Department of Computer Science
Series of Publications A
Report A-2015-2

Approximation Strategies for Structure Learning in Bayesian Networks

Teppo Niinimäki

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium CK112, Exactum, Gustaf Hällströmin katu 2b, on August 21st, 2015, at twelve o'clock noon.

University of Helsinki
Finland

Supervisor
Mikko Koivisto, University of Helsinki, Finland

Pre-examiners
Jaakko Hollmén, Aalto University, Finland
Jin Tian, Iowa State University, United States

Opponent
Timo Koski, Royal Institute of Technology, Sweden

Custos
Jyrki Kivinen, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911, telefax: +358 9 876 4314

ISSN 1238-8645
ISBN 978-951-51-1445-7 (paperback)
ISBN 978-951-51-1446-4 (PDF)
Computing Reviews (1998) Classification: F.2.1, F.2.2, G.1.2, G.2.1, G.3, I.2.6, I.2.8, I.5.1
Helsinki 2015
Unigrafia

Approximation Strategies for Structure Learning in Bayesian Networks

Teppo Niinimäki

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
teppo.niinimaki@helsinki.fi
http://www.cs.helsinki.fi/u/tzniinim/

PhD Thesis, Series of Publications A, Report A-2015-2
Helsinki, Aug 2015, 64+93 pages
ISSN 1238-8645
ISBN 978-951-51-1445-7 (paperback)
ISBN 978-951-51-1446-4 (PDF)

Abstract

Bayesian networks are probabilistic graphical models, which can compactly represent complex probabilistic dependencies between a set of variables. Once learned from data or constructed by some other means, they can both give insight into the modeled domain and be used for probabilistic reasoning tasks, such as prediction of future data points.

Learning a Bayesian network consists of two tasks: discovering a graphical dependency structure on variables, and finding the numerical parameters of a conditional distribution for each variable. Structure discovery has attracted considerable interest in the recent decades. Attention has mostly been paid to finding a structure that best fits the data under certain criterion. The optimization approach can lead to noisy and partly arbitrary results due to the uncertainty caused by a small amount of data. The so-called full Bayesian approach addresses this shortcoming by learning the posterior distribution of structures. In practice, the posterior distribution is summarized by constructing a representative sample of structures, or by computing marginal posterior probabilities of individual arcs or other substructures.

This thesis presents algorithms for the full Bayesian approach to structure learning in Bayesian networks. Because the existing exact algorithms only scale to small networks of up to about 25 variables, we investigate sampling

based, Monte Carlo methods. The state-of-the-art sampling algorithms draw orderings of variables along a Markov chain. We propose several improvements to this algorithm. First, we show that sampling partial orders instead of linear orders can lead to radically improved mixing of the Markov chain and consequently better estimates. Second, we suggest replacing Markov chain Monte Carlo by annealed importance sampling. This can further improve the accuracy of estimates and has also other advantages such as independent samples and easy parallelization. Third, we propose a way to correct the bias that is caused by sampling orderings of variables instead of structures. Fourth, we present an algorithm that can significantly speed up per-sample computations via approximation.

In addition, the thesis proposes a new algorithm for so-called local learning of the Bayesian network structure. In local learning the task is to discover the neighborhood of a given target variable. In contrast to previous algorithms that are based on conditional independence tests between variables, our algorithm gives scores to larger substructures. This approach often leads to more accurate results.

Computing Reviews (1998) Categories and Subject Descriptors:

F.2.1 [Analysis of algorithms and problem complexity] Numerical Algorithms and Problems
F.2.2 [Analysis of algorithms and problem complexity] Nonnumerical Algorithms and Problems
G.2.1 [Discrete mathematics] Combinatorics – combinatorial algorithms, counting problems
G.3 [Probability and statistics] multivariate statistics, probabilistic algorithms (including Monte Carlo)
I.2.6 [Artificial intelligence] Learning – knowledge acquisition, parameter learning
I.2.8 [Artificial intelligence] Problem Solving, Control Methods, and Search – dynamic programming, heuristic methods
I.5.1 [Pattern recognition] Models – statistical, structural

General Terms:

algorithms, design, experimentation, theory

Additional Key Words and Phrases:

Bayesian networks, structure learning, Markov chain Monte Carlo

Acknowledgements

First and foremost, I want to thank my advisor Mikko Koivisto for patient guidance through the process of writing this thesis. In addition to the advisor, the credit also goes to my co-author Pekka Parviainen. I want to thank the pre-examiners for their constructive comments. I am also grateful to Heikki Mannila who recruited me as a research assistant in the first place.

I appreciate the financial support that I received from Hecse (Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems).

Finally, I would also like to thank my current and former colleagues Esa Junttila, Juho-Kustaa Kangas, Jussi Kollin, Janne Korhonen and Jarkko Toivonen.

Helsinki, July 2015
Teppo Niinimäki

Contents

Original publications

This thesis is based on the following publications, which are referred to in the text as Papers I–V. The papers are reprinted at the end of the thesis.

I. Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Partial order MCMC for structure discovery in Bayesian networks. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 557–565. AUAI Press, 2011.

II. Teppo Niinimäki and Mikko Koivisto. Annealed importance sampling for structure learning in Bayesian networks. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1579–1585. AAAI Press, 2013.

III. Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Structure discovery in Bayesian networks by sampling partial orders. (Submitted).

IV. Teppo Niinimäki and Mikko Koivisto. Treedy: A heuristic for counting and sampling subsets. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 469–477. AUAI Press, 2013.

V. Teppo Niinimäki and Pekka Parviainen. Local structure discovery in Bayesian networks. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 634–643. AUAI Press, 2012.

Chapter 1

Introduction

Bayesian networks [59] are graphical models that describe probabilistic dependencies between a set of variables. Although the term "Bayesian network" seems predominant today, they have been also called by other names, such as (Bayesian) belief networks, causal networks or just directed graphical models. While directed models had seen some restricted use in various applications before, it was the theoretical development in the late 1980s [44, 59] that started their widespread acceptance. In the recent decades they have attracted a lot of interest in a growing number of application domains. Initially, Bayesian networks were mainly used in expert systems—typically related to medical diagnosis—to model uncertain expert knowledge, but since then they have spread to numerous fields such as bioinformatics, forensic science, reliability analysis and terrorism risk management [61].

The core of a Bayesian network is a directed acyclic graph (DAG), called also the *structure*. The DAG consists of nodes that correspond to a set of random variables, and arcs that tell which variables might depend probabilistically on each others. Figure 1.1 shows a classic example network, that depicts how some diseases (tuberculosis, lung cancer, bronchitis), their possible causes (visit to Asia, smoking) and possible symptoms (positive X-ray result, dyspnea) are related. While the example tells, for instance, that smoking has some effect on the probability of getting a lung cancer, it does not reveal the exact nature of the dependencies. To describe how the related variables are dependent, a Bayesian network also contains a set of *parameters* that define a set of conditional probability distributions, one for each node in the network. The conditional distribution of each variable tells how it is distributed when conditioned on the values of its *parents*, that is, the variables from which there is an arc pointing to it. Figure 1.2 shows

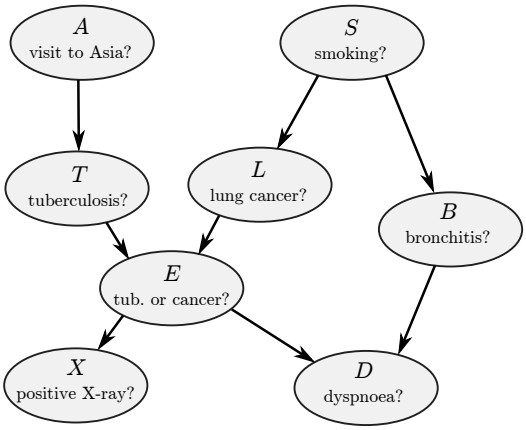


Figure 1.1: An example (fictional) Bayesian network describing the relations of some diseases, environmental factors and symptoms. dyspnea (shortness of breath) may be caused by either tuberculosis, lung cancer or bronchitis, or none of them. Both tuberculosis and lung cancer can be detected in a single chest X-ray test but not distinguished from each others. A recent visit to Asia increases the risk of getting tuberculosis and smoking is a known risk factor for getting lung cancer or bronchitis. (This classic example was first presented by Lauritzen and Spiegelhalter [44].)

these parameters for the example network in Figure 1.1. For example, the probability that a (random) person smokes does not directly depend on any other variable and has been estimated to be 0.50. The probability of a person having lung cancer depends on whether he/she smokes or not and has been estimated to be 0.05 for smokers and 0.01 for non-smokers.

Since Bayesian networks are directed graphs, it is natural to use them to model causality—the directions of arcs correspond to the direction of causation. Indeed, in the network of Figure 1.1 most arcs are causal, an exception being the relations between nodes T, L and E. But it is not always possible (or required) to determine the causal ordering of dependent events. An often-mentioned example of such a situation is the relation between ice cream sales and drownings: There is said to be a statistical correlation between the sales figures of ice cream and the number of deaths by drowning, while neither is a cause of the other. Generally, in some domains the notion of causality does not even make sense, and even if it does, determining causality often requires either additional knowledge or some kind of intervention in the process that generates the observations. However, even in such cases, a compatible but more general semantic can be used, where the arcs only encode the probabilistic dependencies between the

$\Pr(A = 1)$
0.01

$\Pr(S = 1)$
0.50

A	$\Pr(T = 1 \mid A)$
0	0.01
1	0.05

S	$\Pr(L = 1 \mid S)$
0	0.01
1	0.10

S	$\Pr(B = 1 \mid S)$
0	0.30
1	0.60

E	$\Pr(X = 1 \mid E)$
0	0.05
1	0.98

L	T	$\Pr(E = 1 \mid L, T)$
0	0	0
0	1	1
1	0	1
1	1	1

E	B	$\Pr(D = 1 \mid E, B)$
0	0	0.10
0	1	0.70
1	0	0.80
1	1	0.90

Figure 1.2: Conditional probability distributions for the Bayesian network in Figure 1.1. Each variable is binary and takes values 0 and 1 representing the false and true states of the corresponding condition respectively. For each variable a table that contains the conditional probability of getting value 1 for all possible combinations of its parent variable values is shown. (As the structure in Figure 1.1, the probabilities are also due to Lauritzen and Spiegelhalter [44].)

variables. This more general probabilistic interpretation is used throughout the rest of this thesis.

Once a Bayesian network that models a specific problem domain has been constructed, it can be used for tasks such as the *inference* of the values of unobserved variables based on the observed variables, or the *prediction* of future data points. In the context of the example network of Figure 1.1, an inference task could be to compute the probability that a person has tuberculosis if he has dyspnea, the X-ray gives a positive result, he is a non-smoker and has visited to Asia recently. A prediction task, although a bit unusual in this context, could be to compute the probability that a random person has bronchitis but no dyspnea.

How can one construct a Bayesian network that accurately describes the domain of interest? This is the question that we study in this thesis. Especially the Bayesian networks used in early expert systems were typically hand crafted based on expert knowledge. But such expert knowledge is not always available, or construction by hand might be exceedingly laborious. However, often there is a set of jointly observed values of the variables, that is, *data*, available from the domain. In this case, it is possible to automatically *learn* (or infer) the network from data.

Learning a Bayesian network can be divided into two subtasks: learning the structure and learning the parameters. If the structure is known or fixed, then learning the parameters is known to be computationally relatively easy, at least if there are no missing values in the data [16, 31]. While in many applications the structure is either obvious or otherwise

known, in other cases learning the structure is also required. This is especially the case in data analysis tasks, where not much is known about the data beforehand and the main interest is in learning the structure while the parameters are more of a nuance and may be ignored completely.

There are two main approaches to the structure learning problem. The *constraint-based* approach [74, 67, 60, 68, 42] is based on the fact that a structure corresponds to a certain set of conditional independences between the variables. By conducting a set of statistical independence tests, the DAG can be reconstructed piece by piece. The main problems of this approach are the lack of robustness of independence tests and inability to address the uncertainty about the structure. In the *score-based* approach [16, 32] a real-valued score is assigned to each possible DAG based on how well the DAG fits the data. The learning problem is therefore transformed into an optimization task. Compared to the constraint-based approach, the score-based approach typically yields better results, but as a downside it is often computationally harder and can thus be impractical for datasets with a large number of variables. Therefore, algorithms for restricted cases, such as learning tree-like networks, as well as heuristic methods for the general case, such as algorithms based on greedy search, have been introduced [16, 32, 69, 14]. Recently, however, there have also been several improvements to the efficiency of exact score-based learning [55, 64, 65, 78, 34, 18].

In score-based approaches, the scoring criteria typically fall in two categories: information theoretic scores that combine the maximum likelihood of the structure with a penalty for structure complexity, and Bayesian scores that consist of the marginal likelihood of the structure and possible prior knowledge. Bayesian scoring is particularly interesting since it naturally allows a full Bayesian approach. By considering the full posterior distribution of structures it is possible to properly handle the uncertainty about the structure. While explicitly describing the full posterior is not practical, it can be summarized by computing statistics of interest: In addition to finding a most probable (highest scoring) structure, one can compute posterior probabilities of structural features such as the presence of an arc between two given nodes. A full Bayesian approach is useful especially when there are many structures with a high score and thus finding only one of them would not be representative enough.

Computing the posterior probability of a structural feature is a hard problem in general. The existing exponential algorithms can solve the problem exactly for moderate-size networks: The state-of-the-art methods scale up to 20 variables [70] or up to around 25 variables if special type of prior knowledge about the structure is assumed [39, 38]. Several

sampling based estimation algorithms have also been proposed. There are two main approaches, which are both based on the *Markov chain Monte Carlo* method (MCMC) [48, 29] but which differ on how the samples are formed: A straightforward approach, suggested by Madigan and York [46] and later extended by others [20, 28, 17], is to sample a set of structures along a Markov chain whose stationary distribution coincides with the posterior distribution of structures. Feature probabilities can be estimated by taking the average over samples. The other approach, by Friedman and Koller [23], combines sampling and exact averaging over restricted subsets of possible structures. The approach is based on two observations: First, every structure is acyclic and thus has at least one linear extension, that is, an ordering of the variables that agrees with the directions of all arcs. And second, if a variable ordering is fixed, then the average over the corresponding structures can be relatively efficiently computed exactly. The algorithm draws samples along a Markov chain defined over orderings and for each sampled ordering it computes the corresponding average exactly. As the orderings form a sampling space that is smaller and smoother than the space of the structures, this leads to better mixing of the Markov chain and consequently better estimates.

While being arguably an improvement over structure sampling, the question remains whether ordering-based sampling could be further improved in various ways. More specifically, this thesis addresses the following four questions:

(I) Although Markov chains over orderings typically mix much better than Markov chains over structures, sometimes bad mixing can still be an issue. Can the mixing be significantly further improved, either by further modifying the sampling space or by adopting some standard techniques to enhance MCMC methods?

(II) What is common to most MCMC based algorithms is that the generated samples are not independent. Independence could potentially allow some additional theoretical guarantees, such as high-confidence bounds for the estimated quantities. Is it possible to draw independent samples?

(III) Ordering-based sampling causes a bias by favoring structures that have a large number of linear extensions. Although this bias can be viewed as a result of a special prior knowledge that is part of the model, such priors are often unwanted. Is it possible to get rid of the bias?

(IV) Compared to structure sampling, the per-sample computations in ordering-based sampling are much more expensive and therefore fewer samples can be drawn within the same time budget. Can the per-sample computations be sped up significantly?

This thesis addresses the above questions by proposing several improvements to ordering-based sampling. Paper I, Paper II and Paper III study questions I, II and III. The goal of Paper I is to improve mixing by taking another step in to the direction of exact computations. Inspired by the ordering-based MCMC and recent advances in exact algorithms for full Bayesian structure learning [58], the paper proposes a MCMC algorithm that samples partial orderings of variables. Paper II aims to improve mixing and convergence further by applying widely known tempering methods, Metropolis coupled Markov chain Monte Carlo (MC^3) [25] and annealed importance sampling (AIS) [49]. The latter method of the two provides also a solution to question II and has some other nice properties. In order to tackle question III, the paper proposes a simple importance sampling scheme that involves nestedly sampling weighted structures from orderings. Paper III refines and generalizes the results from Papers I and II and introduces some technical improvements that reduce the time and space requirements of the algorithms.

Paper IV concentrates on question IV. The per-sample computations of ordering-based sampling mostly consist of computing large sums of real values that describe how likely different sets of parents are for each variable. In order to speed up these computations, the paper introduces a greedy heuristic that, for any query set of variables, is able to approximate the sum over the subsets of the query set within a guaranteed error bound.

While the main focus of this thesis is on Bayesian structure learning, the thesis also proposes a new approach to a slightly different type of problem: *local structure learning* [1]. In contrast to *global structure learning* that was discussed above, in local learning the interest is restricted to the local substructure around a fixed target variable. More specifically, for a given target variable, the task is to find either the *neighbors* of the target, that is, the other variables that are connected to the target via an arc, or the *Markov blanket* of the target, that is, the minimum set of other variables that explains all the dependencies between the target and the remaining variables. Local learning can be useful if the interest lies only in a certain small part of the structure and a large total number of variables makes learning the full structure infeasible. Another possible motivation is variable selection: For example, if the purpose is to classify the target variable based on the values of the others, typically only a small portion of other

variables is needed to obtain the optimal prediction power. Indeed, the maximum predictivity with the minimum number of predictors is provided by the Markov blanket of the target. The question is, how to learn the neighbors and/or the Markov blanket of a target efficiently without a need to construct the full structure. Several algorithms for local learning exist [41, 47, 71, 1] that all use constraint-based approaches. This leads to the fifth research question of this thesis:

(V) Are there other, either faster or more accurate, ways to solve the local learning problem?

This thesis addresses question V by suggesting a score-based algorithm for the local learning problem. Paper V presents an algorithm that is a variant of the Generalized Local Learning framework introduced by Aliferis and Tsamardinos [1]. The main difference is that, instead of using statistical independence tests, the proposed algorithm constructs the neighborhood and the Markov blanket of the target based on the results of repeated score-based searches for optimal substructures.

The rest of this introductory part of the thesis is organized as follows: Chapter 2 specifies in more detail the learning tasks that this thesis proposes solutions for. The next two chapters concentrate on full Bayesian structure learning. Chapter 3 shows how to apply different Markov chain Monte Carlo based methods and use different sampling spaces to estimate posterior probabilities of structural features. Chapter 4 provides efficient algorithms for the per-sample computation problems that need to be solved in order to be able to us the methods introduced in the previous chapter. Together, Chapters 3 and 4 serve as an introduction to the algorithms and the results that are presented in Papers I–IV. Finally, Chapter 5 concentrates on local structure learning and outlines the approach and the results of Paper V.

1.1 Author contributions

Each paper was jointly written by all authors of the paper. The other contributions were as follows:

Paper I: The idea of combining the MCMC approach and the advances in exact computation was due to Pekka Parviainen and Mikko Koivisto. The implementation and the experiments were conducted by the present author.

Paper II: The algorithmic ideas (the ideas of applying AIS and obtaining lower bounds, as well as the algorithm for counting linear extensions)

were mainly due to Mikko Koivisto. The implementation and the experiments were conducted by the present author.

Paper III: The mentioned improvements to the algorithms were mainly due to the present author. The implementation and the experiments were conducted by the present author.

Paper IV: The algorithms and results were joint work by Mikko Koivisto and the present author. The implementation and the experiments were conducted by the present author.

Paper V: The algorithms and results were joint work by Pekka Parviainen and the present author. The implementation and the experiments were conducted by the present author.

Chapter 2

Preliminaries

The purpose of this chapter is to define the concept of a Bayesian network, explain some of its important properties and introduce the concept of structure learning. After that, the rest of the chapter acts mainly as a preparation for Chapters 3 and 4. The concepts that are only related to local learning and are thus needed only in Chapter 5, will be introduced there.

2.1 Bayesian networks

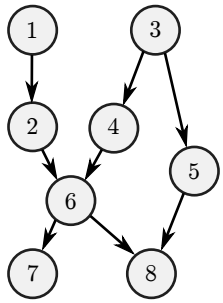
Let $X_1, X_2, \ldots, X_n$ be n random variables. For each random variable X_v, denote by $\mathcal{X}_v$ its state space, that is, the set of values it can take. For any subset $S \subseteq \{1, \ldots, n\}$, denote by X_S the vector that consists of the random variables indexed by the elements of S (in order to make the ordering of the random variables in X_S unique, we assume that they are sorted by the indices), and by $\mathcal{X}_S = \times_{v \in S} \mathcal{X}_v$ the corresponding joint state space.

The goal is to model the joint distribution of the random variables, denoted by $p(X_1, \ldots, X_n)$. To be able to do probabilistic inference, or maybe to just discover how the variables are statistically related, one usually has to express the joint distribution in some form. The variables may be discrete or continuous, but for now assume that they are discrete. To describe the joint distribution, one possibility is to list the probabilities of all joint value assignments the variables can take. (In fact, one of them can be left out as its value follows from the fact that the probabilities need to sum up to one.) However, there are $\prod_{v=1}^{n} |\mathcal{X}_v|$ such value configurations, a number that grows exponentially with respect to n, so explicit listing is not practical unless n is very small. From this explicit representation, it is also usually hard to make any interesting findings about the properties of the

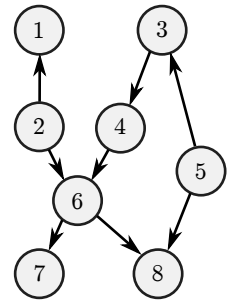
distribution. Fortunately, interesting real world distributions are typically sparse in a sense that they contain lots of (conditional) independences. There are several ways to exploit the independences and describe the joint distribution more succinctly, and often also in a more human friendly manner. Typically they are based on a graphical representation; the random variables are represented as nodes and the dependencies between them as either undirected edges or directed arcs between the nodes [59, 40]. In this thesis we restrict our attention to directed graphical models, usually called *Bayesian networks* [59].

Bayesian networks are based on the fact that by using the chain rule the joint distribution decomposes into $p(X_1)p(X_2 \,|\, X_1) \cdots p(X_n \,|\, X_1, \ldots X_{n-1})$, a product of n terms where each term is a conditional probability distribution. One could describe the joint distribution by encoding these n conditional distributions, but this would not yet result in any savings in the total number probabilities that need to be listed. Now consider for instance the fifth term of the decomposition, $p(X_5 \,|\, X_1, X_2, X_3, X_4)$. If we know that X_4 is independent of X_2 and X_4 given X_1 and X_3, then the term can be replaced by an equal term $p(X_5 \,|\, X_1, X_3)$. In this case we call X_1 and X_3 the *parents* of X_5. The distribution conditioned by only two parents can be encoded more compactly than the original distribution with five conditioning variables. In this case the number of free parameters required to encode the term is reduced from $(|\mathcal{X}_5| - 1)|\mathcal{X}_1||\mathcal{X}_2||\mathcal{X}_3||\mathcal{X}_4|$ to only $(|\mathcal{X}_5| - 1)|\mathcal{X}_1||\mathcal{X}_3|$. By doing a similar replacement for every term in the decomposition, we obtain an equal decomposition in which the vth term depends only on X_v and its parents. Typically we want to minimize the number of parents for each variable. If the distribution has a lot of conditional independences, then the variables will have only few parents, which leads to much more compact representation of the conditional distributions. While in many cases there is a unique minimal set of parents for each variable, this is not always true. Note also that the order in which the chain rule is applied, can be arbitrary; with different orderings one may (or may not) obtain different sets of parents. Indeed, it might be the case, that one ordering leads to a very compact encoding but another ordering does not yield much savings.

Consider an arbitrary order in which the parents for variables are determined. The parent relations can be encoded as a directed acyclic graph (DAG for short), $G = (N, A)$ with the node set $N = \{1, \ldots, n\}$ and the arc set $A \subset N \times N$, as follows. Each node $v \in N$ corresponds to a random variable X_v. The arc set, or the *structure* as we will call it, determines the parents: there is an arc from u to v, that is $(u, v) \in A$, if and only if X_u is a parent of X_v. In this case we also say that u is a parent of v, that v is a



(a) The structure of the Asia network, repeated from Figure 1.1 but with renamed nodes.



(b) Another structure on the same node set, that belongs to the same equivalence class.

Figure 2.1: Two examples of (DAG) structures with the same skeleton.

child of u, and that u and v are *neighbors*. We denote the set of all parents (that is, *parent set*) of v by A_v. Furthermore, if there is a directed path from u to v, we say that u is a *descendant* of v and that v is an *ancestor* of u. If u and v are not neighbors but have a common child w, then u and w are said to be *spouses* and u, w and v are said to form a *v-structure*. From this point on, we may refer to a node and the corresponding random variable interchangeably when the meaning is obvious.

Example 2.1. Figure 2.1a shows an examples of a (DAG) structure. In the structure the node 6 has two parents, nodes 2 and 4, two children, nodes 7 and 8, and one spouse, node 5. There are two v-structures in the structures, one formed by nodes 2, 6, and 4, and another formed by nodes 5, 8 and 6.

We are now ready to define a Bayesian network. By the definition of the structure A above, the joint distribution of $X_1, \ldots, X_n$ can be written as a product $\prod_{v \in N} p(X_v \,|\, X_{A_v})$. The terms $p(X_v \,|\, X_{A_v})$ are called *local conditional probability distributions*, or *LCPD*s for short. If the structure A is given, then the joint distribution can be fully characterized by defining the LCPDs for each node in the structure. Formally, a Bayesian network is a pair (G, θ) that consists of a DAG $G = (N, A)$ with n nodes and a parameter vector of n nonnegative functions $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$, each θ_v satisfying $\sum_{x_v \in \mathcal{X}_v} \theta_v(x_v; x_{A_v}) = 1$ for all $x_{A_v} \in \mathcal{X}_{A_v}$. Together they define a joint distribution

$$p_{(G,\theta)}(X_1, \ldots, X_n) = \prod_{v \in N} \theta_v(X_v; X_{A_v}). \tag{2.1}$$

From this definition, it then follows that the local conditional distributions are directly defined by the parameters, that is, $p_{(G,\theta)}(X_v \mid X_{A_v}) = \theta_v(X_v; X_{A_v})$, and conversely the parameters are directly determined by conditional probabilities. In the case of continuous variables the definition a Bayesian network is otherwise similar, but the parameters θ_v define continuous distributions, usually taken from some parametrized family.

Consider then an arbitrary structure A and an arbitrary joint distribution p. We say that A *contains* p, if p can be encoded by a Bayesian network with A as a structure, or equivalently, if p decomposes according to A into a product of local distributions conditioned on the parent sets. Thus, a complete structure (one which has an arc between every pair of nodes) contains every distribution and absence of arcs encodes conditional independences between the random variables. More specifically, the *local Markov property* holds: each variable X_v is conditionally independent of its non-descendants given its parent variables X_{A_v}. Moreover, if the structure contains no arcs, then all the variables must be mutually independent. If A contains p and, in addition, all independences in p are implied by A, then it is said that p is *faithful* to A and that A is a *perfect map* of p [59, 68]. A necessary but not sufficient requirement for this is that the parent sets in A are minimal with respect to p. If A is faithful to some structure, then A is said to be faithful. Not all distributions are faithful. On the other hand, some faithful distributions have several perfect maps, all implying the same set of independences.

If two structures contain exactly the same set of distributions, or equivalently, imply the same set of conditional independences, then those structures are called *Markov equivalent*. This equivalence relation partitions the set of all structures into *equivalence classes*. It can be shown that an equivalence class consists of exactly those structures which have the same *skeleton*, that is, structure from which the arc directions are removed, and the same set of v-structures [74]. The arcs that are not part of any v-structures can be directed arbitrarily as long as no additional v-structures are formed.

Example 2.2. Figures 2.1a and 2.1b contain two equivalent structures that differ in the directions of two arcs: the arc between nodes 1 and 2, and the arc between nodes 3 and 5. Reversing these arcs does not affect the skeleton nor does it remove or create any v-structures.

2.2 Structure learning

Learning a Bayesian network from data may involve finding the parameter, the structure, or both. If the structure is given, then it is rather straightforward to learn the parameters [59, 16, 31]. Structure learning, however, has turned out to be much more challenging. This thesis concentrates solely on structure learning.

In structure learning we are given m samples of *data*. We make the standard assumption that the samples are i.i.d. draws from an (unknown) n-dimensional distribution p, often called the *data generating distribution*. Our goal is to learn a structure $A \subseteq N \times N$ that best fits the data in the sense that it describes well the distribution p. We model the data as an $m \times n$ matrix, denoted by D. Each row of the matrix contains a single sample from p. From now on, we refer to rows/samples as *data points*. Each column of the matrix corresponds to a single node $v \in N$ and is denoted by $D_v = (D_v^1, \ldots, D_v^m)$, where D_v^i is the vth value of the ith data point.

Since we would like to find a structure that best fits the data (or some aspect of such a structure), the next step is to define what does a good fit mean. Unfortunately, there is no definite answer, but some properties can be seen desirable. Clearly we would like the structure to contain the data generating distribution. On the other hand, we would like to avoid overfitting and thus only include necessary arcs to keep the total number of free parameters low. Thus, if possible, a perfect map would seem to be a best fit. But if the data generating distribution has many perfect maps, which one should we choose? And what should we do if the distribution does not have a perfect map? Another issue is that with a finite amount of data it is not even possible to deduce the correct generating distribution with full certainty. We would also like to address this uncertainty about the distribution.

As explained briefly in Chapter 1, the two main approaches to structure learning are the constraint-based and the score-based approach, both having their strengths and weaknesses. Since understanding the constraint-based approach will be useful later in Chapter 5, we are going to next give a very short introduction to constraint-based learning. After that, we concentrate on the score-based approach, which is used in Chapters 3 and 4.

2.2.1 Constraint-based learning

In the constraint-based approach the idea is to use the fact that the structure implies a certain set of conditional independence statements between the variables, and if we want the structure to be minimal, then conditional

independence statements also also imply some properties of the structure. By performing statistical independence tests (in the case of discrete data usually χ^2-test or G-test) one can examine whether those potential independence statements hold in the data. From each test result, some properties of the structure (for example, existence and/or direction of an arc) can be inferred. There are several algorithms that define which potential independences should be tested, in which order, and how the result should be interpreted [68, 74, 60].

Most constraint-based algorithms assume that the data generating distribution is faithful [15]. Usually these algorithms have been designed to be sound in the sense that, under the faithfulness assumption the algorithm returns a correct result (that is, a perfect map of the distribution) in the limit of large sample size, that is, when the number data points approaches infinity. Although this property is useful to have, it does not tell much about the performance of the algorithm on small data sizes.

While being exponential in the worst case, in practice, constraint-based algorithms are usually relatively fast and can scale to networks of hundreds or thousands of nodes [42]. If the number of variables is small, the dependences are relatively strong and the data has a lot of observations, then constraint-based approaches often work well. Unfortunately, they are quite sensitive to incorrect results from independence tests [13, 42]. If the data size is small or the dependencies are weak, then score-based algorithms often yield better results [77].

2.2.2 Score-based learning

In the score-based approach one assigns each structure A a real valued *score*, denoted by $s(A)$, that measures the goodness of fit to the data; a best fitting structure can be determined by finding one with the largest score. The problem of constructing the structure thus becomes a discrete optimization problem, which can be solved in numerous ways, either with an exact algorithm or with some kind of heuristic search.

While in theory the score s could be an arbitrary function, in practice some properties are required, either to make the search computationally more efficient, or for it to match what we think is a good fit to the data. A scoring function s is said to be *modular*, if it factorizes into a product of *local scores* as $s(A) = \prod_v s_v(A_v)$, a property that is crucial for many algorithms including those that we are going to present. (In the literature the property is often called *decomposability*, but for consistency with the similar properties of prior and likelihood we call it modularity.) There are several popular scoring functions that are all modular and typically belong

to one of two classes: The scores in the first class are based on the maximum log likelihood of the structure combined with some typically information theoretic penalization term, such as AIC or BIC/MDL [9, 13, 42], and the scores in the second class are based on the Bayesian posterior probability of the structure, such as K2 and BDeu [16, 32].

In addition to modularity, it is often also wanted that the score has some other nice properties. Since structures that are Markov equivalent can represent the same set of distributions, they are sometimes considered indistinguishable, and it is often desired that the scoring function gives them a same score. Such scoring functions are called *score equivalent*. Often it is also desirable to have some guarantees of soundness in the limit of large sample size. A scoring function is said to be *consistent* [14], if in the limit of large sample size: (1) a structure that contains the data generating distribution always has a higher score that a structure that does not, and (2) if two structures contain the data generating distribution, the structure with fewer free parameters has a higher score. Faithfulness together with a score equivalent consistent scoring function guarantees that, in the limit of large sample size, a structure with the highest score is a perfect map of the data generating distribution [14].

2.2.3 Bayesian learning

As mentioned, an important family of scoring functions is based on the Bayesian approach [16, 32]. The scores belonging to this family are often collectively called the *Bayesian score*. In the Bayesian approach the structure A and the parameter θ of the generating Bayesian network, as well as the data D itself, are treated as random variables. The data are assumed to consist of independent draws from the distribution defined by the structure and the parameters as given in Equation 2.1. It then follows that, given A and θ, the conditional probability of the data factorizes into a product

$$p(D \,|\, \theta, A) = \prod_{v \in N} p(D_v \,|\, D_{A_v}, A_v, \theta_v)$$

of local conditional probabilities $p(D_v \,|\, D_{A_v}, A_v, \theta_v) = \prod_{i=1}^{m} \theta_v(D_v^i; D_{A_v}^i)$. In addition to the conditional distribution of the data, a probability distribution $p(\theta, A)$, called a *prior*, is defined so that it depicts the modelers initial beliefs of the structure and the parameters before any data has been seen. The prior is usually split into a product $p(\theta \,|\, A)p(A)$ where $p(A)$ is called the *structure prior* and $p(\theta \,|\, A)$ is called the *parameter prior*. The product of the prior and the conditional probability of the data gives us a full joint model $p(D, \theta, A)$. The Bayesian score is then defined to be

proportional to the posterior probability of the structure given the data, which, using Bayes' rule can be expressed as

$$p(A \mid D) = \frac{p(D \mid A)\, p(A)}{p(D)}\,,$$

where $p(D \mid A)$ is called the *likelihood* of structure A and $p(D)$ is called the *normalizing constant* (sometimes also called the *marginal likelihood*). Note that the parameters are marginalized out in the likelihood term and thus do not appear in the equation. Since $p(D)$ does not help distinguish different structures it can be often ignored. Motivated by this, we define the Bayesian score as $p(D \mid A)p(A)$.[1]

The Bayesian approach is interesting not only because it is theoretically justifiable, but also because by considering the full structure posterior it imposes a way to take into account the aforementioned uncertainty in the structure. While in most cases it is not practical to describe the posterior explicitly, it is possible to construct a representative sample of high-probability structures or compute different summaries. Finding a structure with the highest posterior probability is one way to summarize, but another one is computing posterior expectations of so called *structural features*, an approach sometimes called *Bayesian averaging*. For example, Bayesian averaging can be used to find posterior probabilities for presence of arcs or other substructures.

Formally, a structural feature is a function f that maps structures to real values. In Bayesian averaging, the task is to compute its posterior expectation

$$\mathrm{E}[f(A) \mid D] = \sum_{A} f(A)\, p(A \mid D)\,. \tag{2.2}$$

For convenience we may write just f instead $f(A)$ so that the above expectation becomes simply $\mathrm{E}[f \mid D]$. From now, the task of computing $\mathrm{E}[f \mid D]$ with respect to p is called the *feature expectation problem* (the FE problem). As a special case, if f is a binary valued indicator function for the presence of some structure property, this expectation equals the posterior probability of that property. A common example is the so called *arc feature*, which for a give pair of nodes u and w indicates whether an arc from u to w is present in the structure or not. Like a scoring function, a feature is said to be *modular*, if it decomposes in to a product $\prod_{v \in N} f_v(A_v)$. For example, the arc feature for an arc (u, w) is modular and can be defined by setting $f_v(A_v) = 0$ if $w = v$ and $u \notin A_v$ and $f_v(A_v) = 1$ otherwise.

[1] It is common to define the score in a logarithm form $\log p(D \mid A) + \log p(A)$ instead. This is often handy when searching for the best structure, but would turn out inconvenient later when we want to do Bayesian averaging.

Sometimes we can also be interested in the normalizing constant $p(D)$ that tells how well the data fits to the model (and thus the prior assumptions we have made). In addition, as we will see in the next section, computing the normalizing constant is often part of solving the feature expectation problem. Finding the normalizing constant reduces to the computation of the following summation:

$$p(D) = \sum_A p(D \,|\, A)\, p(A)\,.$$

Let us call this the *normalizing constant problem* (the NC problem). Both the normalizing constant problem and the feature expectation problem involve a summation over all possible structures. From now, these problems are collectively referred as the *summation problem(s)*, in contrast to the problem of finding the highest scoring structure, which is referred as the *optimization problem*.

This far, not much have been said about how the structure prior and the parameter prior are or should be defined. Let us first consider the parameter prior $p(\theta \,|\, A)$. When defining the parameter prior, the LCPDs are typically assumed to be taken from a parametrized family of probability distribution, for instance a discrete or a Gaussian distribution [16, 32, 24]. In what follows, we identify the LCPDs with their free parameters. For each LCPD θ_v we denote by $\Theta_{A,v}$ the family it is taken from and correspondingly by $\Theta_A = \Theta_{A,1} \times \Theta_{A,2} \times \cdots \times \Theta_{A,n}$ the family of full parameter vector θ. While the algorithms and results that we present do not rely on any specific family of distributions, in the experiments and examples only discrete variables are considered.

As we saw above, the Bayesian score consists of the structure prior and the likelihood. Using the notation above, the likelihood is obtained by integrating over the free parameters of the LCPDs as follows:

$$p(D \,|\, A) = \int_{\Theta_A} p(D \,|\, \theta, A)\, p(\theta \,|\, A)\, \mathrm{d}\theta\,.$$

While computing this integral in general form is usually computationally infeasible, with certain assumptions about the parameter prior the computation can be made practical. In addition, since most structure learning algorithms also require that the score is modular, the parameter prior needs to satisfy some additional requirements, discussed next.

We require that the parameter prior satisfies the following two properties, as defined by Heckerman et al. [32]: *global parameter independence*, which means that given the structure the parameters of different nodes are

independent, that is $p(\theta \mid A) = \prod_{v \in N} p(\theta_v \mid A)$, and *parameter modularity*, which says that if A and A' are structures such that $A_v = A'_v$ for some v then $p(\theta_v \mid A) = p(\theta_v \mid A')$. Together these imply that $p(\theta \mid A) = \prod_{v \in N} p(\theta_v \mid A_v)$ and it is possible to show that as a consequence, the likelihood decomposes into a product

$$p(D \mid A) = \prod_{v \in N} p(D_v \mid D_{A_v}, A_v) \,,$$

where

$$p(D_v \mid D_{A_v}, A_v) = \int_{\Theta_{A,v}} p(\theta_v \mid A_v)\, p(D_v \mid D_{A_v} \theta_v, A_v)\, \mathrm{d}\theta_v$$

and $\Theta_{A,v}$ depends only on v and A_v. In certain cases the above integral can be computed in closed form. For discrete data this is the case if each $\theta_v(\,\cdot\,; x_{A_v})$ defines a discrete distribution, whose parameters $(\theta_v(x_v; x_{A_v}) : x_v \in \mathcal{X}_v)$ are Dirichlet distributed independently for each $v \in N$ and $x_{A_v} \in \mathcal{X}_{A_v}$. With certain choices of the (hyper)parameters of the Dirichlet distributions this leads to BDeu and K2 scores [32, 16].

Consider then the structure prior $p(A)$. The above assumptions about the parameter prior made the likelihood both decomposable (modular) and easy to evaluate. To transfer these properties to the Bayesian score, the final step is to require similar properties from the structure prior: We say that the structure prior is *modular* if $p(A) = \prod_v \rho_v(A_v)$ for some nonnegative functions $\rho_1, \ldots, \rho_n$. For example, by setting $\rho_v(A_v)$ to a constant that is independent of v and A_v, a uniform prior over structures is obtained. It is easy to see that, if the structure prior, the parameter prior and the likelihood are all modular, also the resulting Bayesian score and the corresponding posterior probability are modular. As we will see in the next section, this allows the optimization problem to take a nice recurrence form, which in turn can be solved using dynamic programming.

It also turns out, that it is possible to use an analogous approach to solve the summation problems but this requires a slightly different type of prior. Remember, that every structure corresponds to one or more node orderings. Formally, we can encode any (linear) order on the nodes in N as a relation $L \subseteq N \times N$, where $(u, v) \in L$ if u precedes v or $u = v$. The set of all predecessors of v in L is denoted by L_v. We say, that a structure A and a linear order L are compatible, if $A \subseteq L$ (or equivalently $A_v \subseteq L_v$ for all $v \in N$), that is, if L is a topological ordering of A. We can now define a joint prior of A and L that decomposes as $p(A, L) = \prod_v \rho_v(A_v)\pi_v(L_v)$ if $A \subseteq L$ and $p(A, L) = 0$ otherwise, for some nonnegative functions $\rho_1, \ldots, \rho_n$ and $\pi_1, \ldots, \pi_n$. D and θ are assumed to be independent of L given A.

A structure prior is said to be *order-modular* if it can be expressed as a marginal $p(A) = \sum_L p(A, L)$ of such a decomposable joint prior. Together with a modular parameter prior and a modular likelihood, an order-modular structure prior leads to an order-modular posterior.

While a modular (or order-modular) score s definitely makes many structure learning problems more manageable, the local scores s_v need still each to be defined for 2^{n-1} potential parent sets. To make the number of potential parent sets smaller, an upper bound k is often defined for the size of the parent set. In the Bayesian setting, this upper bound can simply be incorporated to be a part of the structure prior by setting $\rho_v(A_v) = 0$ for all $v \in N$ and $|A_v| > k$.

2.3 Exact algorithms for structure learning

This section shows how the score-based optimization problem and the summation problems can be solved exactly. This information will be relevant in Chapters 3 and 4, which present randomized estimation algorithms for the summation problems. Constraint-based algorithms are not relevant to the next two chapters, and are thus not handled here. They will be briefly returned to in Chapter 5.

Both the optimization and the summation problems can be trivially solved by exhaustively enumerating all possible structures. Since there is a superexponential number of structures, this brute force approach is feasible only for a small number of nodes (fewer than 10). The next subsection describes a dynamic programming algorithm that solves these problems in $O(kn2^n + n^{k+1})$ time and $O(n2^n)$ space, where k is the maximum number of parents allowed for a single node. This allows solving instances of up to 20–30 nodes.

2.3.1 Dynamic programming over node subsets

Consider first the feature expectation problem. Koivisto and Sood [39] described a dynamic programming formulation that can be used to solve the problem as follows. For simplicity, assume that the feature f is binary. In this case, the expectation from Equation 2.2 can be written as a ratio $\mathrm{E}[f \,|\, D] = p(f, D)/p(D)$, where $p(f, D)$ is a shorthand for $p(f(A) = 1, D)$, and therefore

$$p(f, D) = \sum_A f(A) p(D \,|\, A)\, p(A)\,. \tag{2.3}$$

Thus, the problem is split into two subproblems, namely, solving the unnormalized feature probability $p(f, D)$ and solving the normalizing constant

$p(D)$. Moreover, these problems are very similar in nature: A method for computing $p(f, D)$ can also be used to compute the normalizing constant since $p(D) = p(\mathit{1}, D)$, where $\mathit{1}$ is the constant feature defined by $\mathit{1}(A) = 1$ for all structures A. Thus, it remains to find a way to compute $p(f, D)$ for a (reasonably) arbitrary f. It is easy to see, that what was stated above, also applies naturally to non-binary features, if the notation $p(f, D)$ is defined in general according to Equation 2.3.

After the reduction above, the remaining problem is to compute unnormalized feature probabilities. The key to an efficient solution is to assume an order-modular prior $p(A, L)$ over the structure A and a linear order L as described in Section 2.2.3. Then, by switching the order of summations over A and L, and using the assumptions from the previous section, the unnormalized feature probability can be molded into a sum–product

$$p(f, D) = \sum_{L} \prod_{v \in N} \alpha_v(L_v)\,, \tag{2.4}$$

where the summation is over all linear orders and the terms of the inner product are given by

$$\alpha_v(L_v) = \pi_v(L_v) \sum_{A_v \subseteq L_v} f_v(A_v)\, \rho_v(A_v)\, p(D_v \,|\, D_{A_v}, A_v)\,. \tag{2.5}$$

Assume first that the terms $\alpha_v(L_v)$ have already been computed. The sum–product over linear orders in Equation 2.4 allows a dynamic programming solution that is similar to the classic solution to the traveling salesman problem by Bellman [7]. Namely, with some further manipulation the sum–product can be reformulated as a recurrence

$$F(S) = \sum_{v \in S} \alpha_v(S \setminus \{v\})\, F(S \setminus \{v\})\,, \qquad \text{for } \emptyset \subset S \subseteq N\,, \tag{2.6}$$

and $F(\emptyset) = 1$, so that $p(f, D) = F(N)$. If the values $\alpha_v(L_v)$ are given (accessible in constant time) for all $v \in N$ and $L_v \subseteq N \setminus \{v\}$, then by using dynamic programming over the subsets S the values $F(S)$ for all $S \subseteq N$ can be computed in $O(n2^n)$ time and $O(2^n)$ space.

The remaining problem is to precompute the terms $\alpha_v(L_v)$. Unfortunately, the straightforward application of Equation 2.5 would lead to $\Omega(3^n)$ total time requirement. However, it turns out that for a fixed $v \in N$ the values $\alpha_v(L_v)$ for all $L_v \subseteq N \setminus \{v\}$ can be computed in $O(n2^n)$ total time and $O(2^n)$ space, by using the so-called *fast zeta transform* (see for example [37]). If the number of parents per node is limited to be at most k, then it is possible to further reduce the total computation time per node to

$O(k2^n + n^k)$ [39]. Therefore, computing the terms $\alpha_v(L_v)$ for all n nodes and then computing the unnormalized feature probability by dynamic programming results in total time requirement $O(kn2^n + n^{k+1})$.

Consider then the optimization problem. Unfortunately, with an order-modular prior, the same approach does not work for optimization. But if a modular prior is assumed instead of an order-modular, then the described dynamic programming algorithm can be relatively straightforwardly turned into an optimization algorithm that computes the score of the highest scoring structure [39, 55, 64, 65]. Basically all the summations in the computations just need to be replaced by maximizations. The actual structure that produces the highest score can be constructed by backtracking the results of the dynamic programming and the fast zeta transform.

2.3.2 Other exact algorithms

After the introduction of the above described algorithms there have been some advances in both the summation and optimization problems. Tian and He [70] presented a dynamic programming algorithm that solves the feature expectation problem with modular prior in $O(3^n)$ time and $O(n2^n)$ space. The idea is to use the inclusion–exclusion principle to reduce the problem recursively into smaller subproblems where only a subset of nodes is allowed to have incoming arcs. Parviainen and Koivisto [58] presented a way to compute feature expectations in smaller space with the trade-off of larger time consumption by partitioning the set of all structures into smaller sets represented as partial orders and solving the problem for each partial order separately. Still, for the FE problem, the described $O(kn2^n)$ (for order-modular prior) and $O(3^n)$ (for modular prior) dynamic programming algorithms remain the fastest known exact methods. With current hardware these algorithms scale up to about 25 and 20 nodes respectively.

The optimization problem, on the other hand, has seen a bit more development. Yuan et al. [78, 77] showed that reformulating the optimization problem as a shortest path finding problem and then applying the A* search algorithm with a properly chosen heuristic can for many datasets reduce both the time and space usage compared to dynamic programming. This approach is still closely related to dynamic programming. Another more different strategy that has turned out to be quite successful is to use integer linear programming (ILP) to solve the optimization problem [34, 18, 4]. An advantage offered by this approach is that the reformulation as an ILP problem allow the usage of highly sophisticated ILP solvers. The drawback of this approach is that it does not give any useful worst-case bounds for the running time.

Chapter 3

Monte Carlo estimation

Consider the problem of computing the posterior expectation of a structural feature (the FE problem). If the number of nodes in the structure exceeds about 20 in the case of a modular structure prior or 25 in the case of an order-modular structure prior, the current methods cannot solve the problem exactly (see Section 2.3). In this chapter we will see that in many such cases the solution can be reasonably well estimated by using Monte Carlo methods, which are based on random sampling. This chapter is based on Papers I and II.

3.1 Estimation via sampling structures

Monte Carlo is a technique for numerical integration, and can be used for estimating the expectations of random variables (see for example [62, 56]). Consider the FE problem and let D be the dataset and f be a feature for which we want to estimate the posterior expectation (Equation 2.2). A straightforward Monte Carlo algorithm would draw T independent random samples $A^{(1)}, \ldots, A^{(T)}$ from the posterior distribution over structures and estimate the expectation as follows:

$$\mathrm{E}[f \,|\, D] \approx \frac{1}{T} \sum_{t=1}^{T} f(A^{(t)}) \,. \tag{3.1}$$

From now, let us call this the simple Monte Carlo estimate.

In order to collect enough samples to get an accurate estimate, drawing samples from $p(A \,|\, D)$ should be relatively fast. Unfortunately, if f is assumed to be modular, sampling structures from the posterior distribution seems to be at least as hard as evaluating the expectation exactly. For non-modular features, though, even slow sampling can be useful. Indeed, using

a modification of the dynamic programming algorithm from Section 2.3.1 it is possible to draw samples from an order-modular distribution using only an additional polynomial time per sample [30]. Still, if n about 30 or larger, something else is needed.

In general, if exact sampling from the target distribution is not feasible, there are a couple common approaches (again, see for example [62, 56]). One option is to use *importance sampling*. In importance sampling the samples are drawn from another easier distribution and the terms of the sum in Equation 3.1 are weighted accordingly to correct the caused bias. For good estimates, the sampling distribution should be close to the target distribution. Unfortunately, in our case it seems difficult to find an easy exact sampling distribution that is close enough to the posterior to provide good estimates. Another option is to revert to approximate sampling, which usually means using the *Markov chain Monte Carlo* method or some variation of it. In the following sections we apply Markov chain Monte Carlo and some of its variants to the FE problem.

3.1.1 Markov chain Monte Carlo

The Markov chain Monte Carlo (MCMC) method [48, 29] is a standard solution for the problem of estimating an expectation when sampling exactly from the target distribution is hard. Let S be a random variable which takes values from a sample space $\mathcal{S}$ and let g be a function from $\mathcal{S}$ to real numbers. Consider the problem of estimating $\mathrm{E}[g(S)]$ with respect to a probability distribution p. The MCMC method is based on simulating a Markov chain that has $\mathcal{S}$ as its state space and p as its stationary distribution. To draw a sequence of samples $S^{(1)}, \ldots, S^{(T)}$, an arbitrary starting state is selected from $\mathcal{S}$, the Markov chain is simulated a number of steps and at predefined intervals the current state of the chain is picked as a sample. The expectation is then estimated using the simple Monte Carlo estimate as in Equation 3.1.

MCMC relies on the assumption that the collected samples follow a distribution that is reasonably close to $p(S)$. It can be shown, that if the Markov chain is irreducible and aperiodic, and the number of simulation steps between the samples is increased, then the distribution of each sample approaches the stationary distribution. Usually a long *burn-in* simulation is run before the first sample to give the chain some time to move from a starting state with possibly very low probability to the region of states with high probability. Unfortunately, it is often difficult to tell how long one should simulate the chain to get a sample distribution that is a reasonably good approximation of $p(S)$. A related issue is that consecutive samples are

not independent, which means that the effective sample size is in practice smaller than T. As a consequence, usually it is not possible to obtain any meaningful guarantees about the estimate of the expectation. Despite these theoretical shortcomings, in practice the MCMC method has proven to work well in many real word scenarios.

There are several ways to build a Markov chain that has a desired stationary distribution. A common one, and the one that is used throughout this thesis, is the *Metropolis–Hastings* algorithm [48, 29] which works as follows: On each step, first a transition from the current state S to a (possible) next state S' is proposed according to a special proposal distribution $q(S' \,|\, S)$. Typically, the proposal probability is defined so that it is high only for states S' that are close to S with respect to some natural distance function. The proposal is accepted with probability

$$\min \left\{ 1, \frac{p(S')\, q(S \,|\, S')}{p(S)\, q(S' \,|\, S)} \right\} ,$$

in which case the next state is set to S', and rejected otherwise, in which case the next state remains as S. Often the proposal distribution is symmetric, that is, $q(S' \,|\, S) = q(S \,|\, S')$, and the acceptance probability thus reduces to $\min\{1, p(S')/p(S)\}$.

In many cases, even evaluating the probability $p(S)$ of an arbitrary state S is computationally expensive. In these cases is often useful to note, that these probabilities only appear in the ratio $p(S')/p(S)$. Thus, it suffices to be able to compute $p(S)$ up to an unknown constant. Another potentially useful remark is that there is often only a small difference between consecutive states in the chain. Sometimes this can be exploited by partially reusing the results of the computations that were done for previous states.

3.1.2 Structure MCMC

Several MCMC algorithms that estimate the solution of the FE problem by sampling DAG structures have been suggested [46, 27, 26, 28]. In these algorithms, the state space of the Markov chain consists of all possible structures on the node set N and the transition probabilities have been chosen so that the stationary distribution of the chain is the structure posterior $p(A \,|\, D)$. To achieve this, Madigan and York [46] used the Metropolis–Hastings algorithm with a proposal distribution $q(A' \,|\, A)$ that is uniform over such structures A' that can be obtained from structure A by addition, removal or reversal of a single arc. From now on, we call this algorithm *Structure MCMC*.

The main shortcoming of Structure MCMC is that it is often slow in convergence and mixing. There have been several attempts to overcome this shortcoming: Grzegorczyk and Husmeier [28] suggested an alternative type of edge reversal in order to improve mixing of the Markov chain. Eaton and Murphy [20] introduced a version that first computes biased posterior probabilities of the arcs with the dynamic programming algorithm and then uses these the biased arc probabilities to build an alternative proposal distribution for the structures.

In spite of the mentioned improvements to Structure MCMC, a more successful approach seems to have been also a more fundamental one: to change the actual sampling space from structures to something else. This will be investigated further in the next sections.

3.2 Alternative sampling spaces

A sampling space that consists of structures is only one of many possibilities, and some other sampling spaces can sometimes lead to significantly more accurate estimates. There are a couple of reasons for this. First, the posterior of structures if often spiky, which leads to an estimate that has a large variance. If an alternative sample space has a smoother posterior distribution, then the variance of the estimate can be significantly reduced. Second, perhaps more importantly, an alternative sample space may allow an easy way to construct a Markov chain that mixes better, and thus produces better samples while taking fewer simulation steps. These two reasons are also partially linked, as a smoother posterior generally improves mixing.

The general idea is to augment the model with a new random variable, denoted here by S, resulting in a joint distribution $p(S, A, \theta, D)$. Using S as a sample variable leads to a Monte Carlo algorithm that draws T random samples $S^{(1)}, \ldots, S^{(T)}$ from the posterior distribution $p(S \,|\, D)$ and estimates

$$\mathrm{E}[f \,|\, D] \approx \frac{1}{T} \sum_{t=1}^{T} \mathrm{E}[f(A^{(t)}) \,|\, S^{(t)}, D] \,. \tag{3.2}$$

This technique is sometimes called *Rao–Blackwellization*, *conditioning* or *derandomization* [56].

What mostly restricts the choice of the sampling space, is the computational cost per sample. First of all, drawing samples from the posterior $p(S \,|\, D)$ should be relatively fast. In addition, for each drawn sample $S^{(t)}$, one needs to be able to efficiently compute the sample expectation $\mathrm{E}[f(A^{(t)}) \,|\, S^{(t)}, D]$. These two tasks are actually quite similar. First, as we

saw in the previous section, for the sampling phase of MCMC it is sufficient that the unnormalized sample probability $p(S, D)$ can be efficiently computed for any S. Second, assuming that f is a binary feature, the sample expectation can be written as a ratio $p(f, S, D)/p(S, D)$, where $p(f, S, D)$ is a shorthand for $p(f(A) = 1, S, D)$. This is analogous to the decomposition used in Section 2.3.1. Finally, since $p(S, D) = p(\mathit{1}, S, D)$, where $\mathit{1}$ is a constant feature such that $\mathit{1}(A) = 1$ for all structures A, both tasks can be reduced to a more general task of computing unnormalized sample feature probabilities $p(f, S, D)$. Like in Section 2.3.1, we can also obtain a similar decomposition and reduction for non-binary features. In the next two sections we consider two types of sampling spaces for which this computation is fairly efficient and which also lead to more accurate estimates.

3.2.1 Linear orders

Friedman and Koller [23] introduced the *Order MCMC* algorithm that samples linear orders instead of structures. They demonstrated that this choice leads to significantly improved estimates when compared to Structure MCMC. A key observation behind the algorithm is that, if L is a linear order on the node set N and a joint distribution $p(L, A, \theta, D)$ is defined appropriately, then under some conditions, the joint probability $p(f, L, D)$ can be computed relatively efficiently. Another observation is that an order-modular structure prior $p(A) = \sum_L p(A, L)$ naturally leads to such an appropriate joint distribution $p(L, A, \theta, D)$. This, in turn, leads to a Monte Carlo algorithm that samples linear orders $L^{(1)}, \ldots, L^{(T)}$ from the posterior $p(L \,|\, D)$ and estimates the feature expectation as in Equation 3.2.

It remains to show how to compute the unnormalized sample feature probability $p(f, L, D)$ efficiently. Due to how the order-modular prior is defined, the computation reduces to a summation

$$p(f, L, D) = \sum_{A \subseteq L} f(A)\, p(A, L, D)\,, \tag{3.3}$$

where A goes over all structures that are compatible with L and $p(A, L, D) = p(D \,|\, A)p(A, L)$. The number of terms in the sum is much smaller than the superexponential number of terms in the full summation over all structures (Equation 2.3), but it is still fairly large. In Chapter 4 we will see how to make the computation of this sum practical.

In order to construct a Markov chain sampler for linear orders, Friedman and Koller considered a couple of alternative proposal distribution types and settled to a proposal distribution $q(L' \,|\, L)$ that is uniform over the

$n(n-1)/2$ orders L' that can be obtained from L by swapping the positions of two nodes in the ordering. For example, a node ordering $(4, 2, 5, 1, 7, 3, 6)$ can be obtained from $(4, 7, 5, 1, 2, 3, 6)$ by swapping the positions of nodes 2 and 7. While this move seems to work well, more sophisticated move types could potentially further improve mixing. Ellis and Wong [21], for instance, have proposed an extension which swaps the positions of more than two nodes in a "cylindrical shift", and thus allows larger jumps in the state space of the Markov chain.

3.2.2 Partial orders

Next we will see how Order MCMC can be further improved by grouping several orders together. The idea is to partition the space of linear orders into groups, each of which corresponds to a partial order. This idea was first introduced in the context of exact computation by Parviainen and Koivisto [57, 58]. Paper I applies the same principle and introduces Partial Order MCMC, an extended version of Order MCMC, that samples partial orders instead of structures.

In order to outline how Partial Order MCMC works, we need some additional notation. A relation $P \subseteq N \times N$ is a *partial order* on N if it is reflexive ($(v, v) \in P$ for all $v \in N$), antisymmetric ($(u, v) \in P$ and $(v, u) \in P$ only if $u = v$), and transitive (if $(u, v) \in P$ and $(v, w) \in P$ then $(u, w) \in P$) [63]. If P is a partial order on N and L is a linear order on N such that $P \subseteq L$, then P *contains* L and L is a *linear extensions* of P. In what follows, we consider families of partial orders that are *exact covers* of linear orders, meaning that each linear order is contained in exactly one partial order in the family. As a result, the partial orders in such a family partition linear orders into disjoint groups.

Let $\mathcal{P}$ be an exact cover of linear orders on N. We extend the joint distribution $p(L, A, \theta, D)$ from the previous subsection by adding another random variable, a partial order P that takes values from $\mathcal{P}$, always contains L, and given L, is independent of D, θ and A. Since P always contains L and $\mathcal{P}$ is an exact cover, P is uniquely determined by L. Partial Order MCMC samples partial orders $P^{(1)}, \ldots, P^{(T)} \in \mathcal{P}$ from the posterior $p(P \,|\, D)$ and estimates

$$\mathrm{E}[f \,|\, D] \approx \frac{1}{T} \sum_{t=1}^{T} \mathrm{E}[f(A^{(t)}) \,|\, P^{(t)}, D] \,.$$

Again, it is sufficient that the unnormalized sample feature probability $p(f, P, D)$ can be computed efficiently. This computation reduces to a

summation

$$p(f, P, D) = \sum_{L \supseteq P} p(f, L, D)\,, \tag{3.4}$$

where L goes over the linear extensions of P, and $p(f, L, D)$ is as given in Equation 3.3. In Chapter 4 we are going to see, that for certain types of partial order families this sum can be computed almost in the same time as the corresponding summation for linear orders (Equation 3.3).

Bucket orders (or *weak orders*) [22, 63] are partial orders, for which the above summation is often fairly efficient to evaluate. In a bucket order on N, the nodes in N are distributed into mutually disjoint sets $B_1, \ldots, B_h$, called *buckets*. The nodes that are in a same bucket are mutually incomparable. The ordering of two nodes in different buckets is determined by the ordering of the buckets so that for $i < j$, the nodes in B_i precede the nodes in B_j. If we fix the sizes of the buckets $|B_1| = b_1, \ldots, |B_h| = b_h$ and consider all possible ways to place the nodes in to the buckets, we obtain a partial order family that is an exact cover of linear orders, and thus suffices for our purposes.

Example 3.1. For example, for eight nodes $\{1, 2, 3, 4, 5, 6, 7, 8\}$, we could consider a bucket order family that has 3 buckets of sizes 3, 3 and 2, in this order. A bucket order $(\{1, 5, 6\}, \{3, 4, 8\}, \{2, 7\})$ belongs to this family. (See the leftmost diagram in Figure 3.1.)

Example 3.2. As a special case, if a bucket order family has only buckets of size 1, then the bucket orders in the family correspond to linear orders. In this case, the summation in Equation 3.4 contains only one term and thus reduces to the summation in Equation 3.3.

Example 3.3. On the other extreme, if a bucket order family has only one bucket that contains all the nodes in N, then there is only one bucket order in the family and it contains all linear orders on N. Therefore, the summation becomes equivalent to the full summation in Equation 2.4, for which an exponential algorithm was given in Section 2.3.1.

Partial Order MCMC, as presented in Paper I, uses a proposal distribution that is similar to the proposal distribution that was described above for linear orders: A pair of nodes from different buckets are chosen uniformly at random and proposed for swapping. For example, $(\{1, 5, 6\}, \{3, 4, 8\}, \{2, 7\})$ could be obtained from $(\{1, 2, 6\}, \{3, 4, 8\}, \{5, 7\})$ by swapping nodes 2 and 5. This, and another example swap, are illustrated in Figure 3.1.

The experiments in the Papers I and II demonstrate that sampling bucket orders instead of linear orders often significantly improves mixing.

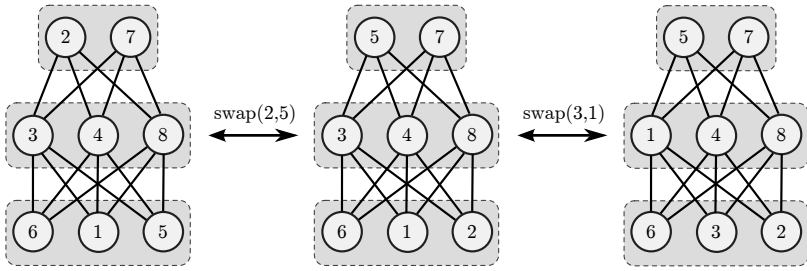


Figure 3.1: Three different bucket orders illustrated as Hasse diagrams. The buckets are denoted by gray squares. It is possible to move between the adjacent bucket orders by swapping the positions of two nodes.

In Figure 3.2 we can see an example of such a behavior: Most of the runs of Order MCMC seem to get stuck in the areas of low probability while the runs of Partial Order MCMC all seem to convergence. Bucket orders often also lead to more accurate estimates as shown in Figure 3.3a.

In addition to bucket orders, there are also some other types of partial orders that may be worth considering. Some of them, such as parallel pairwise orders, are more useful for exact structure learning [57, 58]. For Partial Order MCMC and related algorithms, however, bucket orders seem to be, if not the best, at least a good choice.

3.2.3 Nested sampling of structures

Order MCMC and Partial Order MCMC as described above require that the structure prior is order-modular. In addition, efficient computation of the unnormalized sample feature probabilities requires that the feature function is modular (see Chapter 4). Structure MCMC, on the other hand, allows arbitrary features and modular structure priors, but it is otherwise inferior due to bad mixing of Markov chains. However, it is possible to augment Order MCMC and Partial Order MCMC with a method that samples DAG structures from linear orders or partial orders. This method, which we call *nested sampling*[2], allows both arbitrary features and application of the so called *bias correction*. Bias correction makes it possible to cancel the bias that the "wrong" sampling distribution causes to the estimates. Friedman and Koller [23] originally suggested using nested sampling in Order MCMC for estimating probabilities of arbitrary features but did not consider correcting the bias. Papers II and III propose using nested

[2]Not to be confused with the nested sampling algorithm by Skilling [66] for Bayesian model comparison.

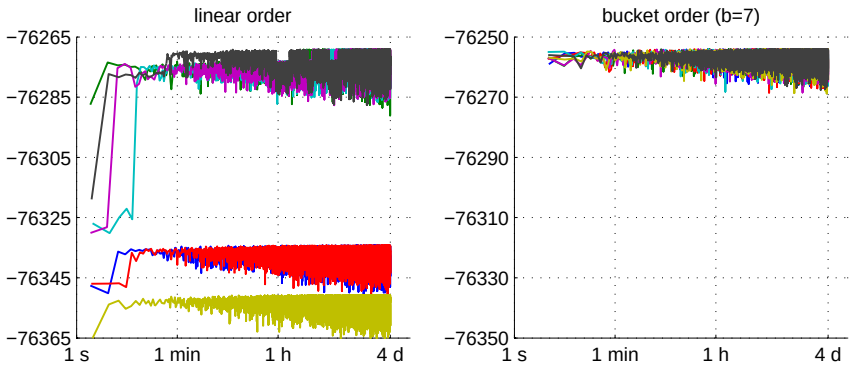
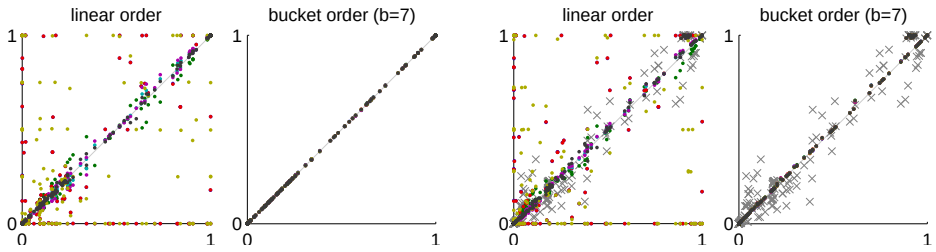


Figure 3.2: Convergence/mixing of seven independent runs of Order MCMC and Partial Order MCMC on the Mushroom dataset from the UCI repository [45], with an order-modular structure prior and BDeu scoring. Partial Order MCMC uses bucket order with a constant bucket size of 7 nodes (except the last bucket, which is smaller). Samples were generated at intervals of 1024 simulation steps of the Markov chain. The unnormalized log probability ($\log p(L, D)$ or $\log p(P, D)$) of the generated samples (Y-axis) has been plotted as a function of of the running time (X-axis). Since the chains have already been simulated 1023 steps before the first sample, some of the runs may appear to have been converged right at the beginning.

sampling for bias correction as described next.

Let us consider nested sampling in the case of partial orders. As linear orders are a special case of partial orders, everything that follows, also holds for linear orders. Let $P^{(1)}, \ldots, P^{(T)}$ be a sequence of partial orders obtained from an order-modular posterior, for example, by the Partial Order MCMC algorithm. In nested sampling, for each $t = 1, \ldots, T$, a structure $A^{(t)}$ is drawn from the posterior conditioned on $P^{(t)}$, that is, from $p(A \,|\, P^{(t)}, D)$. In practice, it is easiest to draw the structures in two phases: For each partial order $P^{(t)}$, first a linear order $L^{(t)}$ is drawn from $p(L \,|\, P^{(t)}, D)$ and then a structure $A^{(t)}$ is drawn from $p(A \,|\, L^{(t)}, D)$. In the special case of nested sampling for linear orders the first phase can obviously be omitted. While in the above description one structure is drawn per each partial order, in practice it is typically useful to draw multiple structures from each partial order, as a larger number of samples generally leads to better estimates.

Once the structure samples $A^{(1)}, \ldots, A^{(T)}$ have been generated, they are used to estimate the feature expectation. If bias correction is not needed, then we can use the simple Monte Carlo estimate from Equation 3.1. The



(a) Order-modular prior. (b) Modular prior (nested sampling).

Figure 3.3: The accuracy of the estimates by Order MCMC and Partial Order MCMC on the Mushroom dataset. For each run shown in the Figure 3.2, the first half of the samples was discarded as a burn-in, and remaining half was used to estimate the posterior probabilities of all arcs. Since Mushroom contains only 22 variables, the probabilities can also be computed exactly. The estimates obtained by Order MCMC and Partial Order MCMC (Y-axis) have been plotted against the exact probabilities (X-axis). In the case of a modular structure prior, the exact order-modular probabilities are also shown against the exact modular probabilities as gray symbols $\times$.

non-modularity of the feature function is no longer a problem, as long as the feature function remains reasonably easy to evaluate. If a modular prior is desired, the bias caused by the order-modular sampling distribution can be corrected in an importance sampling manner, as proposed in Paper II: Let p be the order-modular sampling distribution and q be the modular target distribution. The *self-normalized importance sampling estimate* (see for example [56]) for the feature expectation with respect to q is

$$\mathrm{E}_q[f \,|\, D] \approx \frac{\sum_{t=1}^{T} w^{(t)} f(A^{(t)})}{\sum_{t=1}^{T} w^{(t)}}\,, \tag{3.5}$$

where $w^{(t)} \propto q(A^{(t)})/p(A^{(t)})$ are *importance weights*. The computation of these weights is a hard problem, but it can be feasible if n is relatively small. This will be discussed in more detail in Section 4.3.

Ellis and Wong [21] proposed an alternative approach for correcting the bias of Order MCMC without requiring the potentially expensive computation of the importance weights. Basically, the idea is to collect a large set of unique high probability structures by running the Order MCMC algorithm with nested sampling, and weight the structures by their true unbiased posterior probability. In order to cover enough of the posterior sample space, for each unique linear order L, unique structures are sam-

pled until their total posterior mass is at least $(1 - \epsilon)$ when conditioned on L. Unfortunately, this approach also has problems: First, in some, perhaps not that pathological cases, the estimator can behave badly. (For examples, see Paper II). Second, the number of structures that are sampled from a single order can be prohibitively large, especially if the number of nodes is large and/or the number of data points is small. Moreover, in order to detect possible duplicates, all previously sampled unique structures need to be stored in memory, and thus space consumption can easily become a bottleneck. Third, Ellis and Wong describe the approach only for linear orders, and it is not clear how it should be extended for general partial orders.

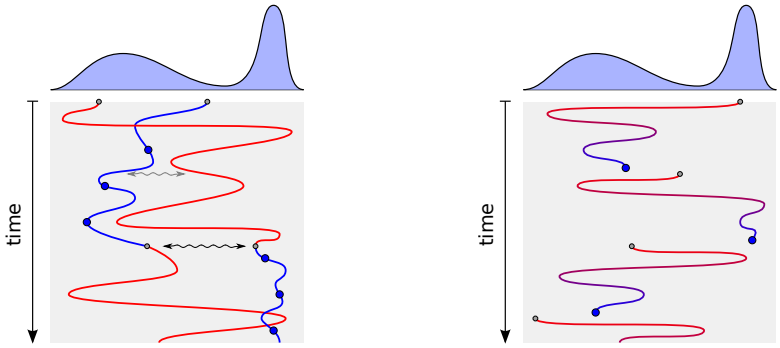
3.3 Advanced sampling methods

Using the alternative sampling spaces considered in previous section is not the only way to improve mixing of MCMC algorithms. In this section, we study two general modifications of MCMC that can further improve mixing. In addition, they provide solutions to the problem of estimating the normalizing constant $p(D)$. Both modifications are based on simulating multiple Markov chains with stationary distributions that are smoother than the target distribution.

3.3.1 Metropolis-coupled Markov chain Monte Carlo

Let $p(S)$ be a general target distribution that we want to generate samples from. In the *Metropolis-coupled Markov chain Monte Carlo* algorithm (or MC^3) [25] a sequence of gradually harder distributions $p_0, \ldots, p_r$ is constructed so that p_0 is some easy distribution that mixes quickly and $p_r = p$ is the actual target distribution. The distributions typically follow a tempering scheme $p_i(S) \propto p(S)^{a_i}$, where $0 = a_0 < a_1 < \cdots < a_r = 1$. It is sufficient that $p_i(S)$ can be computed only up to an unknown constant (which may be different for different i). Denote by $\tilde{p}_i$ the unnormalized version of p_i and let $Z_i = \tilde{p}_i / p_i$ be the corresponding normalizing constant.

For each $i = 0, \ldots, r$, a Markov chain that has p_i as a stationary distribution is constructed. In MC^3, these $r + 1$ chains are simulated independently in parallel. Let $S_0, \ldots, S_r$ be the current states of the chains. In addition to independent simulation, occasionally a swap of the states S_i and S_j of two randomly chosen adjacent chains i and $j = i + 1$ is proposed.



(a) MC^3. (b) AIS

Figure 3.4: An illustration of the behavior of MC^3 and AIS when sampling from a distribution with two modes. Empty circles are random starting states, the curves depict the movement of the Markov chains in the sample space and blue circles are points where a sample is generated from a chain. In this example, MC^3 simulates two chains, one with the target distribution as a stationary distribution (blue) and one with a tempered stationary distribution (red). Wavy arrows are time points when a swap is proposed between the two chains. AIS simulates multiple independent chains that all start with a tempered stationary distribution and gradually cool down to the target distribution.

The swap is accepted with probability

$$\min\left\{1, \frac{\tilde{p}_i(S_j)\,\tilde{p}_j(S_i)}{\tilde{p}_i(S_i)\,\tilde{p}_j(S_j)}\right\}.$$

The idea is that the chains in higher temperatures mix better and the swaps between two chains allow also lower temperature chains to take advantage of this and make larger jumps to other areas of the probability space. For an illustration, see Figure 3.4a. It can be shown that the swaps between chains do not change the stationary distribution of the chains. Therefore, samples from chain S_r can be used in estimation, just like in the basic MCMC algorithm.

MC^3 can be also used to estimate the normalizing constant Z_r of the target distribution. In order to estimate Z_r, T samples $S_i^{(1)}, \ldots, S_i^{(T)}$ are collected from each chain $i = 0, \ldots, r-1$. The ratio Z_r/Z_0, is then estimated by a telescoping product of

$$\frac{Z_i}{Z_{i-1}} \approx \frac{1}{T}\sum_{t=1}^{T} \frac{\tilde{p}_i\big(S_{i-1}^{(t)}\big)}{\tilde{p}_{i-1}\big(S_{i-1}^{(t)}\big)}, \qquad i = 1, \ldots, r\,.$$

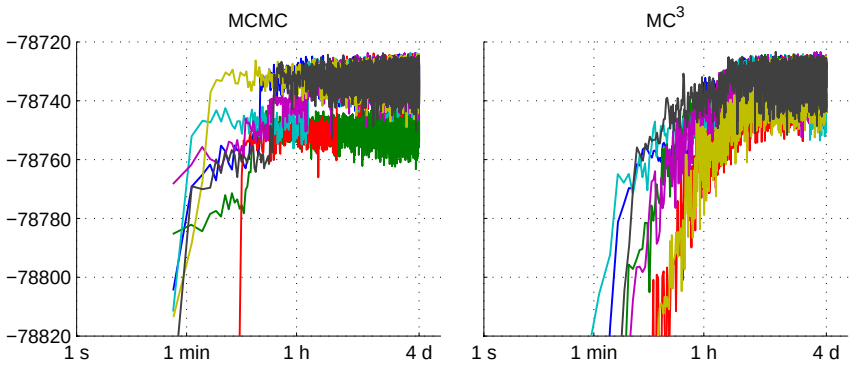


Figure 3.5: Convergence/mixing of seven independent runs of Partial Order MCMC and Partial Order MC^3 with $r = 15$ on the Spambase dataset from the UCI repository [45], with an order-modular structure prior and BDeu scoring. A constant bucket size of 9 nodes (except the last bucket) was used. Samples were generated at intervals of 1024 simulation steps of the Markov chain.

If p_0 is easy enough that Z_0 can be computed, then an estimate of Z_r is obtained by multiplying by Z_0.

In Papers II and III, MC^3 is used to sample linear and partial orders in the context of Bayesian network structure learning. In Figure 3.5 we see, that if the problem instance is difficult, then MC^3 can further improve mixing, even if bucket orders are used as a sampling space.

There are other methods that are similar to or inspired by MC^3, and that can also used for structure learning in Bayesian networks. Most notably, Ellis and Wong [21] proposed a modified version of Order MCMC, that uses a related equi-energy sampler [43]. Corander et al. [17], on the other hand, suggested another type of parallel structure based MCMC algorithm.

3.3.2 Annealed importance sampling

Annealed importance sampling (or AIS) [49] is another general method that improves the convergence compared to the basic MCMC method. Let $\mathrm{E}[g(S)]$ the expectation of a function g that we want to estimate with respect to distribution $p(S)$. Like in MC^3, a sequence of gradually changing distributions $p_0, \ldots, p_r$, such that $p_r = p$, is constructed. In the case of AIS, however, it is required that exact sampling is possible from p_0. A common choice is to set p_0 to be a uniform distribution. Compared to MC^3, in AIS r is usually much larger. Again, it suffices that $p_i(S)$ can be computed

only up to an unknown constant. As in the previous subsection, let $\tilde{p}_i(S)$ and Z_i be the corresponding unnormalized probability and the unknown normalizing constant respectively.

In AIS, the idea is to simulate Markov chains whose target distribution gradually changes during the simulation. For each $i = 1, \ldots, r-1$, let τ_i be a transition kernel of a Markov chain, whose stationary distribution is p_i. First a sequence of samples $S^{(1)}, \ldots, S^{(T)}$ and weights $w^{(1)}, \ldots, w^{(T)}$ is produced. Each sample $S^{(t)}$ and the corresponding weight $w^{(t)}$ are generated independently from other samples and weights as follows:

Generate S_0 from p_0.
Generate S_1 from S_0 using τ_1.
$\vdots$
Generate S_{r-1} from S_{r-1} using τ_{r-1}.

After that, we set $S^{(t)} = S_{r-1}$ and

$$w^{(t)} = \frac{\tilde{p}_1(S_0)}{\tilde{p}_0(S_0)} \frac{\tilde{p}_2(S_1)}{\tilde{p}_1(S_1)} \cdots \frac{\tilde{p}_r(S_{r-1})}{\tilde{p}_{r-1}(S_{r-1})} .$$

Once all the samples and weight have been generated, the expectation is then estimated by the self-normalizing importance sampling estimate

$$\mathrm{E}[g(S)] \approx \frac{\sum_{t=1}^{T} w^{(t)} g(S^{(t)})}{\sum_{t=1}^{T} w^{(t)}}.$$

If Z_0 can be computed, then it is also possible to get an estimate for the normalizing constant Z_r by estimating

$$\frac{Z_r}{Z_0} \approx \frac{1}{T} \sum_{t=1}^{T} w^{(t)} .$$

In addition to improving convergence, AIS also has some other useful properties that MCMC and MC^3 do not have: First, since the samples are generated independently, the sampling process is trivially parallelizable. Second, since each weight is an unbiased estimator for the ratio of normalizing constants, it is possible to obtain various high confidence lower bounds for the normalizing constant .

Papers II and III compare AIS to MC^3 in the context of structure learning in Bayesian network. It was found, that AIS is competitive with MC^3 in solving the FE problem, although when given the same amount of computational resources, MC^3 usually gives better results. However, as

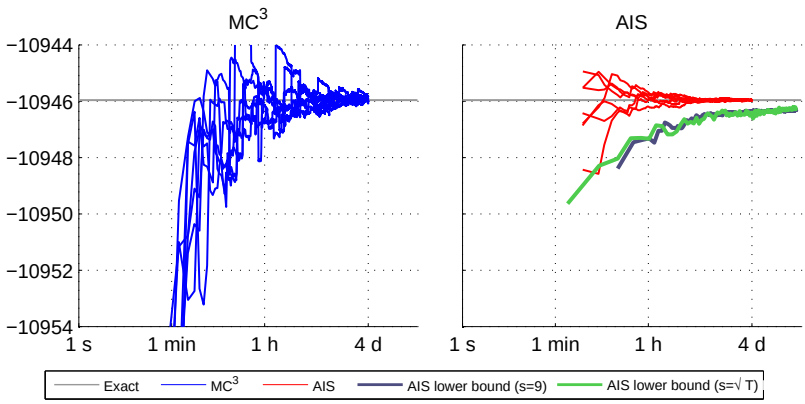


Figure 3.6: The log normalizing constants as estimated by the seven independent runs of MC^3 ($r = 15$) and AIS ($r = nm$), as a function of running time. The experiment was conducted on the Mushroom dataset, with an order-modular structure prior and BDeu scoring. For AIS, two different 0.95-confidence lower bounds have also been plotted. The exact value is also shown for comparison.

mentioned, AIS can be parallelized trivially. In solving the NC problem, AIS seems to be often significantly better than MC^3. This property is illustrated in Figure 3.6. Previously also Battle et al. [5] have applied AIS to structure learning. However, unlike Papers II and III, which sample linear orders and partial orders, they sample network structures.

Chapter 4

Per-sample computations

Chapter 3 described several Monte Carlo methods for the FE and NC problems, all of which rely on efficient per-sample computations. More specifically, given a sample S (a linear order or a partial order) and a modular feature f, we need to be able to efficiently compute the following two quantities:

- For sampling, we need to be able to compute $p(S \,|\, D)$ up to some normalizing constant.
- For computing the actual estimate for the expectation of f, we also need a way to evaluate the sample expectation $\mathrm{E}[f \,|\, S, D]$.

As we saw in Chapter 3, if the feature is binary, these both tasks can be reduced to a more general problem of computing the unnormalized sample feature probability $p(f, S, D)$. In this chapter, we see how to compute this quantity efficiently for linear orders and partial orders in the case of an order-modular structure prior. Everything that will be presented, also generalizes directly to modular non-binary features.

Nested sampling, described in Section 3.2.3, allows modular structure priors and non-modular features, but has also some additional requirements: In all cases , we need an efficient way to sample structures from the posterior conditioned on a linear order. For algorithms that are based on sampling partial orders, we also need an efficient way to sample linear orders from the posterior conditioned on a partial order. And finally, in the case of a modular structure prior, we need to compute the importance weights that are required in the bias-corrected estimate. Solutions to these problems are also described in this chapter.

This chapter is based on Papers I, II and IV.

4.1 Per linear order

In this section, we consider the problem of computing the unnormalized sample feature probability $p(f, L, D)$ and the problem of sampling structures from $p(A \mid L, D)$, for an arbitrary linear order L.

Consider first the problem of computing $p(f, L, D)$, as given in Equation 3.3. Assume, that the structure prior is order-modular, and that the feature f is modular. Then, by the modularity of the likelihood, the unnormalized sample feature probability decomposes into a product

$$p(f, L, D) = \prod_{v \in N} \alpha_v(L_v)\,, \tag{4.1}$$

where $\alpha_v(L_v)$ is defined as in Equation 2.5, repeated here for convenience:

$$\alpha_v(L_v) = \pi_v(L_v) \sum_{A_v \subseteq L_v} f_v(A_v)\, \rho_v(A_v)\, p(D_v \mid D_{A_v}, A_v)\,. \tag{4.2}$$

This decomposition was mentioned by Buntine [12] as well as Cooper and Herskovits [16], and later exploited by Friedman and Koller [23] in Order MCMC.

The inner sum in the decomposition has $2^{|L_v|}$ terms, which would still lead to an exponential computation time. However, it is sufficient to sum only over parent sets A_v that have a nonzero prior probability. Hence, the computation can be made manageable by, for example, bounding the number of the parents of v by a small constant k. As a result, the inner sum can be computed in $O(|L_v|^k)$ time, which leads to $O(n^{k+1})$ time requirement for the full product–sum.

4.1.1 Approximate summing

Despite limiting the sizes of parent sets, computing the inner sums in Equation 4.1 is by far the most time consuming part in Order MCMC. Often we would also like to set a parent set size bound that is not too restrictive, but a large bound leads to slow computations. As we will see next, it is possible to speed up the computations significantly by using approximations. Before describing those approximations in more detail, we express the summing problem in a more general form.

Let N be a ground set of n elements and $\mathcal{C}$ a collection of subsets of N. Let each subset $Y \subseteq N$ be associated with a weight $w(Y) > 0$ if $Y \in \mathcal{C}$ and $w(Y) = 0$ otherwise. In a *subset counting query* on a query set Q, we are

asked to compute a sum

$$W(Q) = \sum_{Y \subseteq Q} w(Y)\,. \tag{4.3}$$

The inner sums in Equation 4.1 correspond to queries where $Q = L_v$ and $w(A_v) = f_v(A_v)\rho_v(A_v)p(D_v \mid D_{A_v}, A_v)$. In general, if the parent sets are not restricted, then $\mathcal{C}$ consist of all subsets of $N \setminus \{v\}$. If the sizes of parent sets are bounded by a constant k, then $\mathcal{C} = \{A_v \subseteq N \setminus \{v\} : |A_v| \leq k\}$.

To answer a counting query exactly, it is sufficient to traverse those sets $Y \subseteq Q$ that are also in the collection $\mathcal{C}$ and to sum up their weights. For now, we call such sets *relevant*. Next we consider approximation algorithms that, for a given positive tolerance d, can answer such queries to within guaranteed relative error d, that is, so that the returned answer is between $(1 - d)W(Q)$ and $(1 + d)W(Q)$. For example, setting $d = 0.01$ should guarantee that the answer is within one percent from the correct value. The general idea is that, if some relevant sets are light compared to others, those can be omitted from the summation and the result will still be relatively accurate. More formally, we consider so-called *collector* algorithms, that visits some subsets of the ground set (possibly also irrelevant subsets), and sum up the weights of visited relevant sets. An ideal collector algorithm would visit the minimum number of the heaviest relevant sets whose total weight is at least $(1 - d)W(Q)$. Unfortunately, we do not know how to implement such an ideal algorithm efficiently. Paper II considers two less ideal approaches, which we will outline next.

For a high-performance implementation of Order MCMC, Friedman and Koller [23] proposed the following heuristic: As a preprocessing step, the sets in $\mathcal{C}$ are sorted in decreasing order by weight. In order to answer a counting query, the heuristic visits a fixed number of heaviest sets and returns the total weight of the visited relevant sets. This heuristic does not guarantee any approximation ratio. However, we can obtain such guarantees by a small change: We do not stop visiting the sets until the total weight of the remaining sets in $\mathcal{C}$ is small enough compared to the total weight of the visited relevant sets. To make the stopping decision efficient, the total weights of possible remaining sets can be precomputed. In addition, if too many sets are visited without termination, we can resorts back to exact computation. We call this modified algorithm *Sorted*.

While in many situations Sorted runs quickly, in some cases, especially if the query set is quite small, it visits a large number of irrelevant sets before terminating. Paper II proposes another algorithm, *Treedy*, that often works better in those situations. It is based on arranging the sets in $\mathcal{C}$ into a lexicographical tree structure, such that $\emptyset$ is the root and each nonempty

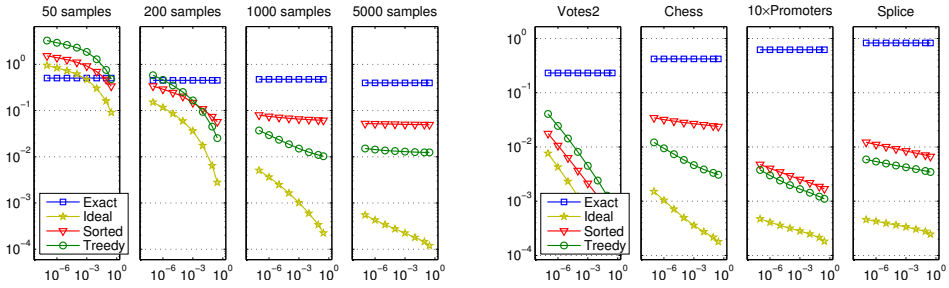
$Y \in \mathcal{C}$ is a son of $Y \setminus \{v\}$ where v is lexicographically the last element of Y. For each query, starting from the root, the tree is traversed greedily with respect to the total weights of the branches: On each step, the traversal visits a set that is the root of the heaviest unvisited branch of the tree. If the set is relevant, its weight is added to the accumulated total weight. If the set is irrelevant, then also all its descendants are irrelevant, meaning that the remaining branch can be skipped and ignored for the rest of the traversal. Due to "branch skipping", the algorithm can avoid traversing large portions of the tree, which makes it often more efficient than Sorted in cases where Sorted performs badly.

An efficient implementation of Treedy requires that the next node to visit can be found quickly. For this purpose, the algorithm maintains a priority queue that contains the unvisited child branches of every visited node. Unfortunately this adds some overhead to the algorithm. To keep the priority queue as small as possible, some additional preprocessing is conducted to turn the lexicographical tree actually into a binary tree. Further details are described in Paper II.

Figure 4.1 shows results from an experiment in which four different versions of Order MCMC were run on both artificial and real-world datasets. The purpose of the experiment was to compare Treedy and Sorted to exact summation and to a simulated performance of an ideal collector algorithm. The results show that often the runtime can be improved by one to two orders of magnitude by using Sorted or Treedy to approximate the inner summations in Equation 4.1. Treedy outperforms Sorted especially if the posterior is particularly spiky. This typically happens if the data size is large, as seen in Figure 4.1a where the number of data points (samples) is varied.

4.1.2 Sampling structures

In nested sampling, structures are drawn from the order-modular posterior conditioned on a linear order, that is, from $p(A \mid D, L)$. Drawing a structure amounts to drawing the parent sets for all nodes. Due to modularity assumptions and conditioning on an order, this can be done independently for each node: For $v \in N$, a parent set $A_v \subseteq L_v$ is drawn from $p(A_v \mid D, L_v) \propto \rho_v(A_v) p(D_v \mid D_{A_v}, A_v)$. As there are $O(|L_v|^k)$ potential parent sets for v, this requires $O(n^{k+1})$ time per linear order, the same that was required already for computing the unnormalized probability of L. Multiple structures can be drawn efficiently from a single linear order by applying the Alias method [76, 75]. This results in $O(n^{k+1})$ preprocessing time per order and an additional constant time per sampled parent set, thus $O(n)$



(a) Alarm (b) Lichman13 datasets

Figure 4.1: Runtime of Order MCMC. The number of seconds per MCMC step for an exact algorithm (Exact), a simulated ideal collector algorithm (Ideal), Sorted, and Treedy are shown as a function of approximation tolerance d for (a) datasets of different sizes sampled from the ALARM network [6], and (b) selected datasets from the UCI repository [45].

time per sampled structure.

If approximate summing is used to speed up the computation in the order sampling phase, then the structure sampling phase can turn out to be the bottleneck due to its per-order time requirement. In this case, it is possible to instead draw the structures from an approximate posterior distribution. By running a collector algorithm, such as Treedy or Sorted, with tolerance d for each L_v and drawing the parent sets from the visited relevant sets (potential parent sets), the resulting parent set distribution is within total variation distance d from the exact posterior.

4.2 Per partial order

In this section we tackle the problem of computing the unnormalized sample feature probability $p(f, P, D)$ and the problem of sampling structures from $p(A \mid P, D)$, for a partial order P. The following presentation will be superficial, leaving out some details. A more detailed description of the matter is given in Paper III.

Again, consider first the problem of computing $p(f, P, D)$, as given in Equation 3.4. If we combine it with Equation 4.1 from the previous section, we get a sum–product expression

$$p(f, P, D) = \sum_{L \supseteq P} \prod_{v \in N} \alpha_v(L_v)\,, \tag{4.4}$$

where $\alpha_v(L_v)$ is as defined in Equation 4.2 (and Equation 2.5). This ex-

pression closely resembles Equation 2.4, the only difference being that the summation is restricted to the linear extensions of P. Indeed, the unrestricted sum–product is just a special case of Equation 4.4, where P does not specify the order of any two nodes (see Example 3.3).

Parviainen and Koivisto [58] showed that a dynamic programming approach similar to the unrestricted case can be applied to partial orders. In fact, both the exact algorithm from Section 2.3.1 and the algorithm for linear orders from Section 4.1 can be seen as special cases of that algorithm. This generalized algorithm is outlined next.

4.2.1 Sum–product over orders

The key observation for efficient evaluation of the restricted sum–product (Equation 4.4) is that the dynamic programming only needs to be carried out over the *downsets* of P, that is, the subsets that can "start" a linear extension of P. More formally, a subset $S \subseteq N$ is a downset of a partial order P, if $v \in S$ and $(u, v) \in P$ together imply that $u \in S$. We denote the set of all downsets of P by $\mathcal{D}(P)$.

Given a partial order P, the recurrence 2.6 can be generalized into a recurrence

$$F(S) = \sum_{\substack{v \in S \\ S \setminus \{v\} \in \mathcal{D}(P)}} \alpha_v(S \setminus \{v\})\, F(S \setminus \{v\})\,, \qquad \text{for } S \in \mathcal{D}(P) \setminus \{\emptyset\}\,, \tag{4.5}$$

where $F(\emptyset) = 1$ and $p(f, P, D) = F(N)$. If the terms $\alpha_v(L_v)$ are given, then by dynamic programming over the donwsets of P, the values of the recurrence for all downsets can be computed in $O(n|\mathcal{D}(P)|)$ time and $O(|\mathcal{D}(P)|)$ space. (In fact, for bucket orders with maximum bucket size b, the time requirement reduces to $O(b|\mathcal{D}(P)|)$.) However, the total time requirement of computing $p(f, P, D)$ is dominated by the computation of the terms $\alpha_v(L_v)$, as seen next.

4.2.2 Sums over parent sets

The exact dynamic programming algorithm of Section 2.3.1 uses the fast zeta transform to compute $\alpha_v(L_v)$ for all $v \in N$ and $L_v \subseteq N \setminus \{v\}$. This requires $O(n^2 2^n)$ time and $O(n2^n)$ space. Such requirements are obviously unacceptable in the cases where n is relatively large and thus a Monte Carlo approach is needed. However, from Equation 4.5 it is easy to see, that for a given P, only values $\alpha_v(S)$ such that S and $S \cup \{v\}$ are downsets of P, are needed.

Fortunately, it is possible to solve such a restricted zeta transform problem in less time and space. To this end, the problem is split in two subproblems by rearranging the right hand side of the definition of α_v (see Equation 4.2) into a nested summation

$$\alpha_v(L_v) = \pi_v(L_v) \sum_{\substack{S \subseteq L_v \\ S \in \mathcal{D}(P)}} \sum_{A_v \in \mathcal{T}_S(P)} f_v(A_v)\, \rho_v(A_v)\, p(D_v \,|\, D_{A_v}, A_v)\,, \qquad (4.6)$$

where $\mathcal{T}_S(P)$ consists of those subsets of downset S that are not subsets of any smaller downset of P.

The first step is to compute and store the results of the inner summations for all downsets $S \in \mathcal{D}(P)$. As (by the definition of $\mathcal{T}_S(P)$) every A_v participates to only one such summation, this step can be completed in $O(|\mathcal{D}(P)| + n^k)$ time for a fixed v, if the iteration over potential parent sets A_v and the mapping from A_v to the corresponding downset S can be done in a constant time per parent set. This is the case for example for bucket orders.

The second step is to compute the outer summation for all needed sets L_v. In order to do this efficiently, Parviainen and Koivisto [58] introduced the *fast sparse zeta transform* algorithm (see also [8]). Fast sparse zeta transform generalizes the fast zeta transform algorithm to work over the downsets of P. It can be used to compute values $\alpha_v(S)$ for all downsets S of P for a fixed v in $O(n|\mathcal{D}(P)|)$ time and $O(|\mathcal{D}(P)|)$ space. The whole algorithm thus requires $O(n^2|\mathcal{D}(P)| + n^{k+1})$ time. (Again, the time requirement can be improved to $O(bn|\mathcal{D}(P)| + n^{k+1})$ for bucket orders with maximum bucket size b.)

4.2.3 Probabilities of all arcs

Koivisto [38] showed that it is possible to extend the exact dynamic programming algorithm of Section 2.3.1 to compute simultaneously the posterior probabilities of all $n(n-1)/2$ potential arcs without increasing the asymptotic time requirement. This result also generalizes to partial orders [58]. The idea is to partially reorganize some of the sums and products in Equations 4.4 and 2.5, and to avoid repeating those parts of computations that need to be conducted for more than one arc. This leads to an algorithm that computes the unnormalized sample feature probabilities of all arcs in $O(n^2|\mathcal{D}(P)| + n^{k+1})$ time (or in $O(bn|\mathcal{D}(P)| + n^{k+1})$ time for bucket orders). The resulting speed-up of factor $n(n-1)/2$ is directly transferred to the algorithms of Chapter 3, if they are used to estimate the probabilities of all arcs in an order-modular posterior distribution.

4.2.4 Sampling structures

As mentioned in Section 3.2.3 about nested sampling, structures can be sampled from partial orders in two phases: first draw a linear order from $p(L|P, D)$ and then draw a structure from $p(A|L, D)$.

Consider first the problem of drawing a linear order from $p(L|P, D)$. Such a sample can be obtained by backtracking the dynamic programming algorithm used to compute F according to Equation 4.5: Positions in L are assigned to nodes in order starting from the last position. Starting from $S = N$, on each step, first a random node v is removed from S according to probability $\alpha_v(S \setminus \{v\})F(S \setminus \{v\})/F(S)$ and then S is assigned to L_v. If F and α are given, one step consumes $O(n)$ time and thus drawing a linear order takes $O(n^2)$ time ($O(bn)$ time for bucket orders).

Sampling structures from linear orders was handled in Section 4.1.2. The solutions presented there are sufficient if only one (or few) structure is drawn from each partial order. On the other hand, drawing multiple independent structures from $p(A|P, D)$ requires that each is generated from an independently sampled linear order. As a consequence, the Alias method cannot directly speed up sampling. A straightforward implementation thus requires $O(n^2+n^{k+1})$ time per a pair of a linear order and a structure, which severely limits number of structures that can be drawn in a reasonable time.

We obtain a more efficient structure sampling algorithm by drawing each random parent set $A_v \subseteq L_v$ in two phases: 1. A set $S \subseteq L_v$ that appears in Equation 4.6 is drawn by backtracking the fast sparse zeta transform. 2. The parent set $A_v \in \mathcal{T}_S(P)$ is drawn from

$$p\left(A_v \,|\, D, L_v, A_v \in \mathcal{T}_S(P)\right) \;=\; \frac{\rho_v(A_v)\, p(D_v \,|\, D_{A_v}, A_v)}{\sum_{A'_v \in \mathcal{T}_S(P)} \rho_v(A'_v)\, p(D_v \,|\, D_{A'_v}, A'_v)}\,.$$

The first phase can be done in $O(n^2)$ time (in $O(b^2)$ time for bucket orders) and the second phase trivially in $O(\mathcal{T}_S(P))$ time. However, now it is possible to apply the Alias method to the second phase separately for each possible S. In the resulting algorithm the second phase requires a constant time per sample, thus yielding $O(n^3)$ total time per sampled structure (or $O(nb^2)$, if P is a bucket order with bucket size of at most b).

4.3 Per structure: counting linear extensions

In Section 3.2.3 we saw that it is possible to obtain estimates from a modular posterior distribution by sampling linear orders or partial orders from a biased order-modular distribution and nestedly sampling structures from

the orders. However, as shown in Equation 3.5, the method requires that each structure $A^{(t)}$ is weighted according to $w^{(t)} \propto q(A^{(t)})/p(A^{(t)})$, where q is the modular target distribution and p is the order-modular sampling distribution. If p is defined so that the factors $\rho_v(A_v)$ of the structure prior are the same as in q and $\pi_v(L_v)$ is a constant for all v and L_v, then $p(A^{(t)})/q(A^{(t)})$ is proportional to the number of linear extensions of $A^{(t)}$. Thus, a reasonably efficient way of counting linear extensions is required to implement the bias correction.

Unfortunately, counting the linear extensions of an arbitrary DAG is a #P-complete problem, so it is not likely that there is a polynomial time algorithm that solves it [10]. The trivial brute force approach that enumerates through all $n!$ possible linear extensions is clearly too slow for other than very small DAGs. With a straightforward dynamic programming over subsets of nodes, one can obtain a $O(n2^n)$ time requirement. Paper II proposes a slightly improved version, which works over the downsets of the transitive closure of the DAG, leading to a $O(n|\mathcal{D}(A)|)$ time requirement and also roughly to a $O(n|\mathcal{D}(A)|)$ space requirement, where $\mathcal{D}(A)$ denotes the set of the downsets of the transitive closure of structure A. In the worst case the structure is empty and $|\mathcal{D}(A)| = 2^n$. In practice, though, the algorithm can scale to over 40 nodes even for relatively sparse DAGs. Still, the algorithm is quite basic, and could probably be further developed.

Instead of being counted exactly, the number of linear extensions could also be approximated. Since the counting problem is self-reducible, an algorithm for generating linear extensions almost uniformly implies an existence of fully polynomial-time randomized approximation scheme for counting the linear extensions [35]. Several such randomized approximation algorithms have been introduced [19, 36, 10, 11, 33]. Unfortunately, the time requirements of such methods have been too high to be practical for our purposes. Further improvements, however, could make them a viable alternative for exact counting.

Chapter 5

Local learning

While the two previous chapters discussed full Bayesian structure learning, this chapter concentrates on the local structure learning problem. In the end of the chapter, we will also briefly discuss the use of local learning algorithms for the construction of a complete Bayesian network structure.

This chapter is based on Paper V.

5.1 The local learning problems

In local structure learning, one is given data and a target node/variable from the unknown data generating Bayesian network, and the task is to solve one or both of the following two problems: One problem is to find the neighbors of the given target node. The neighbors are interesting since they are the only nodes that are directly dependent on the target. The other problem is to find the *Markov blanket* of the target, defined as a minimal set (inclusion-wise) of other nodes that makes the remaining nodes (conditionally) independent of the target [59].[3] While this property is interesting by itself, it can also be a useful criterion to apply to *feature selection*, that is, the problem of selecting a manageable (typically a smallest) subset of a large number of variables for classifying the target variable [1]. The classification task itself can be performed by any method, for example a naive Bayes classifier or a support vector machine (SVM). Since the Markov blanket is a minimal set of other nodes that provides maximum information about the value of the target, it provides an ideal solution for the feature selection problem.

[3]Pearl did not originally require minimality of a Markov blanket. What is usually today called a Markov blanket, was called a *Markov boundary* by Pearl [59].

Before attempting to learn parts of the structure of the Bayesian network from data, it is necessary to define what is a correct structure. We assume that a correct structure of the network contains the data generating distribution and is minimal, as clearly trying to learn arcs that have no support in the data is not interesting. However, this might still leave the neighbor relations ambiguous. Thus, we also assume that the data generating distribution is faithful, in which case the skeleton (and thus the neighbors) of a correct structure is uniquely defined. In addition, it then follows that the Markov blanket of a target node consists of the neighbors and the spouses of the target and is also uniquely defined [59, 74]. (Indeed, often the Markov blanket of a node is actually defined this way: not according to the distribution of interest, but to be the node set that consists of the neighbors and the spouses in the structure of the Bayesian network.)

5.1.1 Approaches to local learning

When do we need specialized methods for local structure learning? If the number of variables is small, then one can just learn the full structure (that is, do *global learning*) and extract the parts of interests from it. But for domains with a large number of variables, learning the full structure can be infeasible. In such cases, concentrating only on the region of interest can make the problem feasibly solvable. Several local learning algorithms have been proposed either directly for the local learning task itself [41, 71], or as an intermediate step in a heuristic construction of the full structure [47]. More recently, Aliferis et al. [1] presented the *Generalized Local Learning* (GLL) framework that can be instantiated several ways to represent many previously proposed algorithms as well as possible novel approaches. They also showed that under certain assumptions, all algorithms conforming to the GLL framework are sound, that is, converge to the correct solution in the limit of large data.

All previous local learning algorithms have been constraint based. This is a natural choice, since as opposed to the score-based approach that is very global in nature, the constraint-based approach is based on local decisions. In global structure learning, however, score-based methods are known to be more robust than constraint-based methods. This raises a question, whether one can get similar benefits by applying the score-based approach to local learning. Indeed, one can obviously execute a score-based algorithm on subsets of the variables. If the subsets are small, then the calls of the score-based optimization algorithm will typically be significantly faster than a call for all variables would be. The real question is, how to select such subsets and how to use the results of the calls to form solutions to the local

learning problems. One possible answer to this question is given in Paper V, that proposes score-based variants of the GLL framework algorithms. The approach taken by GLL and the proposed score-based algorithms is outlined in the next section.

5.2 Algorithms for learning neighbors and spouses

Most local learning algorithms, including those that are instances of the GLL framework, use the following scheme for finding the neighbors of the target node.

Algorithm: FindPotentialNeighbors
Input: Data D on node set N, target node $t \in N$.
Steps: Start with $H = \emptyset$. Add the nodes in $N \setminus \{t\}$ to H one by one (or in larger chunks). After each addition, remove from H the nodes that are found to be non-neighbors of t. In the end, return H.

If the assumptions that the data generating distribution is faithful and some assumptions about the non-neighbor identification step hold, it can be shown that the subset of nodes returned by the above algorithm contains all true neighbors of the target. But typically the returned set can also contain false positives, that is, nodes that are not neighbors of the target in the true structure. Turns out, that such nodes may be removed based on the simple fact that the neighboring relation is symmetric: to detect if $v \in H$ is a true neighbor of the target t, it is enough to check if t also belongs to the potential neighbors of v. The complete neighbor finding scheme with symmetry correction is therefore as follows:

Algorithm: FindNeighbors
Input: Data D on node set N, target node $t \in N$.
Steps: Call FindPotentialNeighbors on t to find its potential neighbors H. For each $v \in H$, call FindPotentialNeighbors on v and remove v from H if t is not a potential neighbor of v. In the end, return H.

The version of the above scheme that is part of the GLL framework is called GLL-PC. Aliferis et al. [1] showed, that under the faithfulness assumption and some assumptions about non-neighbor detection, in the limit of large data size GLL-PC returns the true neighbors of the target.

The GLL framework uses statistical independence tests for detecting the non-neighbors in the FindPotentialNeighbors algorithm. Paper V proposes a version of this scheme, that calls a score-based structure learning

algorithm on subsets of nodes and uses the results to determine the non-neighbors. The scoring function and the optimization algorithm used as a subroutine can be chosen freely.

Consider then the problem of learning the Markov blanket of a target node. The task can be divided into two separate subtasks: learning the neighbors of the target and learning the spouses of the target. Finding the neighbors was handled above. Once the neighbors have been identified, the following approach can be used to find the spouses.

Algorithm: FindSpouses
Input: Data D on node set N, target node $t \in N$, and its neighbors H.
Steps: Apply FindNeighbors to each neighbor of t to find all neighbors of the neighbors of t, and call them the potential spouses. For each potential spouse, determine whether it is a true spouse of t. Return the detected true spouses.

In the GLL framework the corresponding algorithm is called GLL-MB. Aliferis et al. [1] showed that, like GLL-PC, under certain assumptions, GLL-MB returns the correct result (that is, the true Markov blanket of the target) in the limit of large data size.

In the above sketch, the way, how the true spouses are determined, is not given. In GLL-MB the true spouses are detected by statistical independence tests. Again, Paper V proposes a version that uses score-based learning for detecting the spouses.

Do the score-based local learning algorithms (SLL) provide similar guarantees in the limit of large data as GLL? Paper V provides a partial proof of the correctness in the limit. Whether a complete proof can be found, still remains an open question. In spite of the lack of full theoretical guarantees, the experiments in the paper show that in many cases score-based local learning outperforms the previous state-of-the-art constraint-based GLL-algorithm, HITON [3, 1]. Figure 5.1 shows some of these results for learning Markov blankets.

The experiments of Paper V indicate that also the speed of SLL is competitive with GLL. Still, there are some potential ways to make SLL faster. First, how large H can grow during the execution of FindPotentialNeighbors may depend on the order in which the nodes are added to H. If H can be kept smaller, then the optimization steps become faster. Second, the choice of the optimization algorithm also affects the speed. While the implementation of SLL that was compared in the experiments uses the dynamic programming algorithm (from Section 2.3.1), more recent global structure learning algorithms (mentioned in Section 2.3.2) could potentially

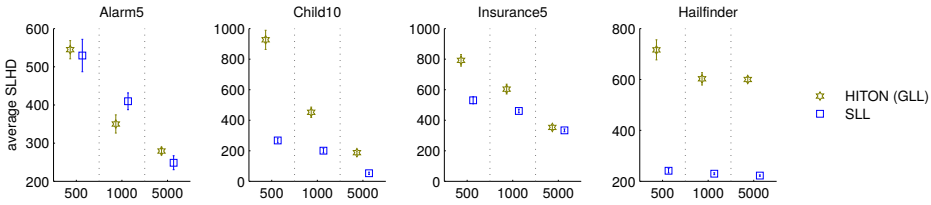


Figure 5.1: Markov blanket learning performance of HITON (constraint based) and SLL (score based) on data generated from four different Bayesian networks (185, 200, 135 and 56 nodes). Three of the networks (Alarm5, Child10, Insurance5) were generated by tiling together 5 or 10 copies of the original network using a method by Tsamardinos et al. [73]. From each network, 10 independent datasets of sizes 500, 1000 and 5000 data points (X-axis) were generated. For each dataset, the algorithms were run with each node as a target, and the total number of differences between the returned results and the correct sets of nodes (SLHD) was measured. Y-axis shows the average SLHD and its empirical standard deviation over the 10 independent runs.

lead to large speedups in some cases.

5.3 From local to global

Since it seems that a local score-based approach can lead to an improved local learning performance, a question arises, whether these local learning techniques can be used to improve global learning. In *local-to-global* learning, one first applies a local learning algorithm to multiple nodes and then combines the results into a full structure [2]. In an broad sense, all constraint-based global learning algorithms are local-to-global algorithms. Tsamardinos et al. [72] presented a dedicated local-to-global algorithm MMHC, that applies their previous MMPC neighbor learning algoritm to each node in order to construct the skeleton and then orients the edges by a greedy search algorithm. Aliferis et al. [2] proposed a more general local-to-global learning framework, and another MMHC based algorithm that uses HITON-PC (that is, the neighbor learning part of the HITON algorithm) instead of MMPC. Learning Markov blankets has also been proposed as an intermediate step for global learning [47].

Paper V proposes two local-to-global methods, SLL+G and SLL+C, that are based on SLL. SLL+G takes an approach similar to MMHC: It first finds a potential skeleton by learning the potential neighbors of each

node, and then uses a greedy search algorithm to select and orient the edges. SLL+C, on the other hand, uses a similar method to construct the skeleton, but learns also the spouses of each node after that, and uses the information gathered in the spouse learning phase to orient the edges.

Chapter 6

Conclusions

This thesis studied solutions to problems that are faced in Bayesian network structure learning when the large number of variables makes finding the exact global solution infeasible.

Papers I–III concentrated on improving the Order MCMC algorithm for Bayesian structure learning. First, Paper I introduced the idea of sampling partial orders instead of linear orders and proposed a sampler that uses bucket orders, a simple partial order scheme that includes linear orders as a special case. Papers I and III demonstrated that (1) moderately increasing the bucket size—and thus moving away from linear orders—can dramatically improve the mixing of the Markov chains and the estimates of arc probabilities, with only negligible increase in per-sample computation times. However, it remains an open question, whether bucket orders are the best partial order scheme for the task.

Papers II and III applied two tempering based methods, MC^3 and AIS to the problem and showed that both methods (2) improve the mixing and arc probability estimates and (3) allow the estimation of the model marginal likelihood. In addition, as the samples provided by AIS are independent and unbiased, it (4) allows computing lower bounds for the marginal likelihood with a guaranteed high probability. An open question is, whether it is possible to compute similar upper bounds for the marginal likelihood, and further, lower and upper bounds for feature probabilities. These two papers also showed that (5) the bias introduced by order based sampling can be corrected by sampling weighted structures from the (partial or linear) orders and this approach is often superior to the previously proposed method, although there can be cases where the convergence to the corrected probabilities is impractically slow. An obvious improvement would be a faster algorithm for counting linear extensions, either exact or approximate, that would allow both a larger number of nested samples and

scaling to larger and/or sparser networks. Another possible direction could be to apply a tempering approach, such as AIS, also to the nested sampling phase. This could potentially produce smoother structure weights, thus resulting in lower estimate variances.

In Paper IV the goal was to speed up the per-sample computations of the order based sampling. To this end, the paper introduced Treedy, a data structure and an algorithm for approximating sums over the subsets of arbitrary query sets ("subset counting queries") as well as a algorithm for related sampling tasks ("subset sampling queries"). The experiments showed that Treedy can give a substantial speedup compared both to exact computation and to the previously proposed approximating heuristic. In its current form Treedy is designed for linear order sampling, and thus an obvious future challenge would be to generalize it for partial orders. However, there are a couple concerns: First, there are unresolved questions regarding the optimal way to do the generalization. And second, since Treedy makes the per-sample computations faster, the extra computations required by partial orders start to dominate more easily. Thus, to keep the acquired speed benefit partial orders must be kept thin, which reduces the advantage over linear orders. Another question regarding Treedy is, whether some heuristic changes in the algorithm could possibly result in further improvements.

Finally, Paper V presented a new score-based algorithm for local structure learning and showed that it often provides better results than the previous state-of-the-art constraint-based algorithm, although usually being also considerably slower. In addition to possible heuristic improvements to the algorithm, a task that remains incomplete is to show whether the algorithm is asymptotically correct, that is, always converges to the correct solution when data size is increased.

References

[1] Constantin F. Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part I: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11:171–234, 2010.

[2] Constantin F. Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part II: Analysis and extensions. *Journal of Machine Learning Research*, 11:235–284, 2010.

[3] Constantin F. Aliferis, Ioannis Tsamardinos, and Alexander Statnikov. HITON: A novel Markov blanket algorithm for optimal variable selection. In *AMIA Annual Symposium Proceedings*, pages 21–25. American Medical Informatics Association, 2003.

[4] Mark Bartlett and James Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*. Forthcoming.

[5] Alexis Battle, Martin Jonikas, Peter Walter, Jonathan Weissman, and Daphne Koller. Automated identification of pathways from quantitative genetic interaction data. *Molecular Systems Biology*, 6:379–391, 2010.

[6] Ingo Beinlich, Henri J. Suermondt, R. Martin Chavez, and Gregory F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine (AIME)*, pages 247–256. Springer Berlin Heidelberg, 1989.

[7] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9:61–63, 1962.

[8] Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2012.

[9] Remco Bouckaert. Probabilistic network construction using the minimum description length principle. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 41–48. Springer Berlin Heidelberg, 1993.

[10] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8:225–242, 1991.

[11] Russ Bubley and Martin Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201:81–88, 1999.

[12] Wray Buntine. Theory refinement on Bayesian networks. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 52–60. Morgan Kaufmann, 1991.

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

[14] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Reseach*, 3:507–554, 2002.

[15] Gregory Cooper. An overview of the representation and discovery of causal relationships using Bayesian networks. In Clark Glymour and Gregory Cooper, editors, *Computation, Causation, and Discovery*. AAAI Press and The MIT Press, 1999.

[16] Gregory Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[17] Jukka Corander, Magnus Ekdahl, and Timo Koski. Parallell interacting MCMC for learning of topologies of graphical models. *Data Mining and Knowledge Discovery*, 17:431–456, 2008.

[18] James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 153–160. AUAI Press, 2011.

[19] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991.

[20] Daniel Eaton and Kevin Murphy. Bayesian structure learning using dynamic programming and MCMC. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 101–108. AUAI Press, 2007.

[21] Byron Ellis and Wing Hung Wong. Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103:778–789, 2008.

[22] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 47–58. ACM, 2004.

[23] Nir Friedman and Daphne Koller. Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1–2):95–125, 2003.

[24] Dan Geiger and David Heckerman. Learning gaussian networks. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 235–243. Morgan Kaufmann, 1994.

[25] Charles Geyer. Markov chain Monte Carlo maximum likelihood. In *Proceedings of the 23rd Symposium on the Interface*, pages 156–163. Interface Foundation of North America, 1991.

[26] Paolo Giudici and Robert Castelo. Improving Markov chain Monte Carlo model search for data mining. *Machine Learning*, 50:127–158, 2003.

[27] Paolo Giudici, Peter Green, and Claudia Tarantola. Efficient model determination for discrete graphical models. Discussion Paper 99-93, Department of Statistics, Athens University of Economics and Business, 2000.

[28] Marco Grzegorczyk and Dirk Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71:265–305, 2008.

[29] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[30] Ru He, Jin Tian, and Huaiqing Wu. Structure learning in Bayesian networks of moderate size by efficient sampling. arXiv:1501.04370 [cs.AI], 2015.

[31] David Heckerman. A tutorial on learning with Bayesian networks. Technical report, Microsoft Research, 1995.

[32] David Heckerman, Dan Geiger, and David Maxwell Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[33] Mark Huber. Fast perfect sampling from linear extensions. *Discrete Mathematics*, 306:420–428, 2006.

[34] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 358–365. Society for Artificial Intelligence and Statistics, 2010.

[35] Mark Jerrum, Leslie Valiant, and Vijay Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[36] Alexander Karzanov and Leonid Khachiyan. On the conductance of order Markov chains. *Order*, 8:7–15, 1991.

[37] Robert Kennes and Philippe Smets. Computational aspects of the Möbius transform. In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 401–416. Elsevier Science Inc., 1990.

[38] Mikko Koivisto. Advances in exact Bayesian structure discovery in Bayesian networks. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 241–248. AUAI Press, 2006.

[39] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[40] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[41] Daphne Koller and Mehran Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

[42] Timo Koski and John Noble. A review of Bayesian networks and structure learning. *Mathematica Applicanda*, 40:51–103, 2012.

[43] S. Kou, Qing Zhou, and Wing Wong. Equi-energy sampler with applications in statistical inference and statistical mechanics. *The Annals of Statistics*, 34:1581–1619, 2006.

[44] Steffen Lauritzen and David Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50:157–224, 1988.

[45] Moshe Lichman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2013.

[46] David Madigan and Jeremy York. Bayesian graphical models for discrete data. *International Statistical Review*, 63:215–232, 1995.

[47] Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 505–511. MIT Press, 1999.

[48] Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[49] Radford Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

[50] Teppo Niinimäki and Mikko Koivisto. Annealed importance sampling for structure learning in Bayesian networks. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1579–1585. AAAI Press, 2013.

[51] Teppo Niinimäki and Mikko Koivisto. Treedy: A heuristic for counting and sampling subsets. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 469–477. AUAI Press, 2013.

[52] Teppo Niinimäki and Pekka Parviainen. Local structure discovery in Bayesian networks. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 634–643. AUAI Press, 2012.

[53] Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Structure discovery in Bayesian networks by sampling partial orders. (Submitted).

[54] Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Partial order MCMC for structure discovery in Bayesian networks. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 557–565. AUAI Press, 2011.

[55] Sascha Ott and Satoru Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14(14):124–133, 2003.

[56] Art B. Owen. *Monte Carlo Theory, Methods and Examples.* 2013. Available at `http://statweb.stanford.edu/~owen/mc/`.

[57] Pekka Parviainen and Mikko Koivisto. Exact structure discovery in Bayesian networks with less space. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 436–443. AUAI Press, 2009.

[58] Pekka Parviainen and Mikko Koivisto. Bayesian structure discovery in Bayesian networks with less space. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistic (AISTATS)*, pages 589–596. JMLR: W&CP, 2010.

[59] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[60] Judea Pearl. *Causality: Models, Reasoning and Inference.* Cambridge University Press, 2000.

[61] Olivier Pourret, Patrick Naïm, and Bruce Marcot, editors. *Bayesian Networks: A Practical Guide to Applications.* John Wiley, 2008.

[62] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 3rd edition, 2007.

[63] Fred Roberts and Barry Tesman. *Applied Combinatorics.* Chapman & Hall/CRC, 2nd edition, 2009.

[64] Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 445–452. AUAI Press, 2006.

[65] Ajit Singh and Andrew Moore. Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University, 2005.

[66] John Skilling. Nested sampling. In *Proceedings of the 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MaxEnt)*, volume 735, pages 395–405. AIP Publishing, 2004.

[67] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9:62–72, 1991.

[68] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning. MIT Press, 2nd edition, 2000.

[69] Peter Spirtes and Christopher Meek. Learning Bayesian networks with discrete variables from data. In *Proceedings of 1st International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 294–299. Morgan Kaufmann, 1995.

[70] Jin Tian and Ru He. Computing posterior probabilities of structural features in Bayesian networks. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 538–547. AUAI Press, 2009.

[71] Ioannis Tsamardinos, Constantin Aliferis, and Alexander Statnikov. Algorithms for large scale Markov blanket discovery. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 376–381. AAAI Press, 2003.

[72] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 2006.

[73] Ioannis Tsamardinos, Alexander Statnikov, Laura E. Brown, and Constantin F. Aliferis. Generating realistic large Bayesian networks by tiling. In *Proceedings of the 19th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 592–597. AAAI Press, 2006.

[74] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 220–227. Elsevier Science Inc., 1990.

[75] Michael Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17:972–975, 1991.

[76] Alastair Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3:253–256, 1977.

[77] Changhe Yuan and Brandon Malone. Learning optimal Bayesian networks: a shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

[78] Changhe Yuan, Brandon Malone, and Xiaojian Wu. Learning optimal Bayesian networks using A* search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2186–2191. AAAI Press, 2011.