a skeleton that contains as less false positives as possible.

Based on different sizes of the tiled Alarm networks with various sample sizes, we show the properties of the skeletons produced by PMMS as well as MMPC. In addition to the comparison study in the previous section, we present also the result for smaller data sets with a size of 50 and 100 samples. Table 5.10 shows the maximum number of parents and children in the networks, separately for several sample sizes. Two observations are apparent: First, the maximum number increases with the size of the networks: For 50 samples, the Alarm network has a maximum number of parents and children of 11.6, while the 50 times greater Alarm_50 network has 116.4. Second, the number decreases with growing number of samples: For 50 samples, the Alarm_50 network has a number of 116.4 that decreases to 6.6 for 5000 samples. The same observations hold for the mean number of parents and children (see table 5.11). While for 50 and 100 samples, the number gets too large to learn efficiently with substructures, with 200 or more samples the average number is always smaller than 4. For 500 samples or more it is even smaller than 3. This coincides with the averaged number of different statistical calls for S-DAG MMPC in table 5.9. The normalized number shrinks from 1.030 for 200 samples to 1.015 for 5000 samples. This gain of less statistical calls seems to be small, but later we show that far more than 95% of the statistical calls are needed for the skeleton reconstruction and not for the BN learning. Thus, the reduction of about 1.5% of statistical calls is a big performance improvement for the BN learning procedure.

We also investigate the single steps of the S-DAG algorithm in more detail. For this, table 5.12 shows the characteristics of a single S-DAG PMMS call for the Alarm_50 network. For each of the steps of algorithm 8 (PMMS (line 1), Substructure (line 2), Remove Cycles (line 4) and Greedy (line 5)), we show the number of statistical calls (NSC) with BDeu score cache, the NSC without caching, and the runtime in milliseconds. In the rows labelled with "Total", either the absolute number of statistical calls or the runtime in milliseconds

Table 5.10: Skeleton: Maximum number of parents and children

		Sample size						
Network	Method	50	100	200	500	1000	5000	Avg
Alarm	PMMS	11.6	12.8	9.4	7.2	6.6	5.2	8.8
Alarm	MMPC	11.6	12.8	9.4	7.2	5.0	5.0	8.5
Alarm_10	PMMS	35.0	26.6	10.6	7.0	6.0	6.0	15.2
Alarm_10	MMPC	35.0	26.6	10.6	7.0	6.0	6.0	15.2
Alarm_20	PMMS	60.2	44.4	11.2	7.0	6.0	6.0	22.5
Alarm_20	MMPC	60.2	44.4	11.4	6.8	5.6	6.0	22.4
Alarm_30	PMMS	75.2	53.8	15.6	7.0	5.6	6.4	27.3
Alarm_30	MMPC	75.2	53.8	15.6	7.0	5.4	6.4	27.2
Alarm_50	PMMS	116.4	89.0	19.8	7.4	6.0	6.6	40.9
Alarm_50	MMPC	116.4	89.0	19.8	7.4	6.0	6.6	40.9

Table 5.11: Skeleton: Mean number of parents and children

		Sample size						
Network	Method	50	100	200	500	1000	5000	Avg
Alarm	PMMS	3.5	5.0	3.3	2.5	2.4	2.3	3.1
Alarm	MMPC	3.5	5.0	3.2	2.4	2.2	2.3	3.1
Alarm_10	PMMS	12.5	8.2	3.1	2.5	2.4	2.6	5.2
Alarm_10	MMPC	12.5	8.2	3.1	2.5	2.4	2.6	5.2
Alarm_20	PMMS	21.5	12.1	3.2	2.5	2.4	2.5	7.4
Alarm_20	MMPC	21.5	12.1	3.2	2.5	2.4	2.5	7.4
Alarm_30	PMMS	30.4	15.6	3.5	2.6	2.4	2.5	9.5
Alarm_30	MMPC	30.4	15.6	3.4	2.6	2.4	2.5	9.5
Alarm_50	PMMS	47.9	23.0	3.8	2.6	2.5	2.6	13.7
Alarm_50	MMPC	47.9	23.0	3.7	2.6	2.5	2.6	13.7

Table 5.12: Characteristics of S-DAG PMMS for Alarm_50 network

	Sample size				
	200	500	1000	5000	Avg
NSC with BDeu score cache					
Total	1909436	1912872	1919588	1936746	1919661
PMMS*	0.986	0.991	0.991	0.990	0.989
Substructure*	0.013	0.009	0.008	0.010	0.010
Remove Cycles*	0.000	0.000	0.000	0.000	0.000
Greedy*	0.001	0.000	0.000	0.000	0.000
NSC without cache					
Total	2339790	2163860	2145831	2189520	2209750
PMMS*	0.805	0.876	0.887	0.876	0.861
Substructure*	0.185	0.115	0.104	0.115	0.130
Remove Cycles*	0.002	0.002	0.002	0.002	0.002
Greedy*	0.007	0.006	0.006	0.006	0.006
Runtime in ms					
Total	28770	42945	68640	286892	106812
PMMS*	0.894	0.951	0.968	0.983	0.949
Substructure*	0.094	0.042	0.027	0.014	0.044
Remove Cycles*	0.001	0.001	0.001	0.000	0.001
Greedy*	0.011	0.006	0.004	0.003	0.006
Tarjan statistics					
Length 2 Cycles	299.6	275.8	321.2	348.2	311.2
Removed Edges	20.4	18.6	18.4	22.4	20.0

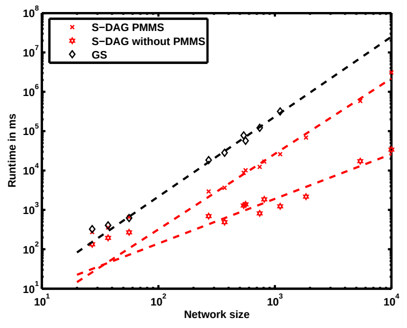
*: Values denote the fraction of the contribution to the values shown in the "Total" rows

is shown. The values for the single steps show the fraction of statistical calls or runtime needed by the step.

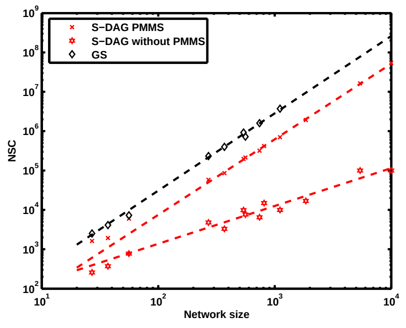
If the BDeu score cache is activated, around 99% of the score calculations are performed by the PMMS skeleton reconstruction method. Only around 1% are needed for the substructure algorithm, while “Remove Cycles” and “Greedy” can be neglected. To show the real number of statistical calls, we also present the numbers without BDeu score cache. Thus, consecutive calls to calculate the BDeu score for the same set of parameters are considered, as well. Without BDeu score cache, the substructure part needs about 13% of the statistical calls, on average. “Remove Cycles” and “Greedy” together do not even reach 1%. The majority of all statistical calls is performed by PMMS with a fraction of 86%. The runtime acts very similar: around 95% of the runtime is needed for PMMS, but only 4% for substructure.

We have shown that the step to remove the cycles is very performant and has only a marginal influence on the total runtime of the algorithm. To see the cause we have a closer look at this step and show the number of cycles. After combining all substructures into a single graph structure, there are on average 311 cycles with length two in the graph. Length two cycles consist of two nodes X_i and X_j that are linked with two edges, one from node X_i to node X_j and one from node X_j to node X_i. So there are 311 edges that have a different direction in at least two substructures. To break all cycles in the graph that exist additionally to the length two cycles, the “Remove Cycles” algorithm has removed 20 edges, on average. This justifies the approach we use to make the network acyclic: Since we start with the full network and remove edges (and do not start with the empty network and add edges), we only have to remove around 20 edges after breaking the cycles of length 2 instead of adding a huge amount of edges (the original Alarm_50 network has 2854 edges). Thus, this approach saves a lot of computational time.

S-DAG is meant to be a scalable algorithm that can be used to learn in large domains. One interesting question is how S-DAG performs if the number of variables increases. To answer this question we show the runtime of the S-DAG PMMS algorithm as function of the size of the network in figure 5.1(a). The results are based on data sets with 1,000 samples, results for other sizes of the data sets are not presented. For all networks in our comparison study, we plot the size of the network on the x-axis and the runtime in milliseconds on the y-axis. The runtime result for each network is the average over the runtime results of the five data sets with 1,000 samples. We show the results separately for the algorithm S-DAG PMMS and S-DAG PMMS without the PMMS part. Additionally,



(a) Runtime



(b) NSC

Figure 5.1: Runtime and NSC depending on the size of the network for all data sets with 1000 samples.

we show the runtime of Greedy Hill Climbing (GS) for comparison. A more detailed analysis of the GS results is presented in the next section. Besides the size of the network, its topology can also have an influence on the runtime. This influence is not considered here.

Under the assumption that S-DAG shows polynomial runtime behaviour if the size of the network is increased, the log-log graph of the runtime as function of the network size should be asymptotically linear. The figure shows that the logarithm of the runtime of the S-DAG PMMS algorithm is approximately linear dependent on the logarithm of the network size. To gain an estimation of the dependency between runtime and network size we applied a linear regression on the logarithmic values for the three cases. Thereby we considered only networks with more than 100 variables in order to avoid a massive influence of small-scale networks and to focus more on the asymptotic behaviour. The gradient of the line for S-DAG PMMS has a value of 1.9. Thus, the runtime is roughly proportional to $network\_size^{1.9}$ with *network_size* as the size of the network.

Before we have shown that the part of learning substructures shows linear complexity if the maximum number of parents and children is fixed. To show the real performance of the substructure part including the part to retrieve the DAG, we added the values for S-DAG without the PMMS part to the figure. The results strengthen the theoretical considerations: The runtime depending on the network size is approximately proportional to $network\_size^{1.1}$ which is almost linear.

The real runtime can always be influenced by the physical hardware and the system environment. For instance, we experienced that for large networks the computer sometimes started to perform swapping operations. This can dramatically influence the runtime in a negative way. Thus we also report the results for NSC. The NSC in dependence on the network size is plotted in figure 5.1(b) as log-log plot. To obtain the results, we performed the same steps as before for the runtime. In this figure we see that the logarithm of NSC is approximately linearly dependent on the logarithm of the network size for S-DAG PMMS and GS. The slopes of the lines have a value of 1.9 and 2.0, respectively. The runtime results that indicate an almost linear dependency for S-DAG are approved by the results for NSC: With 1.0 as the slope of the line the NSC is approximately linearly dependent on the network size.

In general, the results for the dependency between the runtime or NSC on the network size must be treated with care. For networks with different topology, the results could look differently. Also the number of samples have an influence on the behaviour of the algorithms. Last but not least we expect that PMMS has a complexity that is at least quadratic (see algorithm 11): With a dependency in the magnitude of $network\_size^{1.9}$ we assume that the real dependency is a little bit underestimated. On the other hand, at least to a certain extent the results can be used to predict how S-DAG PMMS performs for larger networks. For instance, the network with 54,675 nodes that is presented in chapter 6 was learned with about $1.1 \cdot 10^9$ statistical calls, while the expectation based on our results amounts to $1.3 \cdot 10^9$ statistical calls.

5.3.3 Comparison to Other BN Structure Learning Algorithms

In this section we compare the S-DAG method to other state-of-the-art Bayesian network reconstruction methods. We omit the comparison with optimal reinsertion, sparse candidate, PC, three phase dependency analysis and greedy equivalent search since they are all included in the MMHC performance study (Tsamardinos, Brown & Aliferis 2006). Here, we focus on methods that are not included in this study. All the algorithms we use to benchmark S-DAG were already presented in section 3.3.

The first group of methods including Greedy Hill Climbing, Simulated Annealing and Ant Colony Optimization are included in our software. Thus we are able to establish a good comparison of these methods and to provide detailed results. For the methods that are implemented in our software, we show summary results graphically in figures. The legend for all the figures is shown in figure 5.2. The Structural Hamming Distance is compared in figure 5.3, the BDeu score results are shown in figure 5.4, and the number of statistical calls and runtime are shown in figures 5.5 and 5.6. More detailed results can be found in appendix A.3. For some configurations, some methods failed to learn the networks. In such cases the result is not shown in the figures and marked with n/a in the tables in the appendix.

We included also the results of two other methods, Constraint Hill Climbing (CHC) and Recursive Autonomy Identification (RAI). Both methods are not included in our software, but the authors of the corresponding publications presented direct comparisons to MMHC. So we show the results of the original publications in order to compare these two methods to S-DAG. To clearly distinguish between results we achieved with our software, and results that are taken over from other publications, we present the results of CHC and RAI in separate tables. The Structural Hamming Distance is shown in table 5.13, the BDeu score in table 5.14 and the number of statistical calls in table 5.15. We do not include runtime results since it is useless to compare the runtimes if the implementations of the algorithms and the platforms differ.

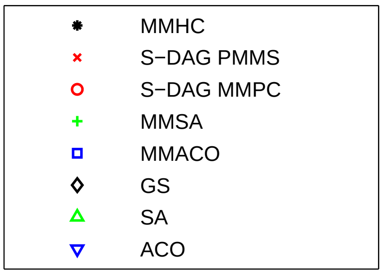


Figure 5.2: Legend for figures 5.3 - 5.6

5.3.3.1 Greedy Hill Climbing: GS

One of the most common algorithms to learn the structure of BNs from data is Greedy Hill Climbing Search. The general algorithm is further on referred to as GS and outlined in algorithm 1. In contrast to S-DAG and MMHC, Greedy Hill Climbing does not imply any restrictions on the network structure and is thus performed without any structural restrictions: There is no skeleton to restrict the search space on. If looking at the averaged BDeu score results of all algorithms (table A.7) one can see that all skeleton-based learning

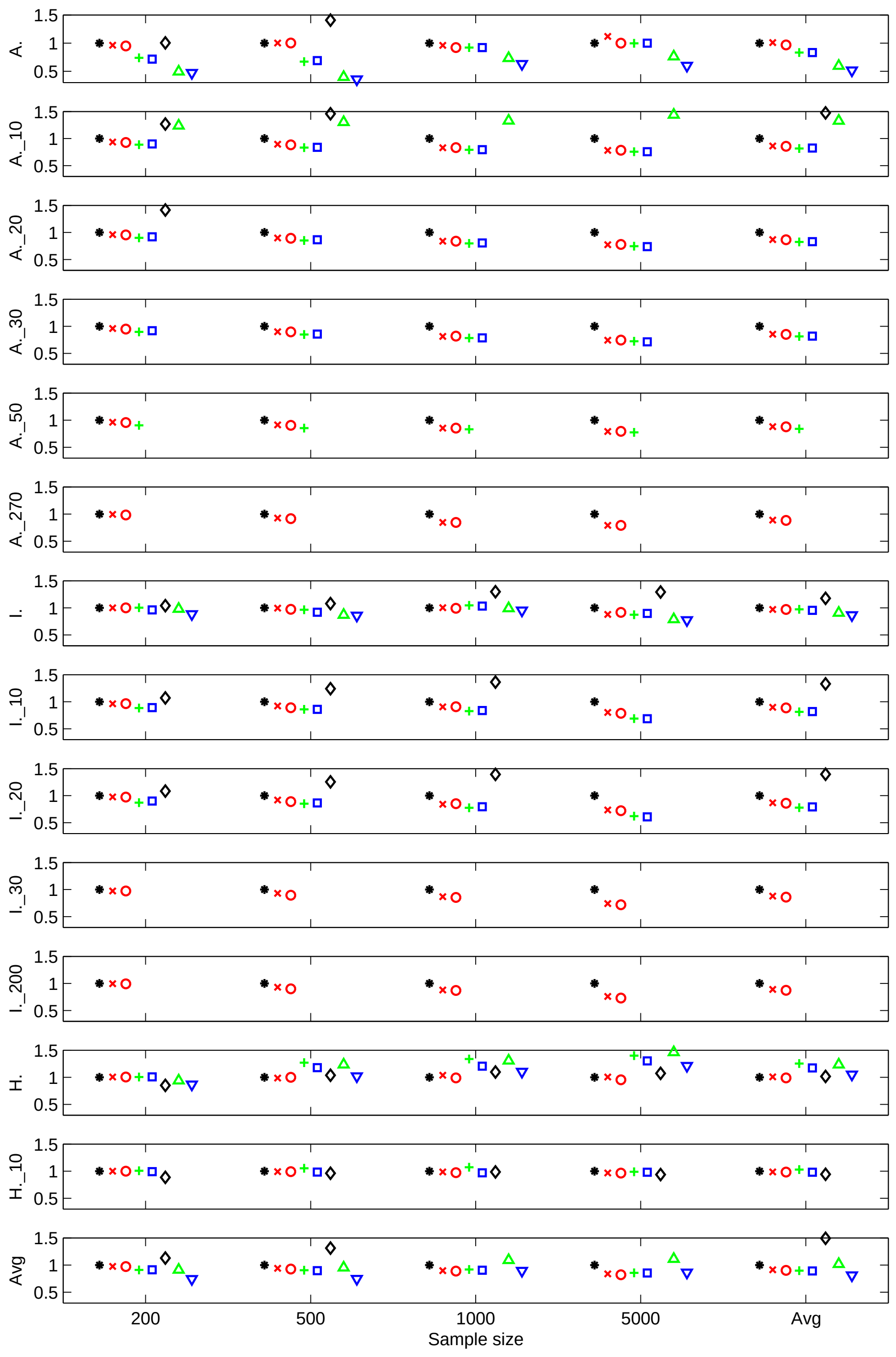


Figure 5.3: Normalized “SHD” results

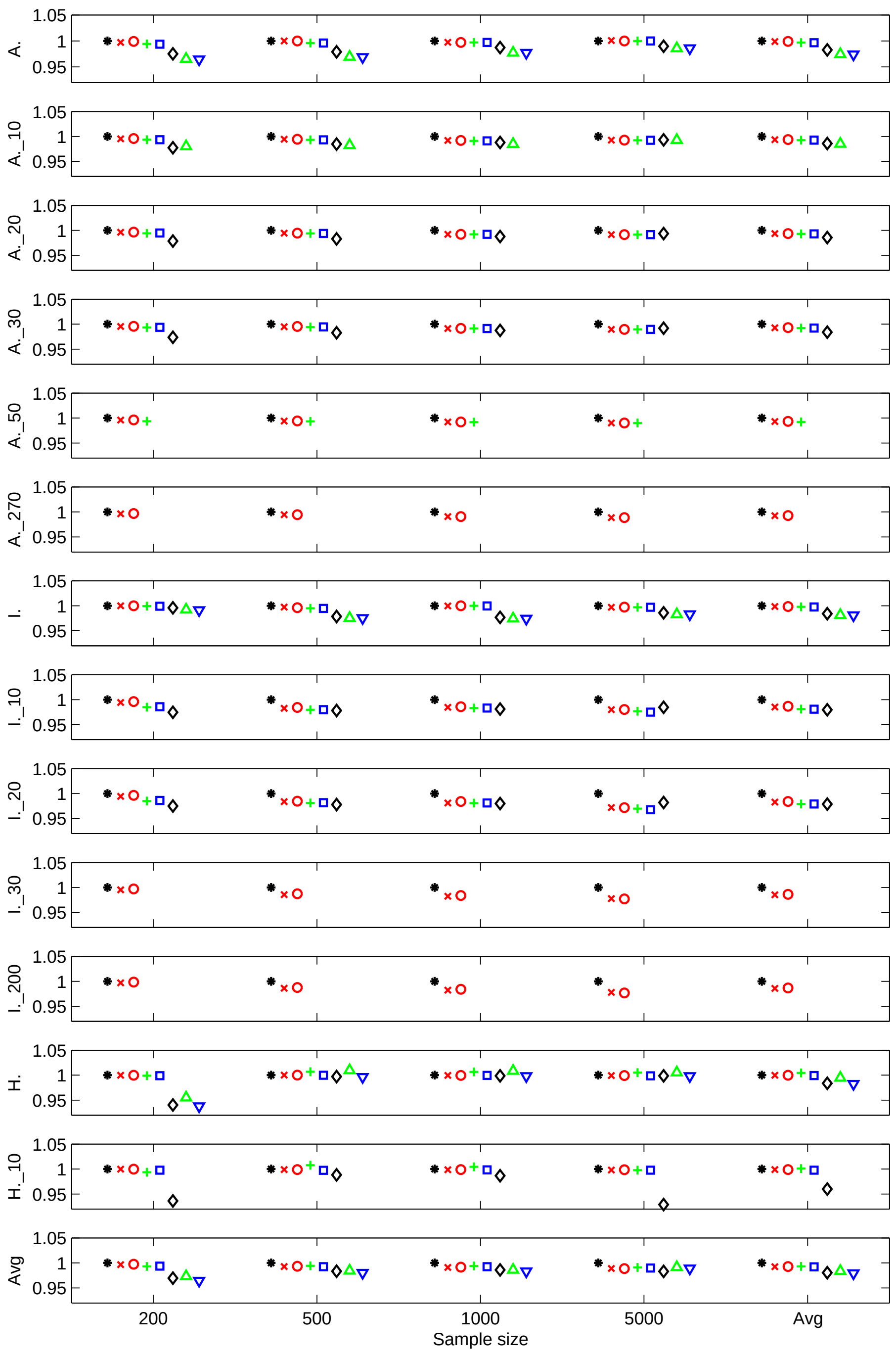


Figure 5.4: Normalized “BDeu score” results

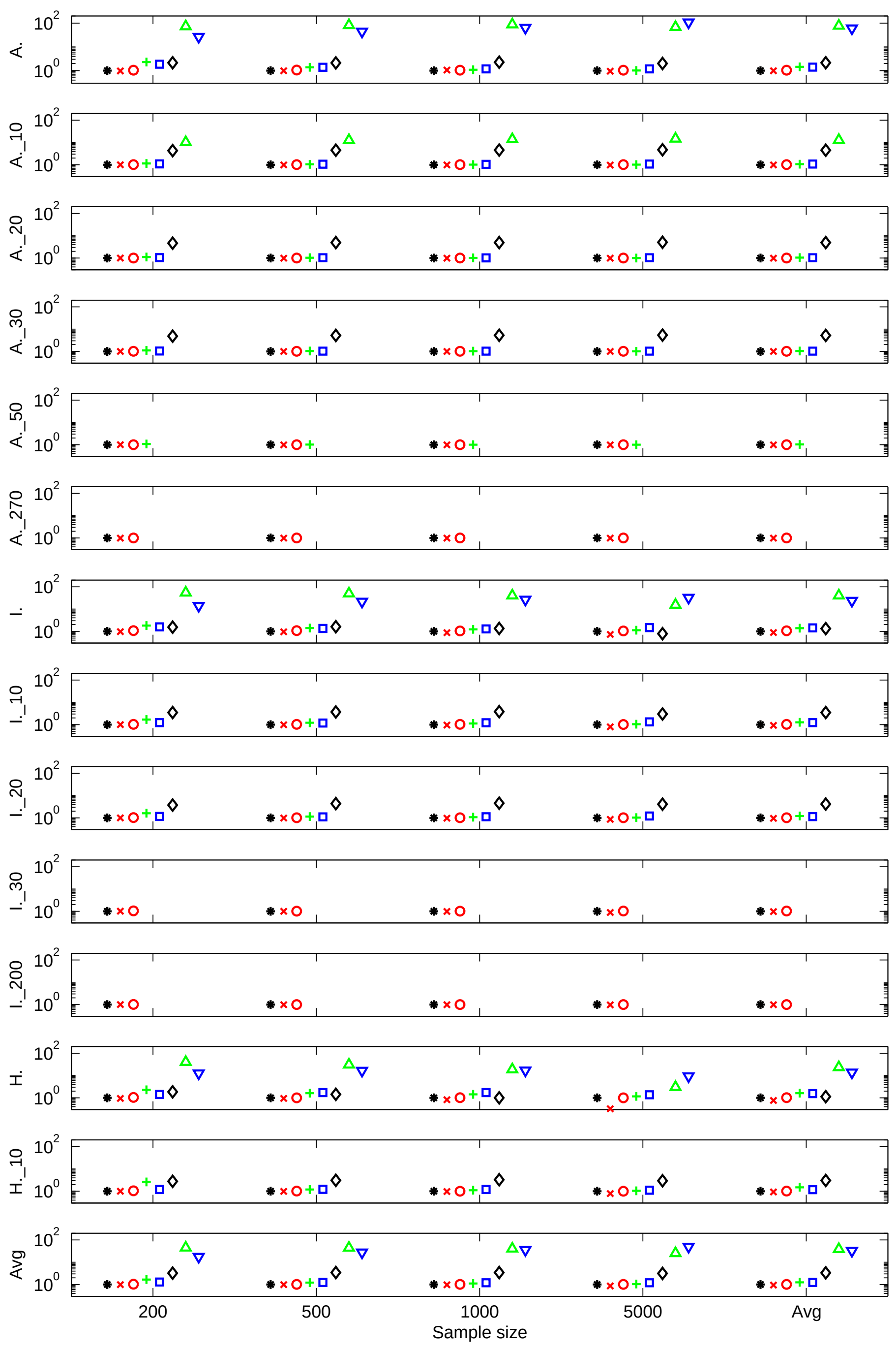


Figure 5.5: Normalized “NSC” results

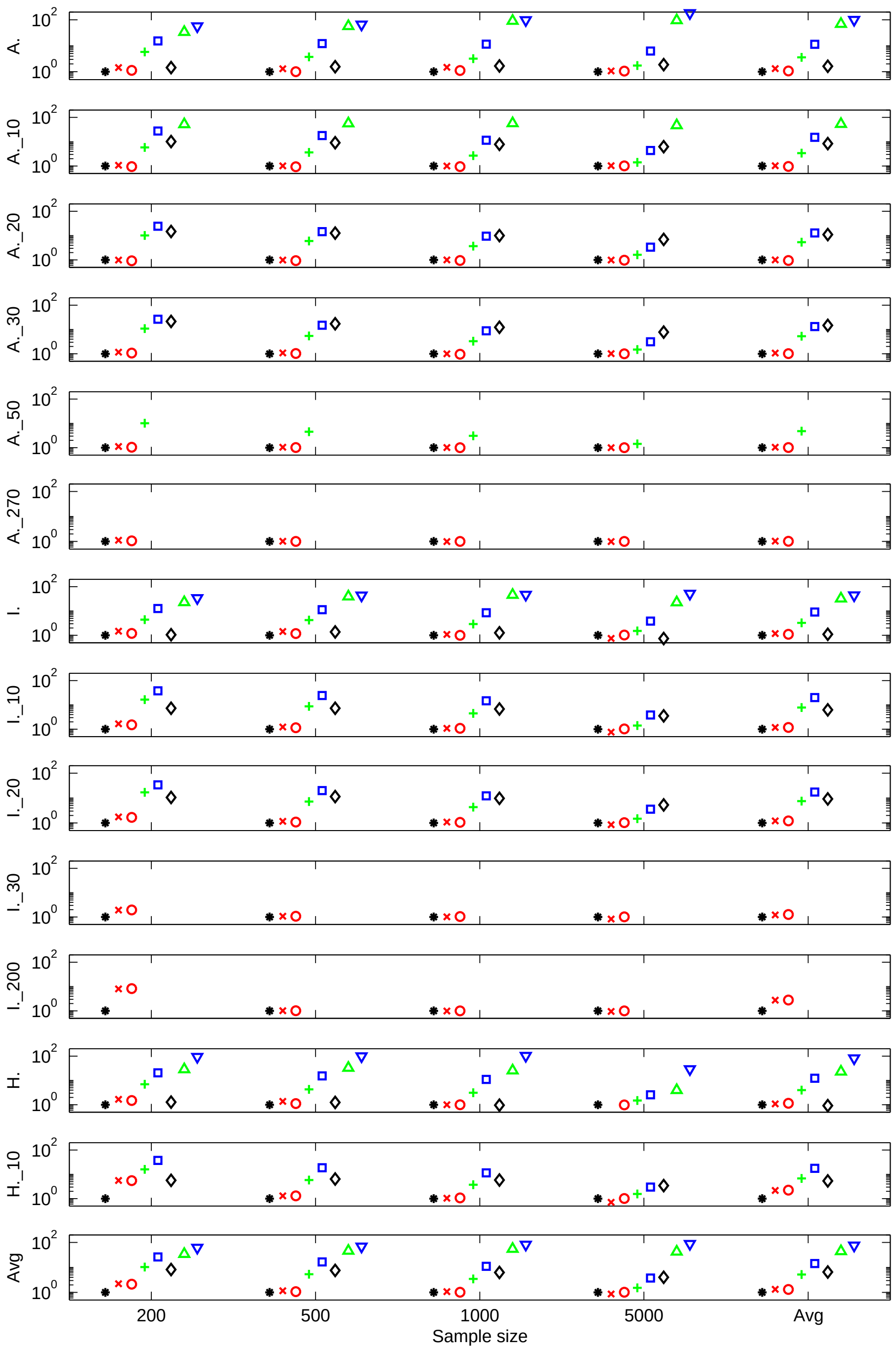


Figure 5.6: Normalized “Runtime” results

algorithms (MMHC, S-DAG, MMSA, MMACO) have worse BDeu score results than algorithms without restriction on a skeleton. This means that, on average, the skeleton limits the search algorithms to structures that can not have as good scores as networks that are not limited to the skeleton. However, achieving better BDeu scores does not always mean that the learned network has a higher quality: The averaged and normalized BDeu score of the true network structure is with 0.993 higher than the averaged BDeu score results of any method that is not based on a skeleton (e.g. GS: 0.981). In fact, the skeleton-based approach prevents overfitting to the data. This means on the other hand that the BDeu score with an equivalent sample size (ESS) of ten can lead to overfitting. In general, the BDeu score is very sensitive on the ESS and it is hard to determine a good value for it (Ueno 2011). Here, we do not focus on optimizing the ESS but use the commonly used value of ten, even if it leads to overfitting.

GS does not restrict the search space on a skeleton. This implies that the number of BDeu score calculations can be higher than for approaches that restrict the search space. In fact this leads to a major increase of number of statistical calls (NSC) compared to S-DAG (see figure 5.5): It is about three times higher than the NSC results for the S-DAG algorithms (see appendix A.3, table A.8). This means that the number of BDeu score calculations is higher than the sum of dependence or independence tests or BDeu score calculations in the S-DAG case. Also the runtime is around five times slower than the runtime of the S-DAG algorithms (see table A.9). Thereby, the average values are based upon relatively small networks: GS failed to learn the networks Alarm_30, Alarm_50, Alarm_270, Insurance_30 and Insurance_200 within a reasonable time. From a network quality point of view, GS produces around 50% more structural errors than S-DAG, on average (see table A.6).

The direct comparison between S-DAG and GS shows that GS finds networks with better BDeu score, but S-DAG outperforms GS in terms of number of statistical calls, runtime and quality of the reconstructed network (SHD), while GS even fails to learn large networks within a reasonable runtime.

5.3.3.2 Simulated Annealing: SA and MMSA

Greedy Hill Climbing often gets stuck in local optima. A metaheuristic that can be used to overcome this limitation is Simulated Annealing (SA) (Kirkpatrick et al. 1983). The algorithm we used as benchmark here is outlined in algorithm 2 and was also used by others (Dejori 2005, Pinto et al. 2009). In our benchmark study, we use two different kinds of Simulated Annealing algorithms: The standard *SA* algorithm and a version that restricts the edge search on a skeleton. This algorithm is called *MMSA* and uses the skeleton produced by MMPC to restrict the search space. MMSA was first published in Pinto et al. (2008).

Since Simulated Annealing is a non-deterministic algorithm, we applied SA and MMSA five times on each data set. The results are based upon the average results of these five runs. Besides, for the sake of completeness we also report the result of the best run. This run is chosen as best run that has the best BDeu score. In the results tables (A.2 - A.9), the result with the best SA run is labelled *SA b*, and the result with the best MMSA run *MMSA b*.

First, we compare S-DAG to MMSA since both algorithms restrict their search space on a skeleton. On average, MMSA reconstructs networks that have a BDeu score close to the networks reconstructed with both S-DAG algorithms (see table A.7). MMSA produces a normalized BDeu score of 0.993 which is close to both S-DAG algorithms (0.992 for S-DAG PMMS and 0.993 for S-DAG MMPC). The quality of the reconstructed networks is slightly better than for S-DAG. With 0.897 as the averaged normalized SHD, MMSA is better than S-DAG PMMS (0.912) and S-DAG MMPC (0.901) (table A.6). The drawback of this method is that the runtime performance of MMSA is much worse than the performance of the S-DAG algorithms: With around 25% more statistical calls, the runtime is about four times higher (tables A.8 and A.9). The Alarm_270, Insurance_30 and Insurance_100 are too large to be learned with MMSA within a reasonable time.

The standard SA algorithm shows similar characteristics as the GS algorithm: The BDeu score is with an average value of 0.984 better than the averaged BDeu score of both S-DAG algorithms (0.992 and 0.993). However, in terms of SHD results, the standard SA algorithm performs worse than S-DAG and MMSA. With 1.017 as averaged normalized SHD, SA has around 10% more structural errors than both S-DAG algorithms. The runtime results show that SA does not scale well: On average it needs around 40 times more statistical calls than both S-DAG algorithms. This leads to an about 30-fold runtime. Only the networks Alarm, Alarm_10, Insurance and HailFinder were learned in reasonable time, for larger networks we did not apply the SA algorithm since the algorithm did not finish within a reasonable time.

In general, MMSA reconstructs the networks with a slightly better quality than S-DAG, but pure SA is worse than S-DAG. The runtime performance of both algorithms show that they do not scale well and even fail for larger networks.

5.3.3.3 Ant Colony Optimization: ACO, MMACO and related algorithms

Another meta-heuristic approach is Ant Colony Optimization which mimics the way how ants seek for food (Dorigo & Stützle 2004, Stützle, López-Ibáñez, Dorigo, Cochran, Cox, Keskinocak, Kharoufeh & Smith 2011). If applied to BN structure learning, the idea is that an ant makes a decision not only based upon the score, but uses so-called pheromones, as well. The pheromones contain information about networks learned in previous iterations. After every iteration, the pheromones are updated with the information about the learned network, typically with the best network (called global update) and the network from the current iteration (called local update). The algorithm used as benchmark for our S-DAG algorithms is called ACO, respectively MMACO for the version that uses a skeleton to restrict the search space. Ant colony optimization algorithms heavily depend on the meta-parameters of the algorithm: Thus, we use the same parameters as proposed by Pinto et al. (2009). For more information we refer to section 3.3.3 or the original publication (Pinto et al. 2009).

Similarly to the Simulated Annealing algorithms, ACO/MMACO are non-deterministic algorithms. Thus we run the algorithm five times and report the average result, as well as the result for the best BDeu score. The result with the best ACO run is labelled *ACO b*, and the result with the best MMACO run *MMACO b*.

On average, MMACO achieves almost identical BDeu score results as S-DAG. With a value of 0.992 for the normalized BDeu score it is in the same range as 0.992 for S-DAG PMMS and 0.993 for S-DAG MMPC (table A.7). The networks learned by MMACO have a normalized Structural Hamming Distance to the original network of 0.891, on average. This is slightly better than the SHD for both S-DAG algorithms (0.901 and 0.912, respectively). Similarly to Simulated Annealing, the number of statistical calls is higher for MMACO: With 1.236 as NSC, both S-DAG algorithms outperform MMACO with 0.953 and 1.020 as NSC. The runtime for MMACO is much worse: It is about 10 times higher than for both S-DAG algorithms. MMACO fails to learn Alarm_50, Alarm_270, Insurance_30 and Insurance_200 within reasonable runtime.

ACO without any restriction on a skeleton achieves better results regarding the quality of the reconstructed networks. The results (figures 5.3 - 5.6 or tables A.2 - A.9) show that ACO achieves the best networks for all algorithms included in this study. For the Alarm network, ACO has only around 50% of the network errors of S-DAG (table A.2). Also the BDeu score is with a relative value of 0,974 better than for any other network induction method (SA has 0,975, for instance) (table A.3). But the number of statistical calls and the runtime show the performance problem of the ACO algorithm: For the Alarm network, the NSC is around 60 times higher than for S-DAG, and the runtime about 100 times (table A.5). The problem becomes manifest if ACO is applied to large networks: ACO fails to learn larger networks than Alarm, Insurance and HailFinder.

There is a series of other Ant Colony Optimization algorithms to learn the structure of Bayesian networks. ACO-B, I-ACO-B, ChainACO and K2ACO are shortly introduced in section 3.3.3: All these algorithms have the same problem with the runtime performance. Since Ant Colony Optimization is a meta-heuristic that additionally has normally a greedy hill climbing step inside its optimization procedure, it is typical that the runtime performance is worse than for a greedy hill climbing algorithm. Since we are interested in a good performance combined with a good quality of the networks, we do not compare S-DAG to further Ant Colony Optimization algorithms, even if they slightly differ in runtime and quality.

As a conclusion, MMACO and ACO produce networks with higher quality than S-DAG, in general, but with the drawback of poor runtime performance. For larger networks, MMACO and ACO even fail to learn the structure within reasonable runtime.

Constraint Hill Climbing: CHC*, iCHC and 2iCHC

The authors of constraint hill climbing provide detailed results about their algorithms (Gámez & Puerta 2005). Since there was no implementation of the algorithms available we are unable to provide a full comparison to our methods. Instead, we just transform the provided results into their normalized form. It is obvious that their values can not be directly compared to our values since the data sets are different. But the values normalized to MMHC just show the difference between the method to rate and MMHC. Thus it is likely that the normalized values can be compared at least to a certain degree, even if the data sets are not identical, but the network remains the same. Results for their methods and MMHC are provided in Gámez et al. (2011), so we normalized the results to MMHC and compare these values to our normalized results (see tables 5.13 - 5.15, the values shown here for CHC*, iCHC and 2iCHC are based upon the values presented in Gámez et al. (2011)). Only the results for the largest networks in Gámez et al. (2011) are shown.

The goal of constraint hill climbing is to provide a fast and efficient structure learning algorithm. This can be seen if the number of statistical calls are compared to S-DAG (see

Table 5.13: Normalized “SHD” results for Constraint Hill Climbing and RAI

		Sample size			
Network	Method	500	1000	5000	Avg
Alarm_10	CHC*	1.243	1.376	1.415	1.345
Alarm_10	iCHC	1.240	1.343	1.415	1.333
Alarm_10	2iCHC	1.243	1.333	1.391	1.322
Alarm_10	RAI	n/a	n/a	n/a	0.87
Alarm_10	*S-DAG PMMS*	0.896	0.831	0.782	0.836
Alarm_10	*S-DAG MMPC*	0.883	0.831	0.782	0.832
Insurance_10	CHC*	1.044	1.065	1.172	1.094
Insurance_10	iCHC	1.163	1.134	1.129	1.142
Insurance_10	2iCHC	1.149	1.148	1.265	1.187
Insurance_10	RAI	n/a	n/a	n/a	0.88
Insurance_10	*S-DAG PMMS*	0.921	0.905	0.804	0.877
Insurance_10	*S-DAG MMPC*	0.890	0.907	0.786	0.861
HailFinder_10	CHC*	0.957	1.028	0.913	0.966
HailFinder_10	iCHC	1.011	1.099	1.058	1.056
HailFinder_10	2iCHC	1.011	1.099	1.054	1.055
HailFinder_10	RAI	n/a	n/a	n/a	0.74
HailFinder_10	*S-DAG PMMS*	0.989	0.985	0.966	0.980
HailFinder_10	*S-DAG MMPC*	0.989	0.973	0.965	0.976

Table 5.14: Normalized "BDeu score" results for Constraint Hill Climbing and RAI

		Sample size			
Network	Method	500	1000	5000	Avg
Alarm_10	CHC*	0.992	0.998	0.995	0.995
Alarm_10	iCHC	0.994	0.999	0.996	0.996
Alarm_10	2iCHC	0.995	0.999	0.998	0.997
Alarm_10	RAI	n/a	n/a	n/a	n/a
Alarm_10	*S-DAG PMMS*	0.994	0.992	0.993	0.993
Alarm_10	*S-DAG MMPC*	0.994	0.992	0.993	0.993
Insurance_10	CHC*	0.986	0.986	0.987	0.986
Insurance_10	iCHC	0.995	0.995	0.989	0.993
Insurance_10	2iCHC	0.995	0.997	1.003	0.998
Insurance_10	RAI	n/a	n/a	n/a	n/a
Insurance_10	*S-DAG PMMS*	0.983	0.985	0.980	0.983
Insurance_10	*S-DAG MMPC*	0.984	0.986	0.980	0.983
HailFinder_10	CHC*	0.975	0.972	0.997	0.981
HailFinder_10	iCHC	0.986	0.978	0.999	0.988
HailFinder_10	2iCHC	0.986	0.978	0.999	0.988
HailFinder_10	RAI	n/a	n/a	n/a	n/a
HailFinder_10	*S-DAG PMMS*	0.999	0.999	0.998	0.999
HailFinder_10	*S-DAG MMPC*	0.999	0.999	0.999	0.999

Table 5.15: Normalized “NSC” results for Constraint Hill Climbing and RAI

		Sample size			
Network	Method	500	1000	5000	
Alarm_10	CHC*	1.476	1.444	1.348	1.423
Alarm_10	iCHC	0.954	0.937	0.899	0.930
Alarm_10	2iCHC	0.882	0.846	0.818	0.849
Alarm_10	RAI	n/a	n/a	n/a	0.75
Alarm_10	*S-DAG PMMS*	0.993	0.989	0.981	0.988
Alarm_10	*S-DAG MMPC*	1.014	1.014	1.016	1.015
Insurance_10	CHC*	1.392	1.328	1.192	1.304
Insurance_10	iCHC	0.875	0.827	0.708	0.803
Insurance_10	2iCHC	0.817	0.766	0.586	0.723
Insurance_10	RAI	n/a	n/a	n/a	0.63
Insurance_10	*S-DAG PMMS*	0.991	0.956	0.799	0.915
Insurance_10	*S-DAG MMPC*	1.031	1.029	1.027	1.029
HailFinder_10	CHC*	1.284	1.323	1.536	1.381
HailFinder_10	iCHC	0.749	0.775	0.830	0.785
HailFinder_10	2iCHC	0.744	0.764	0.779	0.762
HailFinder_10	RAI	n/a	n/a	n/a	0.77
HailFinder_10	*S-DAG PMMS*	0.994	0.970	0.790	0.918
HailFinder_10	*S-DAG MMPC*	1.015	1.008	1.010	1.038

table 5.15): 2iCHC, which is the most competitive constraint hill climbing algorithm, needs always less statistical calls than both S-DAG algorithms. But the reduction of statistical calls is attended with an increase of structural errors. Results for SHD (see table 5.13) show that S-DAG usually leads to much better network reconstructions than all constraint hill climbing methods. Only CHC*, which is rated by the authors as a non-competitive algorithm in terms of performance, has sometimes better results than S-DAG.

Recursive Autonomy Identification: RAI

Recursive Autonomy Identification (RAI) (Yehezkel & Lerner 2009) employs purely constraint based techniques and uses independence tests to obtain the Bayesian network structure. No scoring function is used here, so the method differs from all methods used in this thesis so far. But for the sake of completeness, we compare our method also to RAI as one of the most competitive constraint based algorithms.

Yehezkel & Lerner (2009) compared RAI to MMHC, thus we are able to compare our method directly to RAI. We added the values for SHD and NSC presented in Yehezkel & Lerner (2009) to the tables 5.13 and 5.15. BDeu score results are not available. The authors of RAI also presented only average results for all sample sizes (500, 1000 and 5000). We just added the largest networks RAI was applied to: Alarm_10, Insurance_10, HailFinder_10.

SHD results show (see table 5.13) that both S-DAG methods reconstruct the networks Alarm_10 and Insurance_10 with better quality. However, RAI outperforms both S-DAG algorithms in terms of SHD for the HailFinder_10 network. The number of statistical calls is about 25% lower for RAI than for S-DAG.

While it seems that RAI partially outperforms S-DAG, the comparison is based upon little data, indeed: The original authors of RAI did not publish results for larger networks, so the comparison is just made with three networks. HailFinder_10 is the largest one with 560 variables. For the other methods we used benchmark networks with up to 9990 variables.

A second point is also quite interesting if comparing score-based methods with constraint-based methods: The score based algorithms depend heavily on the hyperparameters of the BDeu score. We have chosen an equivalent sample size (hyperparameter) of ten to enable a direct comparison to other score-based methods, since this value is widely used in the literature. But other values for the hyperparameter might be better and would lead to better network reconstructions. However, the optimization of the hyperparameter is out of the scope of this thesis. On the other hand, the value for the significance level heavily influences the network quality for constraint-based approaches, as well as the number of free parameters for the significance test. Both values are subject to be optimized.

Because of the differences between score-based and constraint-based approaches, one has to do further experiments in order to do a fair comparison between S-DAG and RAI: RAI must be applied to much larger networks in order to test its scalability, and the hyperparameters for the BDeu score should be optimized. However, we leave this open for a future investigation.

5.3.4 Discussion

We have introduced the S-DAG algorithm as an efficient and scalable Bayesian network structure learning algorithm, which is based on the substructure algorithm presented in section 5.2 (Nägele et al. 2007). While the substructure algorithm returns a set of independent Bayesian subnetworks, S-DAG combines all the subnetworks and learns a single Bayesian network.

An algorithm that also reduces the global learning problem by learning a local network was introduced in Peña, Björkegren & Tegnér (2005). They particularly address the problem of scalability and propose an algorithm that starts from a seed variable that has to be manually selected, and learn the BN structure around this seed variable in an iterative way. However, this algorithm is intended to learn small local models instead of the complete network. An algorithm that deals with domains up to hundreds of thousands of variables was introduced in Goldenberg & Moore (2004). However, it restricts the BN on binary variables paired with very sparsely linked graphs.

Another work that is somehow related to our approach is presented in Hulten et al. (2003). They learn BNs from a dependency network. In dependency networks the joint distribution is defined by a set of conditional probabilities, in this case by means of a decision tree. Unlike BNs where the conditional probability of a variable is defined given its parents, the conditional probability for each variable is defined independently, which allows cycles in the global network structure. Hulten et al. (2003) presented an approach that is very similar to our method: Based on a dependency network that is learned from data, they remove edges until the global graph contains no cycles. However, on benchmark networks they report results that are slightly worse than a simple BN greedy forward selection algorithm.

Another approach to learn high-quality networks in large domains is presented in Herscovici & Brock (2007). They use a model-based search in order to identify regions likely to contain high-quality networks and to restrict the search to these regions. On a ten-fold Alarm network this algorithm produces a network that outperforms MMHC by around 18% by means of SHD. This is consistent with our results (17% decrease of SHD for the Alarm_10 network with 1000 samples). For other networks they report better results: A network with 801 variables was learned with around 50% less structural differences to the original network compared to MMHC. Since we do not have the data for this network, it is not possible to compare this approach directly to our approach. The largest network the model-based search was applied is the network with 801 nodes. This network is much smaller than our Alarm_270 network with 9990 nodes.

An algorithm published in Zeng & Hernandez (2008) also decomposes the complete network into smaller subnetworks. However, the network is just divided into smaller clusters on the basis of a dependency network. Each cluster is learned independently from each other with constraint-based techniques and is combined afterwards on a graphical basis to create a complete BN. Results from benchmark cases show that this algorithm is faster than comparable constraint-based algorithms. The authors emphasize the good performance of their algorithm, however the largest benchmark network contains only 223 nodes.

A quite interesting approach is presented in de Campos & Ji (2011): Based upon properties of scoring functions, they avoid the score computation for edge configurations that can not have a higher score than the currently found solution. However, instead learning efficiently in large domains, their goal is to find the best scoring network within a domain of reasonable size. They managed to find the optimal solution for a domain with around 70 variables.

5.4 Summary

The problem of learning the best scoring Bayesian network from data is $\mathcal{NP}$-hard. In this chapter, we have introduced the substructure algorithm that efficiently estimates the features of the underlying network structure by independently learning small subnetworks. Based on a skeleton that is learned by MMPC, the substructure algorithm takes each variable of the network and learns a local BN around the variable. As learning algorithm, a very fast Random Hill Climbing algorithm was used. We have shown that the network learning phase scales even linearly with the number of variables, if the number of parents and children of each variable is restricted to a constant value. Results from benchmark cases show that structural features of large networks can be learned with high accuracy, comparable to the results of MMHC that uses a greedy search to orient the edges of the skeleton. However, substructure learning scales better for large domains, if the network is only sparsely linked. We have also shown that the framework of dependency networks can be utilized to perform inference on subnetworks. While learning structural features with high quality, the substructure algorithm lacks of a uniform representation of the global structure by means of a BN.

Motivated by the high quality of the reconstructed networks, we introduced the S-DAG algorithm that learns a single BN based on the subnetworks learned by the substructure algorithm. Following the goal to learn networks with high quality, we abandon the substructure's high scalability and use a TABU-enriched greedy search for each substructure. Each subnetwork is afterwards combined to a global network structure, and cycles are removed until a DAG structure is reached. Based on a large study with several large benchmark networks and reasonable large data set sizes, we have shown that our S-DAG algorithm produces networks with competitive quality by comparing it to several state-of-the-art structure learning algorithms. Just RAI which is a constraint-based method to learn the structure of a Bayesian network produces networks with a quality comparable to S-DAG, and this even with less statistical calls. However, this comparison is based upon relatively small networks. Since we did not optimize the hyperparameters of the BDeu score, it is thus hard to compare this constraint-based method to our score-based approach. A closer comparison is left open for future research.

If compared to all other score-based approached included in our study (GS, MMHC, ACO, MMACO, SA, MMSA and the CHC algorithms), there is no algorithm that is faster and better than S-DAG. Either the methods produce networks with better quality (see ACO), but fail to learn in large domains, or S-DAG outperforms the methods in terms of structural quality of the reconstructed networks.

6 Estimating Genetic Networks

6.1 Introduction

In recent years, the reconstruction of the genetic networks with graphical models, in particular Bayesian networks, from microarray data (mRNA expression levels) has shown promising results (Friedman et al. 2000, Pe'er, Regev, Elidan & Friedman 2001b, Hartemink, Gifford, Jaakkola & Young 2002, Peña et al. 2005, Bernard & Hartemink 2005). Bayesian networks can represent both, the quantitative distribution as well as the structural dependencies between the variables. Regarding genetic network reconstruction with BNs, the genes (more precisely: the measured transcripts) are represented as variables (nodes) in the network, while edges between the variables describe relationships between genes.

However, learning BN models from microarray data is problematic due to several aspects. First, BN structure learning requires a sufficiently large amount of independent observations (here: expression profiles) to learn statistically reliable relationships. However, publically available data sets contain usually less than 1000 profiles. This is much less than the number of measured genes which is usuallay up to tens of thousands. Additionally, a BN represents an abstract gene-gene dependency network which might not reflect the true underlying molecular interaction network, but only the statistically based relationships between the measured genes.

Furthermore, current learning methods usually do not scale to large domains with tens of thousands of variables. One typically restricts the feature dimensions to a feasible subset of relevant variables (genes) that are of high interest (Friedman et al. 2000) to make the structure learning process feasible. As a result, a typical pipeline for learning BNs from high-dimensional data could be stated as a two-step process (Stetter et al. 2007): (1) Based on a statistical method or on some other reasonable methods, choose a number of highly-relevant variables, and (2) learn a Bayesian network with the set of variables selected in step 1. As discussed in more detail in section 4.3, the restriction on a small set of variables for learning is a potentially problematic step which can lead to a strongly corrupted estimation of the true structure since (a) edges incident to missing variables cannot be learned by definition, and (b) additional false positive edges might be learned to explain statistical dependencies that can not be represented without the missing variables. Peña et al. (2005) avoid this problem by starting structure learning from a predefined seed gene and iteratively discover parents and children of variables that are already added to the learned network structure. While none of the genes measured in the data set is excluded in advance, the authors show results with only one or two iterations, resulting in a local genetic network estimation around a single seed gene with only tens of genes, but not of the complete dependency structure between all genes.

In the previous section (see section 5.3), we have shown that the S-DAG algorithm scales polynomially, thus being able to learn networks with thousands of variables, enabling a full-genome analysis of microarray data with BNs. We apply the S-DAG PMMS algorithm to a *S. cerevisiae* and a *homo sapiens* microarray data set. To the best of our knowledge, the resulting network from the latter data set with 54,675 transcripts is the largest unrestricted Bayesian network that was learned so far. Based on both data sets, we show the biological relevance of the learned network structures.

6.2 Biological Data

To illustrate the capability of our method to learn from large-scale mRNA data, we apply it to two microarray data sets, the Rosetta compendium (Hughes, Marton, Jones, Roberts, Stoughton, Armour, Bennett, Coffey, Dai, He, Kidd, King, Meyer, Slade, Lum, Stepaniants, Shoemaker, Gachotte, Chakraburtty, Simon, Bard & Friend 2000) and the data set of the expression project for oncology (expO) (igc 2004). In the following we will refer to these data as *Rosetta compendium* for the *S. cerevisiae* and as *oncology data set* for the *homo sapiens* data set.

Table 6.1: Network characteristics

	Rosetta compendium	*Oncology data set*
Samples	300	1,911
Variables (transcripts)	6,146	54,675
Edges	8,599	96,799
Genes	6,109	20,055
Genes with Ontology	5,446	15,697
Genes with Location	6,109	20,053
Genes with Pathway	1,190	3,917

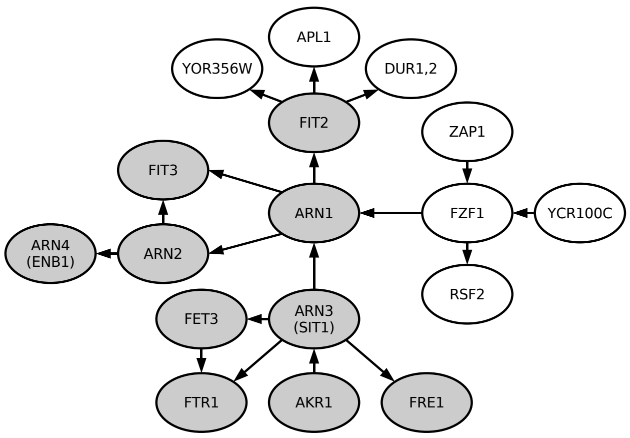
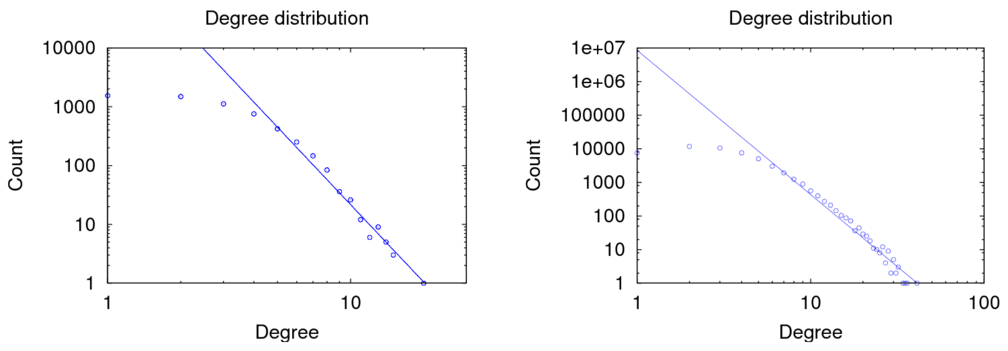
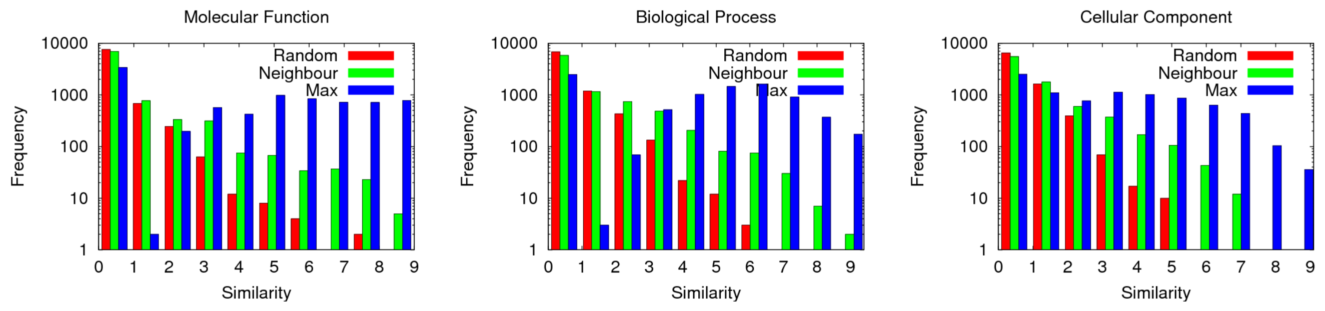


Figure 6.1: BN model of the iron homeostasis pathway learned from the Rosetta compendium. The figure shows the network centered around gene ARN1 with a radius of two, all other genes are not shown. The grey colored nodes represent genes that are supposed to be related to iron homeostasis, while white colored genes are not known to play such a role.

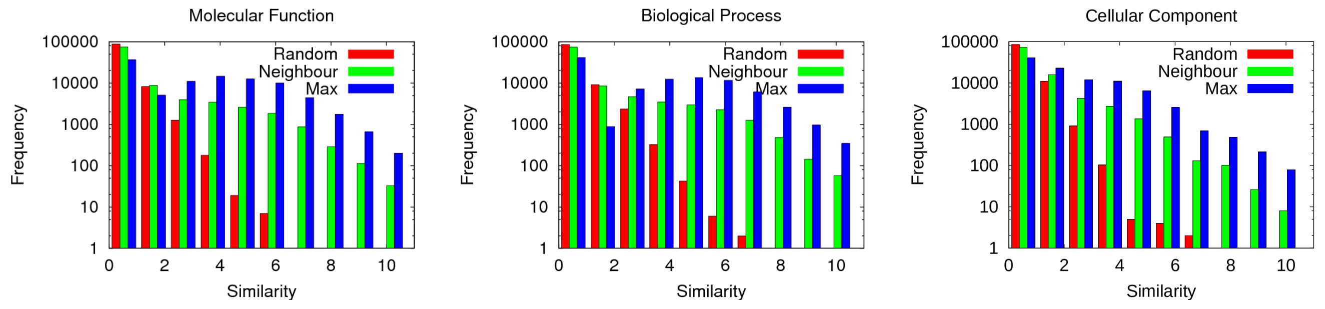


(a) *Rosetta compendium* (b) *Oncology data set*

Figure 6.2: Scale-free characteristics of learned network structures. In a scale-free network, the plot of the degree k against the occurrence frequency forms a straight line when using a log-log scale. The slope of this line has a value of -4.371 for the (a) *S. cerevisiae* (*Rosetta compendium*) network and a value of -4.2904 for the (b) Oncology network. Both graphs show a exponential cut-off for small degree values (degree < 4).



(a) *Rosetta compendium*



(b) *Oncology data set*

Figure 6.3: Semantic similarity distribution. Separately for the Gene Ontology subtrees "molecular function", "biological process" and "cellular component", the figures show the semantic similarity distribution for the (a) *Rosetta compendium* and (b) Oncology networks. Each histogram shows the similarity distribution for genes that are connected in the learned network (denoted by "Neighbour"), genes that are chosen by random ("Random") and genes that are compared to itself as the possible maximum value for similarity ("Max").

6.2.1 Rosetta Compendium

The Rosetta compendium consists of 300 expression profiles of diverse mutations and chemical treatments in *Saccharomyces cerevisiae*, containing the observation of 6,316 measured transcripts (see table 6.1). To obtain a high-quality data set for the network learning task, we removed all transcripts which have missing values in more than 30 profiles, resulting in a data set with 6,146 transcripts. Remaining missing values are replaced according to Troyanskaya, Cantor, Sherlock, Brown, Hastie, Tibshirani, Botstein & Altman (2001). Following the procedure of Stetter et al. (2007), the gene expression levels of this preselected data set (6,146 transcripts $\times$ 300 samples) were normalized to a sample-wise and gene-wise mean value of zero and unity standard deviation and discretized to three levels (over-expressed, unchanged and under-expressed), by using the gene-wise negative and positive standard deviation of the normalized expression levels as thresholds for under-expressed and over-expressed.

To benchmark the validity of network reconstruction models from microarray data, others (Pe'er et al. 2001b, Margolin, Banerjee, Nemenman & Califano 2004, Peña et al. 2005) have used the iron homeostasis pathway in yeast, which regulates the uptake, storage and utilization of iron. Thereby, they learned a network using the Rosetta compendium and extracted the genes centered around ARN1 with a radius of two. ARN1 plays an important role in this pathway together with the genes FRE1, FRE2, FTR1, FET3, ARN2, ARN3, ARN4, FIT1, FIT2 and FIT3 (Protchenko, Ferea, Rashford, Tiedeman, Brown, Botstein & Philpott 2001, Philpott, Protchenko, Kim, Boretsky & Shakoury-Elizeh 2002). In figure 6.1 we show our learned ARN1 subnetwork, and all genes associated with iron homeostasis are marked in grey. Besides the mentioned genes, we additionally marked AKR1, because it is discussed to play a role in iron homeostasis (de Freitas, Kim, Poynton, Su, Wintz, Fox, Holman, Loguinov, Keles, Van Der Laan & Vulpe 2003). All other genes are not known to be related to iron homeostasis. With a radius of two around ARN1, we have 17 genes in the network, 10 of them related to iron homeostasis. This result is consistent with other work, e.g. Peña et al. (2005) obtained also 10 related of 17 total genes with their BN induction method.

Note that while others have learned Bayesian networks only for a subset of genes, we provide a full-genome BN estimation of the regulatory network. This enables an investigation of features and properties of the large-scale network instead of focusing on small-scale characteristics such as the reconstruction of the iron homeostasis pathway.

It is known that genes that show a similar expression behaviour may be functionally related (Zhu, Gerstein & Snyder 2007) or closely positioned on the genome (Darvasi 2003). Thus, genes that are connected in our network should have significantly more common characteristics than expected by chance. To investigate this, we utilize each gene's biological annotation, i.e. the gene ontology (GO) terms (Ashburner, Ball, Blake, Botstein, Butler, Cherry, Davis, Dolinski, Dwight, Eppig, Harris, Hill, Issel-Tarver, Kasarskis, Lewis, Matese, Richardson, Ringwald, Rubin & Sherlock 2000), KEGG pathways (Kanehisa & Goto 2000) and chromosomal location. The annotation was downloaded from the NCBI gene database in December, 2007 and integrated into the GeneSim knowledge platform (Stetter et al. 2007). At first, we use the hierarchical organized gene ontology (GO) to extract functional similarities between genes that are connected in our network. Each GO term describes a biological property and belongs to one of three categories: molecular functions, biological processes and cellular components. Even though two terms are different, they can be closely related by common ancestors. Thus, to measure the annotation similarity between two genes, we do not only count exactly matching GO terms but utilize the hierarchical organization to estimate the similarity between closely related, but different terms. Here, we use the measure of Resnik to calculate the similarity of two terms in an ontology (Resnik 1999): The similarity $RS(c_i, c_j)$ of two terms c_i, c_j is defined as the information content of the closest common parent of both terms, where the information content is the negative logarithmic value of the occurrence frequency. Since each gene can be annotated with several GO terms, we introduce a new score and calculate the mean over all annotation terms to measure the similarity $S(g_i, g_j)$ between two genes g_i, g_j:

$$\begin{aligned} S(g_i, g_j) &= \frac{1}{2} \sum_{c_i \in \mathcal{A}(g_i)} \max_{c_j \in \mathcal{A}(g_j)} RS(c_i, c_j) \\ &+ \frac{1}{2} \sum_{c_j \in \mathcal{A}(g_j)} \max_{c_i \in \mathcal{A}(g_i)} RS(c_j, c_i), \end{aligned} \tag{6.1}$$

where $\mathcal{A}(g_i)$ describes all GO terms associated with gene g_i.

Figure 6.3(a) shows the distribution of the similarity values between genes that are neighbours in the learned BN structure. We report the similarity distribution separately for each subtree of the Gene Ontology (molecular function, biological process and cellular component). We compare the results to the similarity distribution of a random graph and the maximum possible values (similarity of each gene with itself). The similarity val-

Table 6.2: Edge characteristics

		R. compendium	*Oncology*
Same Pathway	Learned Network	351	4,271
	Random Network	17±4	161±14
Same Location	Learned Network	1,220	30,770
	Random Network	654±22	3,471±59

Number of edges in the learned *S. cerevisiae* (*Rosetta compendium*) and Oncology networks that exist between genes that belong to the same pathway or are located on the same chromosome. Additionally, we report the number of expected edges and its standard deviation by sampling random networks.

ues between connected genes are significantly higher than between two randomly selected genes-pairs, which indicates that two genes that are neighbours in a BN show significantly more similar biological characteristics than two genes selected randomly.

We next report the number of gene-pairs that are known to belong to the same pathway (see table 6.2). Totally, 351 of 8,599 edges represent dependencies between genes that belong to the same pathway. To show that these are significantly more edges than expected by chance, we created 100 random networks with 8,599 edges. On average, only 17 (standard deviation: 4) of these edges are present between genes participating in the same pathway, showing the statistical significance of our result.

It has been shown that genes that are closely located on the chromosome can show similar expression characteristics. It is supposed that this can be caused e.g. by a sequence variation that affects several genes in the chromosomal region where this variation occurs (Darvasi 2003). Thus, we finally report also the number of edges between pairs of genes that are located on the same chromosome: 1,220 of these edges exist in the learned network, while in a random network occur only 654±22 of such edges.

A couple of years ago, researchers (Barabási & Oltvai 2004, Basso, Margolin, Stolovitzky, Klein, Dalla-Favera & Califano 2005, Scholz, Dejori, Stetter & Greiner 2005) have shown that cellular networks show scale-free characteristics (Barabási & Bonabeau 2003). Lee & Lee (2005) show the scale-free architecture of the genetic network, estimated from the Rosetta compendium with the “modularized network learning” (MONET) algorithm, while Dejori et al. (2004) obtained a scale-free network when BN learning was applied on a leukemia data set, however on a restricted set of only 271 genes. In contrast, we use the full genome for the genetic network estimation with Bayesian networks. In figure 6.2(a), the log-log plot shows that the probability $p(k)$ of finding a node with degree k follows a power law: $p(k) \sim k^{-\gamma}$ with $\gamma = 4.3741$. The plot shows a exponential cut-off for nodes that have a small degree ($k < 4$), which is consistent with the results presented in other work (Dejori et al. 2004).

6.2.2 Oncology Data Set

As a second biological data set, we use the data from the expression project for oncology (expO) (igc 2004), containing clinically annotated tumor expression profiles. The data were downloaded from the NCBI homepage (ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/SeriesMatrix/GSE2109/) in November, 2007. Data are normalized and discretized to three levels according to the description of the Rosetta compendium. Totally, the data set consists of 54,675 transcripts representing 20,055 different genes with 1,911 samples. On this data set, we trained a BN model using our S-DAG algorithm, resulting in a genetic network estimation of the complete human genome. To the best of our knowledge, this is the largest unrestricted BN that was learned from data so far.

As reported for the Rosetta compendium, the similarity values between connected genes in the network is significantly higher than between two random genes (see figure 6.3(b)). The shift is even stronger than for the *S. cerevisiae* network, and the distribution is close to the “Maximum” similarity distribution. This means that two genes that are connected in the Bayesian network model significantly show a higher biological similarity than two random genes. Edges between genes that belong to the same pathway are overrepresented as well: 4,271 of 96,799 edges belong to that class, while by chance only 161 of such edges occur. The number of co-located genes is also significantly higher: 30,770 of such edges occur, but only 3,471 by chance. The complete network with its 96,799 edges between the transcripts has approximately the properties of a scale-free network with $\gamma = 4.2904$, but has an exponential cut-off for nodes with a small degree ($k <= 4$).

6.3 Summary

We applied our S-DAG method on the expression profiles of a *S. cerevisiae* (Hughes et al. 2000) and a human oncology data set (igc 2004), enabling a full-genome analysis of the learned genetic network estimation. Results from both data sets show that the learned networks have high biological relevance: First, as supposed by previous work (Barabási & Oltvai 2004, Scholz et al. 2005, Basso et al. 2005), the networks have approximately a scale-free architecture. Second, genes that are connected in the graph have significantly more common biological characteristics than random genes. To the best of our knowledge, the oncology network with 54,675 transcripts is the largest unrestricted BN that was learned from data so far.

7 Conclusions

During the last two decades, probabilistic graphical models have become quite popular tools for reasoning under uncertainty. Due to their intuitive graphical representation together with a sound theoretical basis, they are widely accepted tools for both modeling knowledge and making predictions in different domains. Bayesian networks, also known as belief networks, are a prominent representative of the class of probabilistic graphical models. They represent the dependency among random variables by means of a directed acyclic graph, and link each node with a local conditional probability distribution. Both together form a definition of the joint probability distribution of all random variables.

Bayesian networks are applied to provide solutions in many different domains. The applications range from a pure modeling of existing knowledge in so-called expert systems, where the network is build by human experts, over learning the conditional probabilities for a given network structure to learning both, the parameters and the network structure itself. They can be used as pure density estimators to make predictions about random variables. This allows to perform “what-if” scenarios: Given the state of a set of variables, predictions can be made about variables that aren’t observed. But not only the probability distribution, but also the learned network structure can be used as a basis for a topological analysis of the network. To mention only a few: Learned edges in the network give information about direct dependencies among random variables, the direction of edges can in certain cases be suggestive for the causality of the dependency, or the degree of a variable indicates its importance for the network structure.

When drawing such conclusions from a learned BN, it is of major importance to reconstruct the network structure with high accuracy. To improve the quality of learned networks, we introduced several methods in this thesis.

One of the challenging applications for structure learning of Bayesian networks is the estimation of the cellular molecular interaction network, especially the so-called genetic network. The web of mutual biochemical interactions between DNA, RNA, and proteins forms the basis for the genetic network. The genetic network can be modeled as a BN by representing genes as variables and the complex interactions between the genes, their RNA and proteins by means of edges in the BN. High-throughput measurements based on microarrays allow the estimation of the activity of thousands of genes in parallel. If a series of measurements is combined into one microarray data set, the dependencies between the genes can be learned by BN structure learning, deliver an insight into the complex molecular interactions that take place in cells.

Contributions of this work

If learning in such real-world scenarios, one faces problems that do typically not arise if learning from toy data. Microarray data contain with thousands or tens of thousands of measured transcripts much more variables than usual benchmark networks. Due to the $\mathcal{NP}$-completeness of BN structure learning, the size of such a large network poses an enormous challenge. A common approach to put a handle on the problem is the downscaling of the network size: Based on a selection criterion, a feasible set of variables is selected, and the network among these variables is learned from data. Another problem appears with the sparseness of data: Typical microarray data sets usually contain many fewer samples than observed transcripts. With subnetwork learning which is based upon Nägele (2005) and an additional novel approach, we investigated with various measures the robustness of BN structure learning under such real-world circumstances (chapter 4). We have first shown that the downscaling of a data set based on a statistical method increases the number of false positives just because of fluctuations in the data. There are edges that do not represent any true dependency in the data. The separation between these edges and edges that are based upon true dependencies can be done by means of a threshold for the edge confidence if data bootstrap is applied. By using the new measure predictive value which was introduced in chapter 4, one can control the fraction of edges just learned because of fluctuations in the data and those edges based upon true dependencies. Second, we have shown by means of subnetwork learning that the downscaling of the network size has a measurable, deteriorating influence on the learned network structure since variables that are important for the network reconstruction are removed from the data set. To avoid such influences, we introduced dimensional bootstrap as a method to enrich the initially selected subset of variables with those that are of importance to reconstruct the network among the selected variables. This approach assigns a confidence to each edge enabling a separation between false positive and true positive edges. Results have shown that dimensional bootstrap leads to a better reconstruction of the network structure compared to the reconstruction only based upon the downscaled data set.

While the estimation of small subnetworks can be efficiently done with dimensional bootstrap, the main goal is to learn a complete network that contains all observed variables. Many algorithms have been developed to learn Bayesian networks, and some of them address large domains. However, there is need for algorithms that learn unrestricted BNs in large domains very efficiently with high quality. In this thesis, we introduced two novel algorithms (substructure and S-DAG) to learn in such large-scale domains (chapter 5). Both algorithms have the same basic idea: Learning the network is split into many small subproblems. Based on an undirected skeleton that can be estimated by constraint-based methods, small sets of variables are created. Following the idea of dimensionality reduction, for each small set a single Bayesian network, called substructure, is learned. Thereby, edges that can be learned are restricted to those edges that are contained in the skeleton. The first algorithm, called substructure learning, focuses on the graphical representation of large networks. Instead of creating a single BN, substructure learning represents the graph structure as a feature partially directed acyclic graph (fPDAG). Under the condition that the size of the subnetworks is restricted to a maximum value, the phase of learning the Bayesian networks performs even linearly with the number of variables. We have shown that substructure learning offers a better performance than MMHC as being one of the state-of-the-art algorithms, and even outperforms MMHC in terms of structural quality of the estimated network.

Learning the graph structure by substructure learning gives insights about the network structure, however the resulting fPDAG lacks of a representation of the probability distribution by means of a Bayesian network. To solve this problem, we introduced S-DAG as a novel algorithm that utilizes the small subnetworks learned by substructure learning, combines all edges that are learned in these subnetworks in a single network and iteratively removes edges in the complete network until a directed acyclic graph structure remains (chapter 5). We have shown that S-DAG outperforms MMHC in terms of several quality measures based on a large set of benchmark networks, while the performance of S-DAG is comparable to MMHC. Based on several benchmark data we have shown that S-DAG also outperforms many other state-of-the-art score-based Bayesian network structure learning algorithms either in runtime or in quality of the network reconstruction: Greedy Hill Climbing (GS), MMHC, Ant Colony Optimization (ACO, MMACO), Simulated Annealing (SA, MMSA) and Cyclic Hill Climbing (CHC).

Since benchmarks on toy examples have shown promising results, we applied S-DAG on two real microarray data sets to enable a full-genome analysis with Bayesian networks for the first time (chapter 6). The goal of this task was two-fold: First, some important network properties that exist in smaller subnetworks still exist in the large network. The architecture of the network approximately remains scale-free as genetic networks are supposed to be. We have also shown that genes that are a member of the iron homeostasis pathway in yeast, which regulates the uptake, storage and utilization of iron, are closely related in the learned BN. Moreover, we have shown that S-DAG is capable to learn unrestricted Bayesian networks with tens of thousands of variables without any problem. One of the microarray data sets contains almost 55.000 measured transcripts, resulting in a BN with this number of variables. This network, learned with our novel algorithm S-DAG, is one of the largest published BNs that was learned from data so far.

A Appendix

A.1 Algorithm MMPC (Max-Min Parents and Children)

Algorithm 11: MMPC Algorithm

```
Input: target variable $X_i$, data set $\mathbf{D}$, set of variables $\mathbf{X}$
Output: $\mathbf{PC}_i$: the parents and children of $X_i$
// First phase: add positives to $\mathbf{PC}_i$
1  $\mathbf{PC}_i = \emptyset$;
2  repeat
       // add the best candidate to $\mathbf{PC}_i$
3      foreach $X_j \in (\mathbf{X} \setminus \{\mathbf{PC}_i \cup \{X_i\}\})$ do
           // Minimize dependency $dep(X_i, X_j \mid \mathbf{Z})$
4          $Sep[X_j] = \operatorname{argmin}_{\mathbf{Z} \subseteq \mathbf{PC}_i} dep(X_i, X_j \mid \mathbf{Z})$;
5      end
6      $Y = \operatorname{argmax}_{X_j \in (\mathbf{X} \setminus \{\mathbf{PC}_i \cup \{X_i\}\})} dep(X_i, X_j \mid Sep[X_j])$;
       // $X_i$ and $Y$ cond. independent?
7      if not $ind(X_i, Y \mid Sep[Y])$ then
8          $\mathbf{PC}_i = \mathbf{PC}_i \cup \{Y\}$;
9      end
10 until $\mathbf{PC}_i$ has not changed ;
// Second phase: remove false positives from $\mathbf{PC}_i$
11 foreach $X_j \in \mathbf{PC}_i$ do
12     if $ind(X_i, X_j \mid \mathbf{Z})$ for some $\mathbf{Z} \subseteq \mathbf{PC}_i \setminus \{X_j\}$ then
13         $\mathbf{PC}_i = \mathbf{PC}_i \setminus \{X_j\}$ ;
14     end
15 end
16 return $\mathbf{PC}_i$
```

MMPC is an algorithm to detect the parents and children $\mathbf{PC}_i$ of a variable X_i. This algorithm was introduced in Tsamardinos et al. (2003) and is sketched in algorithm 11.

The algorithm is taken from Pinto et al. (2009).

The detection of parents and children is done in two phases: A candidate set of parents and children is created in the first phase (growing phase). In the second phase all false positives are removed. The dependency *dep* is calculated by using the negative p-value of the G^2 independence test (Spirtes et al. 2001, Tsamardinos, Brown & Aliferis 2006). Variables are rendered conditionally independent (function *ind*) if they are conditionally independent on a significance level of 0.05.

A.2 Benchmark Networks

Throughout this thesis, a set of different benchmark networks is used. The benchmark networks vary in network topology, size, domain range, number of edges, maximum in- and out-degree, besides others. These properties are summarized in table A.1. Since there is the demand for large benchmark networks, commonly used and well-known Bayesian networks (Alarm, Insurance and Hailfinder) are used and are increased in size by the tiling method described in Tsamardinos, Statnikov, Brown & Aliferis (2006), which uses one network as tile and puts several tiles together. The number in the suffix of the network name denotes the number of tiles used for the network.

The **Alarm** network was constructed by human experts for monitoring patients in intensive care and introduced in Beinlich et al. (1989). It has become a popular benchmark network for BN structure learning algorithm.

Insurance is a Bayesian network to model car insurance risk and to estimate the expected claim costs for a policyholder (Binder et al. 1997).

Hailfinder is a Bayesian network, constructred by meteorological data and modeled with expert knowledge to forecast severe weather in Northeastern Colorado (Abramson, Brown, Edwards, Murphy & Winkler 1996).

The **ALL_benchmark_1000** was learned from the ALL microarray data set (Yeoh, Ross, Shurtleff, Williams, Patel, Mahfouz, Behm, Raimondi, Relling, Patel & Cheng 2002). The network consists of 1000 discrete variables, and each of them has three states.

Table A.1: Properties of Bayesian networks: number of variables (Variables), number of edges (Edges), maximum in/out degree of a node (Max In/Out), maximum number of parents and children (Max PC), average number of parents and children (Avg PC) and domain range (DR)

Network	Variables	Edges	Max In/Out	Max PC	Avg PC	DR
Alarm	37	46	4 / 5	6.0	2.49	2 – 4
Alarm_10	370	570	4 / 7	9.0	3.08	2 – 4
Alarm_20	740	1101	4 / 7	7.0	2.98	2 – 4
Alarm_30	1110	1580	4 / 7	8.0	2.85	2 – 4
Alarm_50	1850	2854	4 / 8	9.0	3.09	2 – 4
Alarm_270	9990	15559	4 / 8	9.0	3.11	1 – 4
Insurance	27	52	3 / 7	9.0	3.85	2 – 5
Insurance_10	270	556	5 / 8	11.0	4.12	2 – 5
Insurance_20	540	1074	3 / 8	10.0	3.98	2 – 5
Insurance_30	810	1619	3 / 8	10.0	4.00	2 – 5
Insurance_200	5400	10774	3 / 10	12.0	3.99	1 – 5
HailFinder	56	66	4 / 16	17.0	2.36	2 – 11
HailFinder_10	560	1017	5 / 20	21.0	3.63	2 – 11
ALL_benchmark_1000	1000	1157	3 / 29	31.0	2.31	2 – 3

A.3 Detailed Results of Structure Learning Algorithms

In this section, detailed results about the performance of BN learning algorithms are reported. All the results are normalized to the results of MMHC. Some algorithms need so much runtime that they did not always finish within reasonable time (several hours). In this case, the result is not available and marked with n/a.

Table A.2: Normalized "SHD" results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm	S-DAG PMMS	0.962	1.001	0.960	1.118	1.010
Alarm	S-DAG MMPC	0.949	1.001	0.921	1.000	0.968
Alarm	MMSA	0.738	0.672	0.921	0.995	0.831
Alarm	MMSA b	0.721	0.699	0.921	1.000	0.835
Alarm	MMACO	0.715	0.690	0.921	1.000	0.831
Alarm	MMACO b	0.707	0.665	0.921	1.000	0.823
Alarm	GS	1.005	1.408	3.338	5.074	2.706
Alarm	SA	0.493	0.398	0.734	0.761	0.596
Alarm	SA b	0.493	0.343	0.898	0.715	0.612
Alarm	ACO	0.472	0.356	0.629	0.597	0.514
Alarm	ACO b	0.461	0.356	0.629	0.597	0.511
Alarm	True	0.000	0.000	0.000	0.000	0.000
Alarm	Empty	1.580	2.263	6.242	11.143	5.307
Alarm_10	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_10	S-DAG PMMS	0.935	0.896	0.831	0.782	0.861
Alarm_10	S-DAG MMPC	0.926	0.883	0.831	0.782	0.856
Alarm_10	MMSA	0.885	0.831	0.790	0.756	0.816
Alarm_10	MMSA b	0.887	0.835	0.796	0.751	0.817
Alarm_10	MMACO	0.901	0.838	0.795	0.757	0.823
Alarm_10	MMACO b	0.897	0.842	0.802	0.760	0.825
Alarm_10	GS	1.269	1.459	1.546	1.628	1.475
Alarm_10	SA	1.240	1.303	1.331	1.438	1.328
Alarm_10	SA b	1.269	1.292	1.365	1.413	1.335
Alarm_10	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_10	True	0.000	0.000	0.000	0.000	0.000
Alarm_10	Empty	1.312	1.630	1.854	2.390	1.797
Alarm_20	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_20	S-DAG PMMS	0.961	0.898	0.837	0.772	0.867

Table A.2: Normalized "SHD" results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm_20	S-DAG MMPC	0.953	0.891	0.836	0.778	0.865
Alarm_20	MMSA	0.901	0.851	0.798	0.746	0.824
Alarm_20	MMSA b	0.904	0.857	0.802	0.750	0.828
Alarm_20	MMACO	0.919	0.864	0.804	0.737	0.831
Alarm_20	MMACO b	0.920	0.864	0.804	0.740	0.832
Alarm_20	GS	1.414	1.584	1.663	1.753	1.604
Alarm_20	SA	n/a	n/a	n/a	n/a	n/a
Alarm_20	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_20	True	0.000	0.000	0.000	0.000	0.000
Alarm_20	Empty	1.325	1.627	1.850	2.319	1.780
Alarm_30	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_30	S-DAG PMMS	0.957	0.900	0.813	0.741	0.853
Alarm_30	S-DAG MMPC	0.947	0.895	0.817	0.744	0.851
Alarm_30	MMSA	0.897	0.848	0.782	0.724	0.813
Alarm_30	MMSA b	0.895	0.847	0.784	0.722	0.812
Alarm_30	MMACO	0.918	0.855	0.784	0.713	0.817
Alarm_30	MMACO b	0.917	0.857	0.782	0.711	0.817
Alarm_30	GS	1.528	1.779	1.928	1.935	1.792
Alarm_30	SA	n/a	n/a	n/a	n/a	n/a
Alarm_30	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_30	True	0.000	0.000	0.000	0.000	0.000
Alarm_30	Empty	1.324	1.677	1.935	2.380	1.829
Alarm_50	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_50	S-DAG PMMS	0.961	0.912	0.852	0.791	0.879
Alarm_50	S-DAG MMPC	0.956	0.904	0.852	0.793	0.876
Alarm_50	MMSA	0.905	0.852	0.830	0.774	0.840
Alarm_50	MMSA b	0.905	0.852	0.830	0.773	0.840
Alarm_50	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	GS	n/a	n/a	n/a	n/a	n/a
Alarm_50	SA	n/a	n/a	n/a	n/a	n/a
Alarm_50	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	True	0.000	0.000	0.000	0.000	0.000
Alarm_50	Empty	1.242	1.495	1.741	2.108	1.647
Alarm_270	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_270	S-DAG PMMS	0.993	0.927	0.843	0.789	0.888
Alarm_270	S-DAG MMPC	0.981	0.915	0.845	0.790	0.883

Table A.2: Normalized "SHD" results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm_270	MMSA	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	GS	n/a	n/a	n/a	n/a	n/a
Alarm_270	SA	n/a	n/a	n/a	n/a	n/a
Alarm_270	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	True	0.000	0.000	0.000	0.000	0.000
Alarm_270	Empty	1.104	1.410	1.667	1.978	1.540
Insurance	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance	S-DAG PMMS	1.000	0.996	1.002	0.879	0.969
Insurance	S-DAG MMPC	1.000	0.972	0.993	0.917	0.971
Insurance	MMSA	1.004	0.964	1.047	0.874	0.972
Insurance	MMSA b	1.005	0.948	1.048	0.883	0.971
Insurance	MMACO	0.961	0.920	1.034	0.897	0.953
Insurance	MMACO b	0.961	0.920	1.034	0.917	0.958
Insurance	GS	1.040	1.078	1.299	1.292	1.177
Insurance	SA	0.980	0.871	0.991	0.789	0.908
Insurance	SA b	1.000	0.829	0.927	0.804	0.890
Insurance	ACO	0.883	0.858	0.951	0.772	0.866
Insurance	ACO b	0.883	0.840	0.944	0.774	0.860
Insurance	True	0.000	0.000	0.000	0.000	0.000
Insurance	Empty	1.241	1.514	1.894	2.246	1.724
Insurance_10	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_10	S-DAG PMMS	0.964	0.921	0.905	0.804	0.898
Insurance_10	S-DAG MMPC	0.964	0.890	0.907	0.786	0.887
Insurance_10	MMSA	0.885	0.859	0.826	0.688	0.814
Insurance_10	MMSA b	0.894	0.860	0.828	0.677	0.815
Insurance_10	MMACO	0.893	0.860	0.839	0.686	0.819
Insurance_10	MMACO b	0.891	0.864	0.834	0.691	0.820
Insurance_10	GS	1.070	1.242	1.364	1.649	1.331
Insurance_10	SA	n/a	n/a	n/a	n/a	n/a
Insurance_10	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_10	True	0.000	0.000	0.000	0.000	0.000
Insurance_10	Empty	1.359	1.592	1.902	2.433	1.822
Insurance_20	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_20	S-DAG PMMS	0.975	0.919	0.842	0.734	0.867
Insurance_20	S-DAG MMPC	0.974	0.890	0.851	0.720	0.859

Table A.2: Normalized “SHD” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance_20	MMSA	0.870	0.851	0.774	0.622	0.779
Insurance_20	MMSA b	0.866	0.854	0.771	0.623	0.779
Insurance_20	MMACO	0.900	0.866	0.793	0.608	0.792
Insurance_20	MMACO b	0.903	0.869	0.790	0.602	0.791
Insurance_20	GS	1.085	1.255	1.392	1.851	1.396
Insurance_20	SA	n/a	n/a	n/a	n/a	n/a
Insurance_20	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_20	True	0.000	0.000	0.000	0.000	0.000
Insurance_20	Empty	1.378	1.688	1.954	2.555	1.894
Insurance_30	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_30	S-DAG PMMS	0.974	0.930	0.868	0.741	0.878
Insurance_30	S-DAG MMPC	0.973	0.894	0.854	0.720	0.860
Insurance_30	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	GS	n/a	n/a	n/a	n/a	n/a
Insurance_30	SA	n/a	n/a	n/a	n/a	n/a
Insurance_30	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	True	0.000	0.000	0.000	0.000	0.000
Insurance_30	Empty	1.331	1.672	1.975	2.623	1.900
Insurance_200	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_200	S-DAG PMMS	0.995	0.930	0.879	0.759	0.890
Insurance_200	S-DAG MMPC	0.992	0.900	0.870	0.730	0.873
Insurance_200	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	GS	n/a	n/a	n/a	n/a	n/a
Insurance_200	SA	n/a	n/a	n/a	n/a	n/a
Insurance_200	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	True	0.000	0.000	0.000	0.000	0.000
Insurance_200	Empty	1.265	1.606	1.910	2.600	1.845
HailFinder	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder	S-DAG PMMS	1.005	0.986	1.036	1.003	1.007
HailFinder	S-DAG MMPC	1.005	1.000	0.988	0.954	0.987
HailFinder	MMSA	1.003	1.270	1.337	1.399	1.252
HailFinder	MMSA b	0.995	1.305	1.434	1.434	1.292
HailFinder	MMACO	1.007	1.177	1.204	1.302	1.172
HailFinder	MMACO b	1.003	1.176	1.103	1.372	1.164

Table A.2: Normalized “SHD” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
HailFinder	GS	0.851	1.038	1.095	1.071	1.014
HailFinder	SA	0.940	1.234	1.305	1.461	1.235
HailFinder	SA b	0.937	1.190	1.303	1.520	1.238
HailFinder	ACO	0.867	1.016	1.102	1.210	1.049
HailFinder	ACO b	0.891	1.004	1.092	1.166	1.038
HailFinder	True	0.000	0.000	0.000	0.000	0.000
HailFinder	Empty	0.900	1.385	1.570	1.877	1.433
HailFinder_10	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder_10	S-DAG PMMS	0.998	0.989	0.985	0.966	0.985
HailFinder_10	S-DAG MMPC	0.998	0.989	0.973	0.965	0.981
HailFinder_10	MMSA	1.006	1.052	1.071	0.986	1.029
HailFinder_10	MMSA b	1.005	1.055	1.072	0.990	1.030
HailFinder_10	MMACO	0.991	0.983	0.969	0.979	0.980
HailFinder_10	MMACO b	0.990	0.977	0.973	0.981	0.980
HailFinder_10	GS	0.884	0.962	0.984	0.935	0.941
HailFinder_10	SA	n/a	n/a	n/a	n/a	n/a
HailFinder_10	ACO	n/a	n/a	n/a	n/a	n/a
HailFinder_10	True	0.000	0.000	0.000	0.000	0.000
HailFinder_10	Empty	0.937	1.130	1.289	1.405	1.190

Table A.3: Normalized “BDeu score” results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm	S-DAG PMMS	0.997	1.000	0.997	1.001	0.999
Alarm	S-DAG MMPC	0.999	1.000	0.997	1.000	0.999
Alarm	MMSA	0.994	0.996	0.997	1.000	0.997
Alarm	MMSA b	0.994	0.996	0.997	1.000	0.997
Alarm	MMACO	0.994	0.996	0.997	1.000	0.997
Alarm	MMACO b	0.994	0.996	0.997	1.000	0.997
Alarm	GS	0.975	0.979	0.987	0.990	0.983
Alarm	SA	0.966	0.970	0.977	0.986	0.975
Alarm	SA b	0.964	0.969	0.977	0.986	0.974

Table A.3: Normalized “BDeu score” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	ACO	0.964	0.969	0.977	0.986	0.974
Alarm	ACO b	0.964	0.969	0.977	0.986	0.974
Alarm	True	0.979	0.975	0.980	0.987	0.980
Alarm	Empty	1.528	1.708	1.806	1.902	1.736
Alarm_10	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_10	S-DAG PMMS	0.995	0.994	0.992	0.993	0.994
Alarm_10	S-DAG MMPC	0.996	0.994	0.992	0.993	0.994
Alarm_10	MMSA	0.994	0.993	0.991	0.992	0.993
Alarm_10	MMSA b	0.993	0.993	0.991	0.992	0.993
Alarm_10	MMACO	0.994	0.993	0.991	0.992	0.993
Alarm_10	MMACO b	0.994	0.993	0.991	0.992	0.993
Alarm_10	GS	0.977	0.985	0.988	0.993	0.986
Alarm_10	SA	0.980	0.983	0.985	0.993	0.985
Alarm_10	SA b	0.976	0.979	0.983	0.991	0.982
Alarm_10	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_10	True	1.002	0.985	0.982	0.986	0.989
Alarm_10	Empty	1.389	1.509	1.574	1.662	1.534
Alarm_20	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_20	S-DAG PMMS	0.996	0.994	0.992	0.991	0.993
Alarm_20	S-DAG MMPC	0.996	0.994	0.992	0.992	0.994
Alarm_20	MMSA	0.994	0.994	0.992	0.991	0.993
Alarm_20	MMSA b	0.994	0.994	0.992	0.991	0.993
Alarm_20	MMACO	0.995	0.994	0.992	0.991	0.993
Alarm_20	MMACO b	0.994	0.994	0.992	0.991	0.993
Alarm_20	GS	0.979	0.983	0.988	0.994	0.986
Alarm_20	SA	n/a	n/a	n/a	n/a	n/a
Alarm_20	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_20	True	1.007	0.986	0.984	0.987	0.991
Alarm_20	Empty	1.378	1.487	1.551	1.631	1.512
Alarm_30	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_30	S-DAG PMMS	0.995	0.995	0.991	0.990	0.993
Alarm_30	S-DAG MMPC	0.996	0.995	0.992	0.990	0.993
Alarm_30	MMSA	0.993	0.994	0.991	0.989	0.992
Alarm_30	MMSA b	0.993	0.994	0.991	0.989	0.992
Alarm_30	MMACO	0.994	0.994	0.991	0.989	0.992
Alarm_30	MMACO b	0.994	0.994	0.991	0.989	0.992

Table A.3: Normalized “BDeu score” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm_30	GS	0.974	0.983	0.988	0.992	0.984
Alarm_30	SA	n/a	n/a	n/a	n/a	n/a
Alarm_30	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_30	True	1.003	0.987	0.984	0.986	0.990
Alarm_30	Empty	1.378	1.489	1.551	1.625	1.511
Alarm_50	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_50	S-DAG PMMS	0.996	0.994	0.992	0.990	0.993
Alarm_50	S-DAG MMPC	0.996	0.994	0.992	0.990	0.993
Alarm_50	MMSA	0.994	0.993	0.992	0.990	0.992
Alarm_50	MMSA b	0.994	0.993	0.992	0.990	0.992
Alarm_50	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	GS	n/a	n/a	n/a	n/a	n/a
Alarm_50	SA	n/a	n/a	n/a	n/a	n/a
Alarm_50	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	True	1.010	0.987	0.985	0.985	0.992
Alarm_50	Empty	1.378	1.490	1.553	1.628	1.512
Alarm_270	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_270	S-DAG PMMS	0.996	0.994	0.991	0.989	0.992
Alarm_270	S-DAG MMPC	0.997	0.995	0.991	0.989	0.993
Alarm_270	MMSA	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	GS	n/a	n/a	n/a	n/a	n/a
Alarm_270	SA	n/a	n/a	n/a	n/a	n/a
Alarm_270	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	True	1.015	0.987	0.982	0.983	0.992
Alarm_270	Empty	1.385	1.488	1.566	1.628	1.517
Insurance	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance	S-DAG PMMS	1.000	0.997	1.000	0.997	0.999
Insurance	S-DAG MMPC	1.000	0.996	1.000	0.997	0.998
Insurance	MMSA	0.999	0.995	1.000	0.997	0.998
Insurance	MMSA b	0.999	0.995	1.000	0.997	0.998
Insurance	MMACO	0.999	0.995	1.000	0.997	0.998
Insurance	MMACO b	0.999	0.995	1.000	0.997	0.998
Insurance	GS	0.996	0.979	0.977	0.986	0.984
Insurance	SA	0.992	0.976	0.974	0.983	0.981
Insurance	SA b	0.991	0.975	0.974	0.983	0.981

Table A.3: Normalized “BDeu score” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance	ACO	0.991	0.975	0.974	0.983	0.981
Insurance	ACO b	0.991	0.975	0.974	0.983	0.981
Insurance	True	1.077	1.012	0.995	0.987	1.018
Insurance	Empty	1.364	1.446	1.502	1.582	1.473
Insurance_10	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_10	S-DAG PMMS	0.994	0.983	0.985	0.980	0.985
Insurance_10	S-DAG MMPC	0.996	0.984	0.986	0.980	0.987
Insurance_10	MMSA	0.985	0.980	0.983	0.977	0.981
Insurance_10	MMSA b	0.985	0.980	0.983	0.976	0.981
Insurance_10	MMACO	0.986	0.980	0.983	0.975	0.981
Insurance_10	MMACO b	0.985	0.980	0.983	0.975	0.981
Insurance_10	GS	0.975	0.978	0.981	0.985	0.980
Insurance_10	SA	n/a	n/a	n/a	n/a	n/a
Insurance_10	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_10	True	1.028	0.985	0.975	0.965	0.988
Insurance_10	Empty	1.275	1.380	1.454	1.549	1.415
Insurance_20	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_20	S-DAG PMMS	0.995	0.984	0.981	0.972	0.983
Insurance_20	S-DAG MMPC	0.996	0.985	0.984	0.972	0.984
Insurance_20	MMSA	0.985	0.981	0.981	0.970	0.979
Insurance_20	MMSA b	0.985	0.981	0.981	0.969	0.979
Insurance_20	MMACO	0.986	0.982	0.981	0.968	0.979
Insurance_20	MMACO b	0.986	0.982	0.981	0.968	0.979
Insurance_20	GS	0.975	0.978	0.980	0.982	0.979
Insurance_20	SA	n/a	n/a	n/a	n/a	n/a
Insurance_20	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_20	True	1.008	0.972	0.961	0.958	0.975
Insurance_20	Empty	1.275	1.387	1.456	1.548	1.417
Insurance_30	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_30	S-DAG PMMS	0.995	0.986	0.982	0.978	0.985
Insurance_30	S-DAG MMPC	0.997	0.987	0.984	0.977	0.986
Insurance_30	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	GS	n/a	n/a	n/a	n/a	n/a
Insurance_30	SA	n/a	n/a	n/a	n/a	n/a
Insurance_30	ACO	n/a	n/a	n/a	n/a	n/a

Table A.3: Normalized "BDeu score" results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance_30	True	1.009	0.972	0.961	0.964	0.976
Insurance_30	Empty	1.274	1.389	1.456	1.560	1.420
Insurance_200	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_200	S-DAG PMMS	0.997	0.986	0.982	0.978	0.986
Insurance_200	S-DAG MMPC	0.999	0.988	0.984	0.977	0.987
Insurance_200	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	GS	n/a	n/a	n/a	n/a	n/a
Insurance_200	SA	n/a	n/a	n/a	n/a	n/a
Insurance_200	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	True	1.008	0.968	0.958	0.963	0.974
Insurance_200	Empty	1.274	1.384	1.454	1.558	1.417
HailFinder	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder	S-DAG PMMS	1.000	1.000	0.999	0.999	1.000
HailFinder	S-DAG MMPC	1.000	1.000	1.000	0.999	1.000
HailFinder	MMSA	0.999	1.007	1.007	1.005	1.004
HailFinder	MMSA b	0.999	1.000	0.999	0.999	0.999
HailFinder	MMACO	0.999	1.000	0.999	0.999	0.999
HailFinder	MMACO b	0.999	1.000	0.999	0.999	0.999
HailFinder	GS	0.940	0.997	0.999	0.999	0.984
HailFinder	SA	0.955	1.010	1.009	1.005	0.995
HailFinder	SA b	0.942	0.997	0.998	0.998	0.984
HailFinder	ACO	0.938	0.996	0.998	0.998	0.983
HailFinder	ACO b	0.938	0.996	0.998	0.998	0.982
HailFinder	True	0.988	1.034	1.027	1.010	1.015
HailFinder	Empty	1.171	1.312	1.349	1.393	1.306
HailFinder_10	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder_10	S-DAG PMMS	1.000	0.999	0.999	0.998	0.999
HailFinder_10	S-DAG MMPC	1.000	0.999	0.999	0.999	0.999
HailFinder_10	MMSA	0.994	1.008	1.004	0.998	1.001
HailFinder_10	MMSA b	0.993	1.005	1.002	0.998	1.000
HailFinder_10	MMACO	0.998	0.997	0.998	0.998	0.998
HailFinder_10	MMACO b	0.998	0.997	0.998	0.998	0.998
HailFinder_10	GS	0.936	0.988	0.987	0.929	0.960
HailFinder_10	SA	n/a	n/a	n/a	n/a	n/a
HailFinder_10	ACO	n/a	n/a	n/a	n/a	n/a

Table A.3: Normalized "BDeu score" results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
HailFinder_10	True	1.038	1.078	1.059	0.950	1.031
HailFinder_10	Empty	1.164	1.297	1.336	1.313	1.278

Table A.4: Normalized "NSC" results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm	S-DAG PMMS	0.970	0.990	1.061	0.945	0.992
Alarm	S-DAG MMPC	1.050	1.054	1.048	1.044	1.049
Alarm	MMSA	2.314	1.379	1.086	1.017	1.449
Alarm	MMSA b	2.320	1.380	1.088	1.019	1.452
Alarm	MMACO	1.861	1.381	1.188	1.190	1.405
Alarm	MMACO b	1.865	1.385	1.190	1.187	1.407
Alarm	GS	2.146	2.106	2.276	2.009	2.134
Alarm	SA	73.085	82.244	87.460	67.952	77.685
Alarm	SA b	73.290	82.380	87.458	67.825	77.738
Alarm	ACO	26.385	43.814	62.946	107.030	60.044
Alarm	ACO b	26.420	43.707	63.027	106.671	59.956
Alarm_10	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_10	S-DAG PMMS	1.000	0.993	0.989	0.981	0.991
Alarm_10	S-DAG MMPC	1.018	1.014	1.014	1.016	1.016
Alarm_10	MMSA	1.154	1.055	1.025	1.015	1.062
Alarm_10	MMSA b	1.154	1.055	1.024	1.014	1.062
Alarm_10	MMACO	1.105	1.065	1.050	1.081	1.075
Alarm_10	MMACO b	1.106	1.065	1.050	1.081	1.075
Alarm_10	GS	4.312	4.517	4.593	4.696	4.530
Alarm_10	SA	10.299	12.707	13.999	14.854	12.965
Alarm_10	SA b	10.216	12.718	14.026	14.894	12.963
Alarm_10	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_20	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_20	S-DAG PMMS	0.999	0.995	0.992	0.990	0.994
Alarm_20	S-DAG MMPC	1.010	1.008	1.008	1.009	1.009
Alarm_20	MMSA	1.119	1.042	1.022	1.011	1.049

Table A.4: Normalized “NSC” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm_20	MMSA b	1.119	1.042	1.022	1.011	1.049
Alarm_20	MMACO	1.055	1.032	1.024	1.042	1.038
Alarm_20	MMACO b	1.055	1.032	1.024	1.042	1.038
Alarm_20	GS	4.639	4.915	4.967	5.110	4.907
Alarm_20	SA	n/a	n/a	n/a	n/a	n/a
Alarm_20	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_30	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_30	S-DAG PMMS	0.999	0.994	0.993	0.992	0.995
Alarm_30	S-DAG MMPC	1.008	1.006	1.005	1.006	1.006
Alarm_30	MMSA	1.104	1.032	1.016	1.007	1.040
Alarm_30	MMSA b	1.104	1.032	1.016	1.007	1.040
Alarm_30	MMACO	1.044	1.023	1.017	1.027	1.028
Alarm_30	MMACO b	1.044	1.023	1.017	1.027	1.028
Alarm_30	GS	4.830	5.223	5.262	5.324	5.160
Alarm_30	SA	n/a	n/a	n/a	n/a	n/a
Alarm_30	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_50	S-DAG PMMS	0.999	0.994	0.993	0.994	0.995
Alarm_50	S-DAG MMPC	1.005	1.003	1.003	1.004	1.004
Alarm_50	MMSA	1.083	1.021	1.010	1.005	1.030
Alarm_50	MMSA b	1.083	1.021	1.010	1.005	1.030
Alarm_50	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	GS	n/a	n/a	n/a	n/a	n/a
Alarm_50	SA	n/a	n/a	n/a	n/a	n/a
Alarm_50	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_270	S-DAG PMMS	0.995	0.993	0.993	0.995	0.994
Alarm_270	S-DAG MMPC	1.003	1.001	1.001	1.001	1.001
Alarm_270	MMSA	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	GS	n/a	n/a	n/a	n/a	n/a
Alarm_270	SA	n/a	n/a	n/a	n/a	n/a
Alarm_270	ACO	n/a	n/a	n/a	n/a	n/a
Insurance	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance	S-DAG PMMS	0.970	0.950	0.865	0.724	0.877

Table A.4: Normalized “NSC” results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance	S-DAG MMPC	1.063	1.063	1.041	1.032	1.050
Insurance	MMSA	1.800	1.395	1.233	1.112	1.385
Insurance	MMSA b	1.800	1.397	1.234	1.115	1.386
Insurance	MMACO	1.578	1.341	1.284	1.465	1.417
Insurance	MMACO b	1.579	1.345	1.285	1.469	1.419
Insurance	GS	1.539	1.613	1.344	0.783	1.320
Insurance	SA	54.743	49.705	40.476	15.303	40.057
Insurance	SA b	54.547	49.547	40.161	15.228	39.871
Insurance	ACO	13.742	21.397	26.550	32.094	23.446
Insurance	ACO b	13.585	21.586	26.634	32.067	23.468
Insurance_10	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_10	S-DAG PMMS	1.003	0.991	0.956	0.799	0.937
Insurance_10	S-DAG MMPC	1.040	1.031	1.029	1.027	1.032
Insurance_10	MMSA	1.709	1.222	1.136	1.058	1.281
Insurance_10	MMSA b	1.709	1.223	1.135	1.057	1.281
Insurance_10	MMACO	1.231	1.191	1.221	1.341	1.246
Insurance_10	MMACO b	1.230	1.190	1.223	1.343	1.247
Insurance_10	GS	3.485	3.718	3.836	3.015	3.514
Insurance_10	SA	n/a	n/a	n/a	n/a	n/a
Insurance_10	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_20	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_20	S-DAG PMMS	1.002	0.996	0.976	0.863	0.959
Insurance_20	S-DAG MMPC	1.029	1.019	1.019	1.021	1.022
Insurance_20	MMSA	1.621	1.144	1.085	1.038	1.222
Insurance_20	MMSA b	1.622	1.144	1.085	1.038	1.222
Insurance_20	MMACO	1.163	1.115	1.127	1.226	1.158
Insurance_20	MMACO b	1.163	1.115	1.127	1.226	1.158
Insurance_20	GS	3.779	4.404	4.567	4.096	4.211
Insurance_20	SA	n/a	n/a	n/a	n/a	n/a
Insurance_20	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_30	S-DAG PMMS	1.008	0.994	0.979	0.886	0.967
Insurance_30	S-DAG MMPC	1.031	1.013	1.014	1.016	1.019
Insurance_30	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	GS	n/a	n/a	n/a	n/a	n/a

Table A.4: Normalized "NSC" results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance_30	SA	n/a	n/a	n/a	n/a	n/a
Insurance_30	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_200	S-DAG PMMS	1.010	0.996	0.993	0.975	0.994
Insurance_200	S-DAG MMPC	1.024	1.002	1.002	1.003	1.008
Insurance_200	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	GS	n/a	n/a	n/a	n/a	n/a
Insurance_200	SA	n/a	n/a	n/a	n/a	n/a
Insurance_200	ACO	n/a	n/a	n/a	n/a	n/a
HailFinder	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder	S-DAG PMMS	0.951	0.941	0.833	0.325	0.762
HailFinder	S-DAG MMPC	1.054	1.010	1.014	1.009	1.022
HailFinder	MMSA	2.303	1.633	1.433	1.160	1.632
HailFinder	MMSA b	2.297	1.644	1.441	1.147	1.632
HailFinder	MMACO	1.418	1.728	1.731	1.373	1.562
HailFinder	MMACO b	1.413	1.727	1.738	1.375	1.563
HailFinder	GS	1.846	1.442	1.012	0.255	1.139
HailFinder	SA	40.502	31.067	18.861	3.039	23.367
HailFinder	SA b	40.401	31.119	18.849	2.949	23.329
HailFinder	ACO	12.462	16.526	16.834	9.294	13.779
HailFinder	ACO b	12.482	16.301	16.867	9.280	13.733
HailFinder_10	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder_10	S-DAG PMMS	1.000	0.994	0.970	0.790	0.938
HailFinder_10	S-DAG MMPC	1.052	1.015	1.008	1.010	1.021
HailFinder_10	MMSA	2.619	1.205	1.117	1.050	1.498
HailFinder_10	MMSA b	2.617	1.205	1.117	1.051	1.498
HailFinder_10	MMACO	1.212	1.228	1.206	1.124	1.192
HailFinder_10	MMACO b	1.213	1.227	1.206	1.124	1.193
HailFinder_10	GS	2.807	3.119	3.268	2.979	3.043
HailFinder_10	SA	n/a	n/a	n/a	n/a	n/a
HailFinder_10	ACO	n/a	n/a	n/a	n/a	n/a

Table A.5: Normalized Runtime results

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm	S-DAG PMMS	1.441	1.292	1.471	1.065	1.317
Alarm	S-DAG MMPC	1.129	1.001	1.110	1.055	1.074
Alarm	MMSA	5.761	3.718	3.196	1.720	3.599
Alarm	MMSA b	6.593	4.359	3.491	1.821	4.066
Alarm	MMACO	15.306	12.216	11.532	6.224	11.320
Alarm	MMACO b	17.304	13.711	12.366	6.672	12.513
Alarm	GS	1.436	1.551	1.669	1.864	1.630
Alarm	SA	33.993	57.029	91.509	96.423	69.739
Alarm	SA b	35.572	58.636	93.100	97.458	71.191
Alarm	ACO	56.540	65.907	96.843	181.899	100.297
Alarm	ACO b	60.489	68.592	100.506	184.971	103.639
Alarm_10	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_10	S-DAG PMMS	1.072	1.014	1.005	1.034	1.031
Alarm_10	S-DAG MMPC	0.949	0.932	0.949	1.026	0.964
Alarm_10	MMSA	5.858	3.625	2.681	1.426	3.398
Alarm_10	MMSA b	5.995	3.659	2.695	1.419	3.442
Alarm_10	MMACO	27.310	17.767	11.511	4.355	15.236
Alarm_10	MMACO b	28.464	18.296	11.905	4.335	15.750
Alarm_10	GS	10.151	9.128	7.824	6.207	8.327
Alarm_10	SA	52.085	56.247	57.243	47.259	53.208
Alarm_10	SA b	52.899	57.501	58.817	47.970	54.297
Alarm_10	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_20	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_20	S-DAG PMMS	0.997	0.998	1.010	0.993	1.000
Alarm_20	S-DAG MMPC	0.929	0.932	0.950	0.977	0.947
Alarm_20	MMSA	10.115	5.987	3.682	1.626	5.353
Alarm_20	MMSA b	10.395	6.119	3.798	1.626	5.485
Alarm_20	MMACO	24.146	14.487	9.487	3.358	12.869
Alarm_20	MMACO b	24.486	14.721	9.605	3.383	13.049
Alarm_20	GS	14.724	12.594	9.966	6.959	11.061
Alarm_20	SA	n/a	n/a	n/a	n/a	n/a
Alarm_20	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_30	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_30	S-DAG PMMS	1.175	1.100	1.010	1.025	1.077
Alarm_30	S-DAG MMPC	1.081	1.028	0.964	1.004	1.019

Table A.5: Normalized Runtime results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Alarm_30	MMSA	10.929	5.410	3.320	1.500	5.290
Alarm_30	MMSA b	10.824	5.381	3.402	1.491	5.275
Alarm_30	MMACO	26.230	14.836	8.854	3.135	13.264
Alarm_30	MMACO b	26.679	15.051	8.966	3.157	13.463
Alarm_30	GS	21.528	17.193	12.308	7.805	14.708
Alarm_30	SA	n/a	n/a	n/a	n/a	n/a
Alarm_30	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_50	S-DAG PMMS	1.126	1.053	1.019	0.999	1.049
Alarm_50	S-DAG MMPC	1.050	1.016	1.011	1.006	1.021
Alarm_50	MMSA	10.165	4.572	3.098	1.449	4.821
Alarm_50	MMSA b	10.377	4.733	3.147	1.440	4.925
Alarm_50	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_50	GS	n/a	n/a	n/a	n/a	n/a
Alarm_50	SA	n/a	n/a	n/a	n/a	n/a
Alarm_50	ACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMHC	1.000	1.000	1.000	1.000	1.000
Alarm_270	S-DAG PMMS	1.109	1.011	0.990	0.991	1.025
Alarm_270	S-DAG MMPC	1.057	1.003	1.001	1.000	1.015
Alarm_270	MMSA	n/a	n/a	n/a	n/a	n/a
Alarm_270	MMACO	n/a	n/a	n/a	n/a	n/a
Alarm_270	GS	n/a	n/a	n/a	n/a	n/a
Alarm_270	SA	n/a	n/a	n/a	n/a	n/a
Alarm_270	ACO	n/a	n/a	n/a	n/a	n/a
Insurance	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance	S-DAG PMMS	1.473	1.437	1.088	0.753	1.188
Insurance	S-DAG MMPC	1.199	1.185	1.000	1.032	1.104
Insurance	MMSA	4.401	4.225	2.907	1.513	3.262
Insurance	MMSA b	5.176	5.406	3.668	1.464	3.928
Insurance	MMACO	12.704	11.381	8.529	3.850	9.116
Insurance	MMACO b	14.468	13.102	9.669	4.028	10.316
Insurance	GS	1.060	1.374	1.281	0.743	1.115
Insurance	SA	22.975	39.358	45.599	22.640	32.643
Insurance	SA b	23.863	40.629	45.467	22.564	33.131
Insurance	ACO	33.978	43.111	46.470	51.348	43.727
Insurance	ACO b	36.308	45.539	48.906	51.196	45.487

Table A.5: Normalized Runtime results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance_10	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_10	S-DAG PMMS	1.672	1.237	1.094	0.766	1.192
Insurance_10	S-DAG MMPC	1.513	1.148	1.085	1.035	1.195
Insurance_10	MMSA	16.353	8.704	4.485	1.429	7.743
Insurance_10	MMSA b	16.300	8.579	4.551	1.446	7.719
Insurance_10	MMACO	37.551	24.185	14.774	3.859	20.092
Insurance_10	MMACO b	38.089	24.736	15.263	3.853	20.485
Insurance_10	GS	7.246	7.324	6.828	3.528	6.231
Insurance_10	SA	n/a	n/a	n/a	n/a	n/a
Insurance_10	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_20	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_20	S-DAG PMMS	1.746	1.167	1.094	0.854	1.215
Insurance_20	S-DAG MMPC	1.670	1.082	1.055	1.031	1.209
Insurance_20	MMSA	16.925	7.193	4.321	1.485	7.481
Insurance_20	MMSA b	17.381	7.374	4.280	1.502	7.634
Insurance_20	MMACO	33.686	20.052	12.301	3.596	17.409
Insurance_20	MMACO b	34.750	20.811	12.699	3.617	17.969
Insurance_20	GS	10.531	11.433	9.779	5.311	9.264
Insurance_20	SA	n/a	n/a	n/a	n/a	n/a
Insurance_20	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_30	S-DAG PMMS	1.926	1.078	1.020	0.828	1.213
Insurance_30	S-DAG MMPC	1.960	1.075	1.051	1.026	1.278
Insurance_30	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_30	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_30	GS	n/a	n/a	n/a	n/a	n/a
Insurance_30	SA	n/a	n/a	n/a	n/a	n/a
Insurance_30	ACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMHC	1.000	1.000	1.000	1.000	1.000
Insurance_200	S-DAG PMMS	8.009	1.022	0.996	0.956	2.746
Insurance_200	S-DAG MMPC	8.235	1.015	1.009	1.005	2.816
Insurance_200	MMSA	n/a	n/a	n/a	n/a	n/a
Insurance_200	MMACO	n/a	n/a	n/a	n/a	n/a
Insurance_200	GS	n/a	n/a	n/a	n/a	n/a
Insurance_200	SA	n/a	n/a	n/a	n/a	n/a

Table A.5: Normalized Runtime results (Continued)

		Sample size				
Network	Method	200	500	1000	5000	Avg
Insurance_200	ACO	n/a	n/a	n/a	n/a	n/a
HailFinder	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder	S-DAG PMMS	1.671	1.388	1.012	0.294	1.091
HailFinder	S-DAG MMPC	1.504	1.116	1.006	0.997	1.156
HailFinder	MMSA	7.015	4.286	3.148	1.511	3.990
HailFinder	MMSA b	7.433	4.463	3.281	1.476	4.163
HailFinder	MMACO	20.439	15.428	11.086	2.571	12.381
HailFinder	MMACO b	21.050	16.175	11.486	2.604	12.829
HailFinder	GS	1.291	1.247	0.967	0.220	0.931
HailFinder	SA	28.594	33.044	25.540	4.015	22.798
HailFinder	SA b	30.336	33.890	26.146	3.903	23.569
HailFinder	ACO	93.391	97.492	101.939	29.004	80.457
HailFinder	ACO b	98.416	99.434	101.978	28.934	82.190
HailFinder_10	MMHC	1.000	1.000	1.000	1.000	1.000
HailFinder_10	S-DAG PMMS	5.622	1.306	1.045	0.710	2.171
HailFinder_10	S-DAG MMPC	5.521	1.307	1.072	1.014	2.229
HailFinder_10	MMSA	15.987	5.844	3.722	1.572	6.781
HailFinder_10	MMSA b	16.261	6.028	3.716	1.562	6.892
HailFinder_10	MMACO	37.290	18.722	11.488	3.020	17.630
HailFinder_10	MMACO b	37.923	19.074	11.683	3.037	17.929
HailFinder_10	GS	5.703	6.421	5.867	3.426	5.354
HailFinder_10	SA	n/a	n/a	n/a	n/a	n/a
HailFinder_10	ACO	n/a	n/a	n/a	n/a	n/a

Table A.6: Average normalized “SHD” results

	Sample size				
Method	200	500	1000	5000	Avg
MMHC	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000
S-DAG PMMS	0.975 ± 0.019	0.939 ± 0.021	0.896 ± 0.026	0.837 ± 0.032	0.912
S-DAG MMPC	0.971 ± 0.020	0.925 ± 0.019	0.887 ± 0.028	0.822 ± 0.030	0.901
MMSA	0.909 ± 0.043	0.905 ± 0.041	0.918 ± 0.047	0.856 ± 0.051	0.897
MMSA b	0.908	0.911	0.929	0.860	0.902
MMACO	0.912 ± 0.042	0.895 ± 0.043	0.905 ± 0.050	0.853 ± 0.053	0.891
MMACO b	0.910	0.893	0.894	0.864	0.890
GS	1.127 ± 0.067	1.312 ± 0.073	1.623 ± 0.124	1.910 ± 0.187	1.493
SA	0.913 ± 0.085	0.951 ± 0.102	1.090 ± 0.135	1.112 ± 0.168	1.017
SA b	0.925	0.914	1.123	1.113	1.019
ACO	0.741 ± 0.086	0.743 ± 0.108	0.894 ± 0.134	0.860 ± 0.165	0.810
ACO b	0.745	0.733	0.888	0.846	0.803
True	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000
Empty	1.254 ± 0.038	1.591 ± 0.046	2.137 ± 0.117	2.927 ± 0.191	1.977

Table A.7: Average normalized “BDeu Score” results

	Sample size				
Method	200	500	1000	5000	Avg
MMHC	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000
S-DAG PMMS	0.997 ± 0.001	0.993 ± 0.001	0.991 ± 0.001	0.989 ± 0.002	0.992
S-DAG MMPC	0.998 ± 0.001	0.993 ± 0.001	0.992 ± 0.001	0.989 ± 0.001	0.993
MMSA	0.993 ± 0.001	0.994 ± 0.002	0.994 ± 0.002	0.991 ± 0.002	0.993
MMSA b	0.993	0.993	0.993	0.990	0.992
MMACO	0.994 ± 0.001	0.992 ± 0.002	0.993 ± 0.002	0.990 ± 0.002	0.992
MMACO b	0.994	0.992	0.993	0.990	0.992
GS	0.970 ± 0.004	0.983 ± 0.004	0.986 ± 0.004	0.983 ± 0.004	0.981
SA	0.973 ± 0.006	0.984 ± 0.007	0.986 ± 0.007	0.992 ± 0.007	0.984
SA b	0.968	0.980	0.983	0.989	0.980
ACO	0.964 ± 0.005	0.980 ± 0.005	0.983 ± 0.006	0.989 ± 0.006	0.979
ACO b	0.964	0.980	0.983	0.989	0.979
True	1.013 ± 0.003	0.995 ± 0.003	0.987 ± 0.003	0.978 ± 0.003	0.993
Empty	1.326 ± 0.007	1.444 ± 0.006	1.508 ± 0.007	1.583 ± 0.007	1.465

Table A.8: Average normalized “NSC” results

	Sample size				
Method	200	500	1000	5000	Avg
MMHC	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000
S-DAG PMMS	0.993 ± 0.004	0.986 ± 0.004	0.969 ± 0.006	0.866 ± 0.006	0.953
S-DAG MMPC	1.030 ± 0.004	1.019 ± 0.003	1.016 ± 0.003	1.015 ± 0.003	1.020
MMSA	1.683 ± 0.042	1.213 ± 0.029	1.116 ± 0.024	1.047 ± 0.020	1.265
MMSA b	1.683	1.214	1.117	1.047	1.265
MMACO	1.296 ± 0.029	1.234 ± 0.023	1.205 ± 0.019	1.208 ± 0.017	1.236
MMACO b	1.296	1.234	1.207	1.208	1.236
GS	3.265 ± 0.068	3.451 ± 0.068	3.458 ± 0.081	3.141 ± 0.084	3.329
SA	44.657 ± 3.136	43.931 ± 2.545	40.199 ± 2.462	25.287 ± 2.034	38.519
SA b	44.613	43.941	40.123	25.224	38.475
ACO	17.530 ± 1.220	27.246 ± 1.130	35.444 ± 1.414	49.472 ± 1.447	32.423
ACO b	17.496	27.198	35.509	49.339	32.386

Table A.9: Average normalized “Runtime” results

	Sample size				
Method	200	500	1000	5000	Avg
MMHC	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000
S-DAG PMMS	2.234 ± 0.094	1.162 ± 0.062	1.066 ± 0.055	0.867 ± 0.045	1.332
S-DAG MMPC	2.138 ± 0.084	1.065 ± 0.058	1.020 ± 0.053	1.016 ± 0.041	1.310
MMSA	10.351 ± 0.884	5.356 ± 0.717	3.456 ± 0.604	1.523 ± 0.478	5.172
MMSA b	10.674	5.610	3.603	1.525	5.353
MMACO	26.074 ± 1.203	16.564 ± 0.844	11.062 ± 0.724	3.774 ± 0.570	14.369
MMACO b	27.023	17.297	11.516	3.854	14.923
GS	8.186 ± 0.311	7.585 ± 0.265	6.276 ± 0.266	4.007 ± 0.237	6.514
SA	34.412 ± 2.911	46.419 ± 2.364	54.973 ± 2.686	42.584 ± 2.482	44.597
SA b	35.667	47.664	55.882	42.973	45.547
ACO	61.303 ± 5.840	68.837 ± 4.435	81.751 ± 5.343	87.417 ± 5.027	74.827
ACO b	65.071	71.188	83.797	88.367	77.106

Bibliography

Abramson, B., Brown, J., Edwards, W., Murphy, A. & Winkler, R. L. (1996). Hailfinder: A Bayesian system for forecasting severe weather, *International Journal of Forecasting* **12**(1): 57–71.

Acid, S. & Campos, d. (2001). A hybrid methodology for learning belief networks : BENEDICT, *International Journal of Approximate Reasoning* **27**: 235–262.

Ali, A. R., Richardson, T. & Spirtes, P. (2004). Markov Equivalence for Ancestral Graphs, *Technical Report 466*, Department of Statistics, University of Washington, Seattle, WA, USA.

Aliferis, C. F., Tsamardinos, I. & Statnikov, A. (2003). HITON: a novel Markov Blanket algorithm for optimal variable selection., *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium* pp. 21–25.

Aliferis, C. F., Tsamardinos, I., Statnikov, A. R. & Brown, L. E. (2003). Causal Explorer: A Causal Probabilistic Network Learning Toolkit for Biomedical Discovery, *in* F. Valafar, H. Valafar, F. Valafar & H. Valafar (eds), *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Scienes, METMBS '03, June 23 - 26, 2003, Las Vegas, Nevada, USA*, CSREA Press, pp. 371–376.

Andersson, S. A., Madigan, D. & Madigan, D. (1997). A characterization of Markov equivalence classes for acyclic digraphs, *Annals of Statistics* **25**(2): 505–541.

Andersson, S. A., Madigan, D. & Perlman, M. D. (1997). On the Markov Equivalence of Chain Graphs, Undirected Graphs, and Acyclic Digraphs, *Scandinavian Journal of Statistics* **24**(1): 81–102.

Arndt, H., Bundschus, M. & Nägele, A. (2009). Towards a Next-Generation Matrix Library for Java, *2009 33rd Annual IEEE International Computer Software and Applications Conference*, IEEE, pp. 460–467.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M. & Sherlock, G. (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium., *Nature Genetics* **25**(1): 25–29.

Barabási, A. L. & Bonabeau, E. (2003). Scale-Free Networks, *Scientific American* **288**: 60–69.

Barabási, A. L. & Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization, *Nature Reviews Genetics* **5**(2): 101–113.

Basso, K., Margolin, A. A. A., Stolovitzky, G., Klein, U., Dalla-Favera, R. & Califano, A. (2005). Reverse engineering of regulatory networks in human B cells, *Nature Genetics* **37**(4): 382–390.

Beinlich, I. A., Suermondt, H. J., Chavez, R. M. & Cooper, G. F. (1989). The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks, *in* J. Hunter, J. Cookson & J. Wyatt (eds), *Second European Conference on Artificial Intelligence in Medicine*, Vol. 38, Springer-Verlag, Berlin, Germany, pp. 247–256.

Benjamini, Y. & Hochberg, Y. (1995). Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing, *Journal of the Royal Statistical Society. Series B (Methodological)* **57**(1): 289–300.

Bernard, A. & Hartemink, A. J. (2005). Informative structure priors: joint learning of dynamic regulatory networks from multiple types of data, World Scientific, Duke University, Dept of Computer Science, Durham, NC 27708, USA., pp. 459–470.

Binder, J., Koller, D., Russell, S. & Kanazawa, K. (1997). Adaptive Probabilistic Networks with Hidden Variables, *Machine Learning* **29**(2-3): 213–244.

Bouckaert, R. R. & Studený, M. (1998). On chain graph models for description of conditional independence structures, *Annals of Statistics* **26**(4): 1434–1495.

Brown, L. E., Tsamardinos, I. & Aliferis, C. F. (2005). A Comparison of Novel and State-of-the-Art Polynomial Bayesian Network Learning Algorithms, *in* M. M. Veloso, S. Kambhampati, M. M. Veloso & S. Kambhampati (eds), *Proceedings of the Twentieth National Conference on Artificial Intelligence*, AAAI Press / The MIT Press, Cambridge, MA, USA, pp. 739–745.

Castillo, E., Gutiérrez, J. M. & Hadi, A. S. (1996). *Expert Systems and Probabilistic Network Models*, 1 edn, Springer, New York, NY, USA.

Chen, X.-W., Anantha, G. & Lin, X. (2008). Improving Bayesian Network Structure Learning with Mutual Information-Based Node Ordering in the K2 Algorithm, *Knowledge and Data Engineering, IEEE Transactions on* **20**(5): 628–640.

Cheng, J. (2002). Learning Bayesian networks from data: An information-theory based approach, *Artificial Intelligence* **137**(1-2): 43–90.

Chickering, D. M. (1995). A Transformational Characterization of Equivalent Bayesian Network Structures, *in* P. Besnard & S. Hanks (eds), *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 87–98.

Chickering, D. M. (2002). Learning Equivalence Classes of Bayesian-Network Structures, *Journal of Machine Learning Research* **2**: 445–498.

Chickering, D. M. (2003). Optimal structure identification with greedy search, *Journal of Machine Learning Research* **3**: 507–554.

Chickering, D. M., Geiger, D. & Heckerman, D. (1994). Learning Bayesian Networks is NP-Hard, *Technical Report MSR-TR-94-17*, Microsoft Research, Redmond, WA, USA.

Cooper, G. F. & Herskovits, E. (1991). A Bayesian method for constructing Bayesian belief networks from databases, *in* G. F. Cooper & S. Moral (eds), *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 86–94.

Cowell, R. G., Dawid, P. A., Lauritzen, S. L. & Spiegelhalter, D. J. (2003). *Probabilistic Networks and Expert Systems (Information Science and Statistics)*, Springer, New York.

Darvasi, A. (2003). Genomics: Gene expression meets genetics, *Nature* **422**(6929): 269–270.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*, 1 edn, Cambridge University Press.

Dash, D. & Druzdzel, M. J. (1999). A Hybrid Anytime Algorithm for the Construction of Causal Models From Sparse Data, *Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence*, pp. 142–149.

Davis, G. A. (2003). Bayesian reconstruction of traffic accidents, *Law Probability and Risk* **2**(2): 69–89.

de Campos, C. P. & Ji, Q. (2011). Efficient Structure Learning of Bayesian Networks using Constraints, *Journal of Machine Learning Research* **12**: 663–689.

de Campos, L. M. (2006). A Scoring Function for Learning Bayesian Networks based on Mutual Information and Conditional Independence Tests, *Journal of Machine Learning Research* **7**: 2149–2187.

de Campos, L. M., Fernandez-Luna, J. M., Gamez, J. A. & Puerta, J. M. (2002). Ant colony optimization for learning Bayesian networks, *International Journal of Approximate Reasoning* **31**(3): 291–311.

de Campos, L. M. & Puerta, J. M. (2001). Stochastic Local Algorithms for Learning Belief Networks: Searching in the Space of the Orderings, *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, ECSQARU '01, Springer-Verlag, London, UK, pp. 228–239.

de Freitas, J. M., Kim, J. H., Poynton, H. C., Su, T., Wintz, H., Fox, T. C., Holman, P. S., Loguinov, A. V., Keles, S., Van Der Laan, M. & Vulpe, C. D. (2003). Exploratory and confirmatory gene expression profiling of mac1Delta, *The Journal Biological Chemistry* **279**(6): 4450–8.

Dejori, M. (2005). *Inference Modeling of Gene Regulatory Networks*, PhD thesis, TU München, Garching, Germany.

Dejori, M., Schürmann, B. & Stetter, M. (2004). Hunting Drug Targets by Systems-Level Modeling of Gene Expression Profiles, *IEEE Transactions on Nanobioscience* **3**(3): 180–191.

Dejori, M. & Stetter, M. (2003). Bayesian Inference of Genetic Networks from Gene-Expression Data: Convergence and Reliability, *in* H. R. Arubnia, R. Joshua & Y. Mun (eds), *Proceedings of the 2003 International Conference on Artificial Intelligence*, CSREA Press, pp. 321–327.

Dejori, M. & Stetter, M. (2004). Identifying Interventional and Pathogenic Mechanisms by Generative Inverse Modeling of Gene Expression Profiles, *Journal of Computational Biology* **11**(6): 1135–1148.

Díez, F. J., Mira, J., Iturralde, E. & Zubillaga, S. (1997). DIAVAL, a Bayesian expert system for echocardiography, *Artificial Intelligence in Medicine 10*, Vol. 1, pp. 59–73.

Doctor, J. N. & Strylewicz, G. (2010). Detecting 'wrong blood in tube' errors: Evaluation of a Bayesian network approach, *Artificial intelligence in medicine* **50**(2): 75–82.

Dor, D. & Tarsi, M. (1992). A simple algorithm to construct a consistent extension of a partially oriented graph, *Technical Report R-185*, Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA, USA.

Dorigo, M. & Stützle, T. (2004). *Ant Colony Optimization*, MIT Press.

Efron, B. & Tibshirani, R. J. (1994). *An Introduction to the Bootstrap (Chapman & Hall/CRC Monographs on Statistics & Applied Probability)*, 1 edn, Chapman & Hall/CRC.

Friedman, N. (1997). Learning Belief Networks in the Presence of Missing Values and Hidden Variables, *in* D. H. Fisher (ed.), *Proc. 14th International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 125–133.

Friedman, N. (1998). The Bayesian structural EM algorithm, *in* G. F. Cooper & S. Moral (eds), *Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 129–138.

Friedman, N., Goldszmidt, M. & Wyner, A. (1999). Data Analysis with Bayesian Networks: A Bootstrap Approach, *in* K. B. Laskey & H. Prade (eds), *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 196–205.

Friedman, N. & Koller, D. (2003). Being Bayesian About Network Structure. A Bayesian Approach to Structure Discovery in Bayesian Networks, *Machine Learning* **50**(1): 95–125.

Friedman, N., Linial, M., Nachman, I. & Pe'er, D. (2000). Using Bayesian networks to analyze expression data, *in* R. Shamir, S. Miyano, S. Istrail, P. Pevzner & M. Waterman (eds), *The Fourth Annual International Conference on Computational Molecular Biology (RECOMB)*, ACM, New York, NY, USA, pp. 127–135.

Friedman, N., Nachman, I. & Pe'er, D. (1999). Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm, *in* K. B. Laskey & H. Prade (eds), *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, USA, pp. 206–215.

Frydenberg, M. (1990). The chain graph Markov property, *Scandinavian Journal of Statistics* **17**: 333–353.

Gámez, J., Mateo, J. & Puerta, J. (2007). A Fast Hill-Climbing Algorithm for Bayesian Networks Structure Learning, *in* K. Mellouli (ed.), *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Vol. 4724 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Berlin, Heidelberg, chapter 52, pp. 585–597.

Gámez, J., Mateo, J. & Puerta, J. (2011). Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood, *Data Mining and Knowledge Discovery* **22**(1): 106–148.

Gámez, J. & Puerta, J. (2005). Constrained Score+(Local)Search Methods for Learning Bayesian Networks, *in* L. Godo (ed.), *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Vol. 3571 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Berlin, Heidelberg, chapter 15, p. 470.

Geiger, D., Verma, T. & Pearl, J. (1990). Identifying independence in Bayesian Networks, *Networks* **20**: 507–534.

Goldenberg, A. & Moore, A. (2004). Tractable Learning of Large Bayes Net Structures from Sparse Data, *in* C. E. Brodley (ed.), *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, Vol. 69 of *ACM International Conference Proceeding Series*, ACM.

Grzegorczyk, M. & Husmeier, D. (2008). Learning Bayesian networks from postgenomic data with animproved structure MCMC sampling scheme, *Learning and Inference in Computational Systems Biology (LICSB)*.

Hartemink, A. J., Gifford, D. K., Jaakkola, T. S. & Young, R. A. (2002). Combining location and expression data for principled discovery of genetic regulatory network models., *Pacific Symposium on Biocomputing* pp. 437–449.

Hartemink, A. J., Gifford, D. K., Jaakkola, T. & Young, R. A. (2001). Using Graphical Models and Genomic Expression Data to Statistically Validate Models of Genetic Regulatory Networks, *in* R. B. Altman, K. A. Dunker & L. Hunker (eds), *Pacific Symposium on Biocomputing*, World Scientific Publishing, pp. 422–433.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* **57**(1): 97–109.

Heckerman, D. (1995). A Tutorial on Learning With Bayesian Networks, *Technical report*, Microsoft Research, Redmond, WA, USA.

Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R. & Kadie, C. (2001). Dependency networks for inference, collaborative filtering, and data visualization, *Journal of Machine Learning Research* **1**: 49–75.

Heckerman, D., Geiger, D. & Chickering, D. M. (1995). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data, *Machine Learning* **20**(3): 197–243.

Heckerman, D., Meek, C. & Cooper, G. (1997). A Bayesian Approach to Causal Discovery, *Technical report*, Microsoft Research.

Herscovici, A. & Brock, O. (2007). Improved search for structure learning of large bayesian networks, *Technical report*, Department of Computer Science, University of Massachusetts Amherst.

Hofmann, R. (2000). *Lernen der Struktur nichtlinearer Abhängigkeiten mit graphischen Modellen*, PhD thesis, TU München, Germany.

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., Kidd, M. J., King, A. M., Meyer, M. R., Slade, D., Lum, P. Y., Stepaniants, S. B., Shoemaker, D. D., Gachotte, D., Chakraburtty, K., Simon, J., Bard, M. & Friend, S. H. (2000). Functional discovery via a compendium of expression profiles., *Cell* **102**(1): 109–126.

Hulten, G., Chickering, D. M. & Heckerman, D. (2003). Learning Bayesian Networks From Dependency Networks: A Preliminary Study, *in* C. M. Bishop & B. J. Frey (eds), *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

Husmeier, D., Dybowski, R. & Roberts, S. (eds) (2004). *Probabilistic Modelling in Bioinformatics and Medical Informatics*, 1st edition. edn, Springer.

igc (2004). Nation's first clinically annotated, publicly available gene expression database to help researchers accelerate cancer treatment discoveries, International Genomics Consortium (IGC), Press release, http://www.intgen.org/expo_scientific_release.cfm.

Imoto, S., Goto, T. & Miyano, S. (2002). Estimation of Genetic Networks and Functional Structures Between Genes by Using Bayesian Networks and Nonparametric Regression, *Pacific Symposium on Biocomputing*, pp. 175–186.

Jensen, A. L. & Jensen, F. V. (1996). MIDAS: An Influence Diagram for Management of Mildew in Winter Wheat, *in* E. Horvitz, F. V. Jensen, E. Horvitz & F. V. Jensen (eds), *UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 349–356.

Jensen, F. V. (2002). *Bayesian Networks and Decision Graphs*, Information Science and Statistics, corrected edn, Springer, New York.

Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N. & Barabási, A. L. (2000). The large-scale organization of metabolic networks, *Nature* **407**: 651–654.

Ji, J., Hu, R., Zhang, H. & Liu, C. (2011). A hybrid method for learning Bayesian networks based on ant colony optimization, *Applied Soft Computing* **11**(4): 3373–3384.

Ji, J.-Z., Zhang, H.-X., Hu, R.-B. & Liu, C.-N. (2009). A Bayesian Network Learning Algorithm Based on Independence Test and Ant Colony Optimization, *Acta Automatica Sinica* **35**(3): 281–288.

Jordan, M. (2004). Graphical Models, *Statistical Science (Special Issue on Bayesian Statistics)* **19**: 140–155.

Kanehisa, M. & Goto, S. (2000). KEGG: Kyoto Encyclopedia of Genes and Genomes, *Nucleic Acids Research* **28**(1): 27–30.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. (1983). Optimization by Simulated Annealing, *Science* **220**(4598): 671–680.

Ko, S., Kim, D. & Kan, B.-Y. (2011). A Matrix-Based Genetic Algorithm for Structure Learning of Bayesian Networks, *International Journal of Fuzzy Logic and Intelligent Systems* **11**(3): 135–216.

Kojima, K., Perrier, E., Imoto, S. & Miyano, S. (2010). Optimal Search on Clustered Structural Constraint for Learning Bayesian Network Structure, *Journal of Machine Learning Research* **11**: 285–310.

Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques (Adaptive Computation and Machine Learning series)*, 1 edn, The MIT Press.

Kraaijeveld, P. & Druzdzel, M. (2005). GeNIeRate: An interactive generator of diagnostic Bayesian network models, *16th International Workshop on Principles of Diagnosis*, pp. 175–180.

Laarhoven, P. J. M. & Aarts, E. H. L. (eds) (1987). *Simulated annealing: theory and applications*, Kluwer Academic Publishers, Norwell, MA, USA.

Lam, W. & Bacchus, F. (1994). Using New Data to Refine a Bayesian Network, *In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 383–390.

Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R. H. & Kuijpers, C. M. H. (1996). Structure Learning of Bayesian Networks by Genetic Algorithms: A Performance Analysis of Control Parameters, *IEEE Trans. Pattern Anal. Mach. Intell.* **18**: 912–926.

Lauritzen, S. L. (1996). *Graphical Models*, Oxford Statistical Science Series, Oxford University Press, New York, USA.

Lauritzen, S. L. & Wermuth, N. (1984). Mixed interaction models, *Technical Report R-84-8*, Institute of Electronic Systems, Aalborg University, Aalborg, Denmark.

Lauritzen, S. L. & Wermuth, N. (1998). Graphical Models for Associations between Variables, some of which are Qualitative and some Quantitative, *Annals of Statistics* **17**(1): 31–57.

Lee, J., Chung, W., Kim, E. & Kim, S. (2010). A new genetic approach for structure learning of Bayesian networks: Matrix genetic algorithm, *International Journal of Control, Automation and Systems* **8**(2): 398–407.

Lee, P. H. & Lee, D. (2005). Modularized learning of genetic interaction networks from biological annotations and mRNA expression data, *Bioinformatics* **21**(11): 2739–2747.

Li, X. (2009). Learning Structure of Bayesian Network Using Ant Colony Algorithm Assisted by Genetic Algorithm, *Intelligent Systems and Applications, 2009. ISA 2009.*, IEEE, pp. 1–4.

Listgarten, J. & Heckerman, D. (2007). Determining the Number of Non-Spurious Arcs in a Learned DAG Model: Investigation of a Bayesian and a Frequentist Approach, *Proceedings of the Twenty-Third Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-07)*, AUAI Press, Corvallis, Oregon, pp. 251–258.

Madigan, D., Raftery, A., Wermuth, N., York, J. & Zucchini, W. (1994). Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window, *Journal of the American Statistical Association* **89**: 1535–1546.

Madigan, D., York, J. & Allard, D. (1995). Bayesian Graphical Models for Discrete Data, *International Statistical Review / Revue Internationale de Statistique* **63**(2): 215–232.

Margolin, A. A., Banerjee, N., Nemenman, I. & Califano, A. (2004). Reverse engineering of yeast transcriptional network using the ARACNE algorithm, Columbia University, http://www.menem.com/˜ilya/wiki/index.php/Margolin_et_al.%2C_2004b.

McCarthy, J. (1987). Epistemological problems of artificial intelligence, *Readings in non-monotonic reasoning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 46–52.

McCarthy, J., Minsky, M. L., Rochester, N. & Shannon, C. E. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

Meek, C. (1995a). Causal Inference and Causal Explanation with Background Knowledge, *in* P. Besnard & S. Hanks (eds), *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, Morgan Kaufmann, San Francisco, CA, USA, pp. 403–441.

Meek, C. (1995b). Strong completeness and faithfulness in Bayesian networks, *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 411–418.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines, *The Journal of Chemical Physics* **21**(6): 1087–1092.

Moore, A. & Wong, W.-K. (2003). Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning, *In Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pp. 552–559.

Nägele, A. (2005). *Bayesian Network Structure Learning in High-Dimensional Spaces*, Master's thesis.

Nägele, A., Arndt, H. & Dejori, M. (2008). Large-Scale Genetic Network Learning, *in* A. Tsakalides & L. Skarlas (eds), *ECAI'08 Workshop on Bioinformatics, Genomics & Proteomics, an Artificial Intelligence Approach.*

Nägele, A., Dejori, M. & Stetter, M. (2007). Bayesian Substructure Learning - Approximate Learning of Very Large Network Structures, *in* J. N. Kok, J. Koronacki, R. L. de Mántaras, S. Matwin, D. Mladenic & A. Skowron (eds), *Machine Learning: ECML 2007. 18th European Conference on Machine Learning*, Vol. 4701 of *Lecture Notes in Computer Science*, Springer, Berlin / Heidelberg, Germany, pp. 238–249.

Nägele, A., Dejori, M. & Stetter, M. (2009). *Robust Learning of High-dimensional Biological Networks with Bayesian Networks*, Springer, Heidelberg, Germany, pp. 139–161.

Nielsen, S. H. & Nielsen, T. D. (2008). Adapting Bayes network structures to non-stationary domains, *International Journal of Approximate Reasoning* **49**(2): 379–397.

Peña, J. M., Björkegren, J. & Tegnér, J. (2005). Growing Bayesian network models of gene networks from seed genes, *Bioinformatics* **21**(suppl 2): ii224–ii229.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 1 edn, Morgan Kaufmann, Santa Mateo, CA, USA.

Pearl, J. & Verma, T. S. (1991). A Theory of Inferred Causation, *in* J. F. Allen, R. Fikes & E. Sandewall (eds), *KR'91: Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, San Mateo, CA, USA, pp. 441–452.

Pe'er, D., Regev, A., Elidan, G. & Friedman, N. (2001a). Inferring subnetworks from perturbed expression profiles, *Bioinformatics* **17**(suppl 1): S215–S224.

Pe'er, D., Regev, A., Elidan, G. & Friedman, N. (2001b). Inferring Subnetworks from Perturbed Expression Profiles, *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pp. 215–224.

Philpott, C. C., Protchenko, O., Kim, Y. W., Boretsky, Y. & Shakoury-Elizeh, M. (2002). The response to iron deprivation in Saccharomyces cerevisiae: expression of siderophore-based systems of iron uptake., *Biochemical Society Transactions* **30**(4): 698–702.

Pinto, P. C., Nägele, A., Dejori, M., Runkler, T. A. & Sousa, J. M. C. (2008). Learning of Bayesian networks by a local discovery ant colony algorithm, *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, pp. 2741–2748.

Pinto, P. C., Nägele, A., Dejori, M., Runkler, T. A. & Sousa, J. M. C. (2009). Using a local discovery ant algorithm for Bayesian network structure learning, *IEEE Trans. Evol. Comp* **13**(4): 767–779.

Protchenko, O., Ferea, T., Rashford, J., Tiedeman, J., Brown, P. O., Botstein, D. & Philpott, C. C. (2001). Three cell wall mannoproteins facilitate the uptake of iron in Saccharomyces cerevisiae, *The Journal of Biological Chemistry* **276**(52): 49244–50.

Resnik, P. (1999). Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language, *Journal of Artificial Intelligence Research* **11**: 95–130.

Richardson, T. & Spirtes, P. (2002). Ancestral graph Markov models, *Annals of Statistics* **30**(4): 962–1030.

Roos, T., Silander, T., Kontkanen, P. & Myllymaki, P. (2008). Bayesian network structure learning using factorized NML universal models, *2008 Information Theory and Applications Workshop*, IEEE, pp. 272–276.

Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*, third edn, Prentice Hall.

Scholz, J., Dejori, M., Stetter, M. & Greiner, M. (2005). Noisy scale-free networks, *Physica A Statistical Mechanics and its Applications* **350**: 622–642.

Schulte, O., Frigo, G., Greiner, R. & Khosravi, H. (2010). The IMAP Hybrid Method for Learning Gaussian Bayes Nets, *Canadian Conference on AI*, pp. 123–134.

Schwarz, G. (1978). Estimating the dimension of a model, *Annals of Statistics* **6**: 461–464.

Scutari, M. & Brogini, A. (2011). Bayesian Network Structure Learning with Permutation Tests, *Statistics for Complex Problems: the Multivariate Permutation Approach and Related Topics*.

Silverstein, C., Brin, S., Motwani, R. & Ullman, J. (2000). Scalable Techniques for Mining Causal Structures, *Data Min. Knowl. Discov.* **4**(2): 163–192.

Simon, H. A. (1965). *The Shape of Automation for Men and Management*, Harper & Row, New York.

Spirtes, P., Glymour, C. & Scheines, R. (2001). *Causation, Prediction, and Search*, second edn, The MIT Press, Cambridge, MA, USA.

Spirtes, P., Richardson, T. & Meek, C. (1997). Heuristic Greedy Search Algorithms for Latent Variable Models, *AI & STAT '97*, pp. 481–488.

Steck, H. (2001). *Constraint-Based Structural Learning in Bayesian Networks using Finite Data Sets*, PhD thesis, Department of Informatics, Technical University Munich, Munich, Germany.

Stetter, M., Nägele, A. & Dejori, M. (2007). GeneSim$^{\mathrm{TM}}$: Intelligent IT Platform for the Biomedical World, *in* A. Schuster (ed.), *Intelligent Computing Everywhere*, Springer, Heidelberg, Germany.

Stützle, T., López-Ibáñez, M., Dorigo, M., Cochran, J. J., Cox, L. A., Keskinocak, P., Kharoufeh, J. P. & Smith, J. C. (2011). *A Concise Overview of Applications of Ant Colony Optimization*, John Wiley & Sons, Inc.

Tamada, Y., Imoto, S. & Miyano, S. (2011). Parallel Algorithm for Learning Optimal Bayesian Network Structure, *Journal of Machine Learning Research* .

Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing* **1**(2): 146–160.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D. & Altman, R. B. (2001). Missing value estimation methods for DNA microarrays., *Bioinformatics (Oxford, England)* **17**(6): 520–525.

Tsamardinos, I., Aliferis, C. F. & Statnikov, A. (2003). Time and sample efficient discovery of Markov blankets and direct causal relations, *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, New York, NY, USA, pp. 673–678.

Tsamardinos, I. & Brown, L. E. (2008). Bounding the false discovery rate in local Bayesian network learning, *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI Press, pp. 1100–1105.

Tsamardinos, I., Brown, L. E. & Aliferis, C. (2006). The max-min hill-climbing Bayesian network structure learning algorithm, *Machine Learning* **65**(1): 31–78.

Tsamardinos, I., Statnikov, A. R., Brown, L. E. & Aliferis, C. F. (2006). Generating Realistic Large Bayesian Networks by Tiling, *in* G. Sutcliffe, R. Goebel, G. Sutcliffe & R. Goebel (eds), *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, AAAI Press, Menlo Park, California, USA, pp. 592–597.

Ueno, M. (2011). Robust learning Bayesian networks for prior belief, *in* F. G. Cozman & A. Pfeffer (eds), *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 698–707.

Verma, T. & Pearl, J. (1992). An algorithm for deciding if a set of observed independencies has a causal explanation, *in* D. Dubois, M. P. Wellman, B. D'Ambrosio & P. Smets (eds), *Eighth Conference on Uncertainty in Artificial intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, USA.

Verma, T. S. & Pearl, J. (1990). Equivalence and synthesis of causal models, *in* P. P. Bonissone, M. Henrion, L. N. Kanal & J. F. Lemmer (eds), *UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, Elsevier Science Publishers B.V., North Holland, pp. 255–268.

Winograd, T. (1972). *Understanding Natural Language*, Academic Press, Inc., Orlando, FL, USA.

Wu, Y., McCall, J. & Corne, D. (2010). Two novel Ant Colony Optimization approaches for Bayesian network structure learning, *IEEE Evolutionary Computation (CEC)*, pp. 1–7.

Wu, Y., McCall, J. & Corne, D. (2011). Comparative Analysis of Search and Score Metaheuristics for Bayesian Network Structure Learning Using Node Juxtaposition Distributions, *in* R. Schaefer, C. Cotta, J. Kolodziej & G. Rudolph (eds), *Parallel Problem Solving from Nature PPSN XI*, Vol. 6238 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Berlin, Heidelberg, chapter 43, pp. 424–433.

Yehezkel, R. & Lerner, B. (2009). Bayesian Network Structure Learning by Recursive Autonomy Identification, *Journal of Machine Learning Research* **10**: 1527–1570.

Yeoh, E. J., Ross, M. E., Shurtleff, S. A., Williams, K. W., Patel, D., Mahfouz, R., Behm, F. G., Raimondi, S. C., Relling, M. V., Patel, A. & Cheng (2002). Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling, *Cancer Cell* **1**(2): 133–143.

Yuan, C., Malone, B. & Wu, X. (2011). Learning Optimal Bayesian Networks Using A* Search, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 2186–2191.

Zeng, Y. & Hernandez, J. C. (2008). A decomposition algorithm for learning Bayesian network structures from data, *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'08, Springer-Verlag, Berlin, Heidelberg, pp. 441–453.

Zhang, J. & Spirtes, P. (2005). A Transformational Characterization of Markov Equivalence for Directed Maximal Ancestral Graphs, *Technical Report 168*, Department of Philosophy, Carnegie Mellon University.

Zhu, X., Gerstein, M. & Snyder, M. (2007). Getting connected: analysis and principles of biological networks., *Genes & Development* **21**(9): 1010–1024.

Zoubir, A. M. (1993). Bootstrap: theory and applications, *in* F. T. Luk (ed.), *Advanced Signal Processing Algorithms, Architectures, and Implementations IV*, Vol. 2027, SPIE, pp. 216–235.