Department of Computer Science
Series of Publications A
Report A-2012-1

Algorithms for Exact Structure Discovery in Bayesian Networks

Pekka Parviainen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, University Main Building, on 3 February 2012 at twelve o'clock noon.

University of Helsinki
Finland

Supervisors
Mikko Koivisto, University of Helsinki, Finland
Heikki Mannila, Aalto University, Finland

Pre-examiners
Tapio Elomaa, Tampere University of Technology, Finland
Kevin Murphy, University of British Columbia, Canada

Opponent
Gregory Sorkin, London School of Economics and Political Science, United Kingdom

Custos
Esko Ukkonen, 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: postmaster@cs.helsinki.fi
URL: http://www.cs.Helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

Copyright © 2012 Pekka Parviainen
ISSN 1238-8645
ISBN 978-952-10-7573-5 (paperback)
ISBN 978-952-10-7574-2 (PDF)
Computing Reviews (1998) Classification: G.3, F.2.1, F.2.3, I.2.6
Helsinki 2012
Unigrafia

Algorithms for Exact Structure Discovery in Bayesian Networks

Pekka Parviainen

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

PhD Thesis, Series of Publications A, Report A-2012-1
Helsinki, January 2012, 132 pages
ISSN 1238-8645
ISBN 978-952-10-7573-5 (paperback)
ISBN 978-952-10-7574-2 (PDF)

Abstract

Bayesian networks are compact, flexible, and interpretable representations of a joint distribution. When the network structure is unknown but there are observational data at hand, one can try to learn the network structure. This is called structure discovery. This thesis contributes to two areas of structure discovery in Bayesian networks: space–time tradeoffs and learning ancestor relations.

The fastest exact algorithms for structure discovery in Bayesian networks are based on dynamic programming and use excessive amounts of space. Motivated by the space usage, several schemes for trading space against time are presented. These schemes are presented in a general setting for a class of computational problems called permutation problems; structure discovery in Bayesian networks is seen as a challenging variant of the permutation problems. The main contribution in the area of the space–time tradeoffs is the partial order approach, in which the standard dynamic programming algorithm is extended to run over partial orders. In particular, a certain family of partial orders called parallel bucket orders is considered. A partial order scheme that provably yields an optimal space–time tradeoff within parallel bucket orders is presented. Also practical issues concerning parallel bucket orders are discussed.

Learning ancestor relations, that is, directed paths between nodes, is motivated by the need for robust summaries of the network structures when there are unobserved nodes at work. Ancestor relations are nonmodular features and hence learning them is more difficult than modular features. A dynamic programming algorithm is presented for computing posterior probabilities of ancestor relations exactly. Empirical tests suggest that ancestor relations can be learned from observational data almost as accurately as arcs even in the presence of unobserved nodes.

Computing Reviews (1998) Categories and Subject Descriptors:

G.3	[Probability and Statistics] Multivariate Statistics
F.2.1	[Analysis of Algorithms and Problem Complexity] Numerical Algorithms and Problems - Computation of transforms
F.2.3	[Analysis of Algorithms and Problem Complexity] Tradeoffs between Complexity Measures
I.2.6	[Artificial Intelligence] Learning - Knowledge acquisition, Parameter learning

General Terms:
Algorithms, Experimentation, Theory

Additional Key Words and Phrases:
Bayesian networks, permutation problems, dynamic programming

Acknowledgements

First of all, I would like to thank my advisors. I am deeply grateful to Dr. Mikko Koivisto who has guided me through all these years. I thank Prof. Heikki Mannila who in his role as an “elder statesman” provided the research infrastructure and funding and occasional words of wisdom.

My pre-examiners Prof. Tapio Elomaa and Prof. Kevin Murphy gave comments that helped me finish the thesis. Other persons who gave feedback about the thesis were Dr. Michael Gutmann, Antti Hyttinen, Laura Langohr, and Dr. Teemu Roos. I also thank Marina Kurtén who helped improve the language of the thesis.

Department of Computer Science has been a nice place to work. I thank current and former colleagues, especially Esther Galbrun, Esa Junttila, Jussi Kollin, Janne Korhonen, Pauli Miettinen, Teppo Niinimäki, Stefan Schönauer, Jarkko Toivonen, and Jaana Wessman, for their contributions to various scientific endeavors and well-being at the work place.

Finally, I want to thank my family for their support and encouragement over the years.

Contents

Chapter 1

Introduction

Bayesian networks [83] are a widely-used class of probabilistic graphical models. A Bayesian network consists of two components: a *directed acyclic graph (DAG)* that expresses the conditional independence relations between random variables and *conditional probability distributions* associated with each variable. Nodes of the DAG correspond to variables and arcs express dependencies between variables.

A Bayesian network representation of a joint distribution is advantageous because it is compact, flexible, and interpretable. A Bayesian network representation exploits the conditional independencies among variables and provides a compact representation of the joint probability distribution. This is beneficial as in general the number of parameters in a joint probability distribution over a variable set grows exponentially with respect to the number of variables and thus storing the parameter values becomes infeasible even with a moderate number of variables. Since a Bayesian network represents the full joint distribution, it allows one to perform any inference task. Thus, it is a flexible representation. Furthermore, the structure of a Bayesian network, that is, the DAG, can be easily visualized and may uncover some important characteristics of the domain, especially if the arcs are interpreted to be causal, or in the other words, direct cause-effect relations. To illustrate the interpretability of a DAG, consider Figure 1.1 that shows a DAG describing factors affecting the weight of a guinea pig. This DAG has some historical value as it is an adaptation of the structural equation model presented by Sewall Wright in 1921 [110], which, to our best knowledge, was among the first graphical models used to model dependencies between variables. For example, an arc from node "Weight at birth" to node "Weight at 33 days" indicates that the weight at birth directly affects the weight at 33 days.

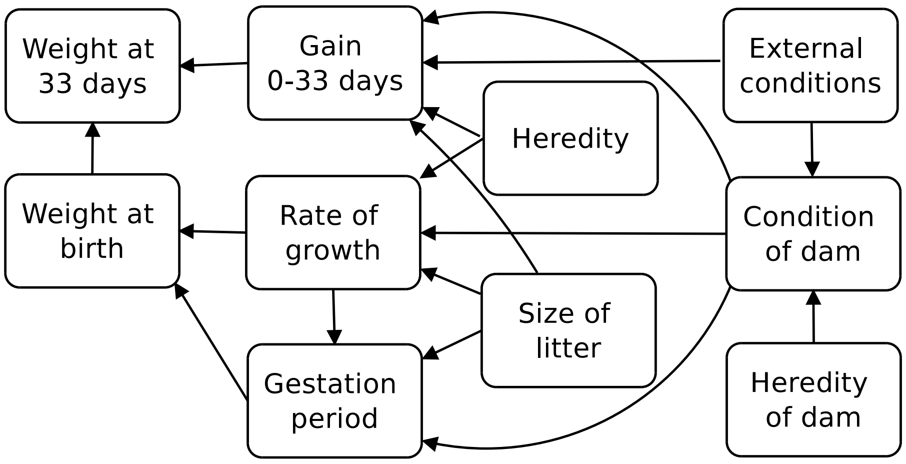


Figure 1.1: A directed acyclic graph (DAG) illustrating the interrelations among the factors which determine the weight of guinea pigs at birth and at weaning (33 days) [110].

In recent decades, Bayesian networks have garnered vast interest from different domains and they have a multitude of diverse applications. Early applications were often expert systems for medical diagnosis such as Pathfinder [49]. Bayesian networks are widely used, for example, in biology, where applications vary from bioinformatical problems like analyzing gene expression data [40, 53] to ecological applications like modeling the population of polar bears [2] and analyzing the effectiveness of different oil combating strategies in the Gulf of Finland [51]. Bayesian networks have been applied also in other fields like business and industry.

How can one find a good Bayesian network model for a phenomenon? An immediate attempt would be to let a domain expert construct the model. However, this approach is often infeasible because either the construction requires too much time and effort to be practical, or the phenomenon to be modeled is not known well enough to warrant modeling by hand. Fortunately, if an expert cannot construct a model, but one has access to a set of observations from the nodes of interest, one can attempt to *learn* a Bayesian network from data. From the machine learning point of view, the question is, how can one learn such a model accurately and with feasible computational resources?

Learning of a Bayesian network is usually conducted in two phases. First, one learns the structure of the network and then one learns the parameters of the conditional distributions. The learning of the structure, or in other words, *structure discovery*, is the more challenging and thus more interesting phase.

There are two rather distinct approaches to structure discovery in Bayesian networks. The constraint-based approach [84, 99, 100, 109] relies on testing independencies and dependencies between variables and constructing a DAG that represents these relations. The score-based approach [19, 47], on the other hand, is based on assigning each DAG a score according to how well the DAG fits to the data. Compared to the constraint-based approach, the score-based approach benefits from being able to assess the uncertainty in the results, to incorporate prior knowledge and to combine several models. The constraint-based approach, however, enables principled and computationally feasible handling of unobserved variables. For a more thorough comparison of the approaches, see, for example, texts by Heckerman et al. [48] or Neapolitan [75, p. 617–624].

Structure discovery in Bayesian networks is a host of several interesting problem variants. In the *optimal structure discovery* (OSD) problem the goal is to find a DAG that is a "best" representation of the data. In the constraint-based approach this means that the DAG explains the dependencies and independencies in the data. In the score-based approach, on the other hand, one tries to find a DAG that maximizes the score. However, it is not always necessary or even preferable to try to infer a single optimal DAG. The score-based approach allows one to take a so-called Bayesian approach [39, 65] and report posterior probabilities of structural features. There are several types of structural features. Computing posterior probabilities of modular features, such as arcs, constitutes the *feature probability* (FP) problem. Further, one may compute posterior probabilities of *ancestor relations*, that is, directed paths between two nodes. In this thesis, we concentrate on the score-based approach, which allows one to take advantage of different problem variants.

All the aforementioned problems are similar in the sense that (under some usual modularity assumptions) the score or the posterior probability of a DAG decomposes into local terms. However, one also has to take into account a global constraint, namely acyclicity. Further, the difference between modular features and ancestor relations is that modular features are local but ancestor relations are global properties of a DAG. The outline of the OSD and FP problems is similar to several classic combinatorial problems like traveling salesman (TSP): One has local costs (or scores) and one wants to find an optimal solution under some global constraint. It turns out that OSD, FP, and TSP can be formulated in such a way that the global constraint is handled by maximizing or summing over permutations of nodes. Thus, such problems are members of a broad class of computational problems called *permutation problems.* Although the score of a DAG is constructed from local scores, the global constraint makes the structure discovery problems computationally challenging. For example, given the data and a consistent scoring criterion[1], finding an optimal Bayesian network is an NP-hard problem [14, 17] even in the case of finding an optimal DAG among some classes of simple DAGs such as polytrees [22] or path graphs [74][2]. Also the feature probability variant seems to be similarly presumably hard, though a formal proof is missing. The hardness of the structure discovery problems has prompted active development of various heuristics. Common approaches include greedy equivalence search [16], max-min hill climbing [107], Markov chain Monte Carlo (MCMC) [32, 39, 45, 72] and genetic algorithms [68].

Although structure discovery in Bayesian networks is hard, it is possible to solve structure discovery problems in small and moderate-sized networks exactly. Compared to the heuristics, *exact algorithms* that are guaranteed to find an optimal solution offer some benefits. As there is no (extra) uncertainty about the quality of the results, the user may concentrate on modeling the domain and interpreting the results. The benefits and theoretical interest have motivated the development of several exact score-based algorithms for structure discovery in Bayesian networks. Most of these algorithms deploy dynamic programming across node sets [27, 62, 65, 73, 78, 79, 86, 96, 98, 103, 112] and run in $O^*(2^n)$ time and space (under some usual modularity assumptions), where n is the number of nodes. The $O^*(\cdot)$ notation is used throughout this thesis to hide factors polynomial in n. Recently, some new approaches, based on branch-and-bound [24, 25, 33] and linear programming [21, 54] have been introduced. These new methods are often fast in a good case, but we do not know whether they are as fast as dynamic programming in the worst case.

In this thesis, we study exact dynamic programming algorithms. Traditionally such exponential-time algorithms have been considered unsatisfactory. However, not all exponential algorithms are useless in practice and as is the case with Bayesian networks, it is often possible to solve moderate-sized problem instances. Besides, for NP-hard problems there is no hope for (worst-case) polynomial time algorithms unless P equals NP. This has motivated lots of research in exponential algorithms; see, for example, a recent textbook by Fomin and Kratsch [35].

[1]A scoring criterion is consistent if it is maximized by the "true" network structure when the size of the data approaches infinity.

[2]As an exception, an optimal Bayesian tree can be found in polynomial time [18].

Although both the time and the space requirement of the dynamic programming algorithms grow exponentially in n, the space requirement is the bottleneck in practice. Modern desktop computers usually have a few gigabytes of main memory, which enables the learning of networks of about 25 nodes with running times of a few hours. The state-of-the-art implementation by Silander and Myllymäki [96] uses the hard disk to store a part of the results. For a network of 29 nodes it consumes almost 100 gigabytes of space, while the running time is only about ten hours. For a 32-node network the space requirement is already almost 800 gigabytes. Therefore, in order to learn larger networks, reducing the space requirement is the top priority. Unfortunately, reducing the space requirement without increasing the time requirement seems to be a difficult task. This observation motivates the first research question in this thesis: How can one trade space against time efficiently in structure discovery in Bayesian networks?

To answer the previous question, the first contribution of this thesis is about *space–time tradeoff* schemes for structure discovery in Bayesian networks. We present algorithmic schemes based on dynamic programming for trading space against time for permutation problems in general; the schemes are then adapted for the OSD and FP problems. We introduce a partial order approach on which we base most of our schemes. We also investigate the practicality of our approach by applying it with a particular family of partial orders called *parallel bucket orders*. It turns out that our schemes allow learning of larger networks than the previous dynamic programming algorithms. Furthermore, our schemes are easily parallelized.

Another research question in this thesis is motivated by the need of handling unobserved variables. In real life, we often encounter datasets in which the observed nodes are affected by some unobserved ones. However, although the score-based methods, especially the Bayesian ones, are flexible and are able to take into account all the prior knowledge, unobserved variables pose a computational problem for score-based methods. Often one either refuses to make any causal conclusions about the DAG or one ignores the possibility of unobserved nodes altogether and accepts an unquantified risk of erroneous claims. Therefore, we are interested in features that are preserved in the presence of unobserved nodes. The questions are: What kind of features are preserved when there are unobserved variables in play? How can one compute posterior probabilities of these features?

To answer the previous questions, we consider learning ancestor relations. One often interprets the arcs of a Bayesian network as causal cause–effect relationships. Thus, the existence of a path from node s to node t can be interpreted as s being either a direct or an indirect cause of t.

Compared to the arc probabilities, the path probabilities yield information also about indirect causes and hence they can be more interesting. In the *ancestor relations problem*, the goal is to output the posterior probability that there is a directed path between two given nodes.

Current exact Bayesian algorithms [39, 65] are able to compute posterior probabilities of modular features like arcs; however, they cannot handle the more challenging nonmodular features such as ancestor relations. The second main contribution of this thesis is a Bayesian averaging algorithm for learning ancestor relations; as far as we know, it is the first non-trivial exact algorithm for computing posterior probabilities of nonmodular features. We also test the algorithm in practice. The empirical results suggest that ancestor relations can be learned almost as well as single arcs.

Overview of the thesis. A formal treatment on Bayesian networks, the basics of permutation problems and a dynamic programming algorithm are given in Chapter 2. Important concepts related to space–time tradeoffs and two simple algorithmic schemes are presented in Chapter 3. Then we introduce the partial order approach in general (Chapter 4) and for parallel bucket orders in particular (Chapter 5). An algorithm for and empirical results about learning ancestor relations are presented in Chapter 6. We end this thesis with a discussion in Chapter 7.

Relation to the author's prior work. Parts of the results concerning space–time tradeoffs (Chapters 3–5) have been published in the following three articles:

[64] Mikko Koivisto and Pekka Parviainen. A Space–Time Tradeoff for Permutation Problems. *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[80] Pekka Parviainen and Mikko Koivisto. Exact Structure Discovery in Bayesian Networks with Less Space. *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2009.

[81] Pekka Parviainen and Mikko Koivisto. Bayesian Structure Discovery in Bayesian Networks with Less Space. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

The results concerning learning ancestor relations (Chapter 6) have been published in the following article:

[82] Pekka Parviainen and Mikko Koivisto. Ancestor Relations in the Presence of Unobserved Variables. *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2011.

Chapter 2

Preliminaries

Let us begin this chapter with a formal introduction to Bayesian networks. Further, we introduce permutation problems in general and formulate the Osd and Fp problems as permutation problems. We end this chapter by presenting a basic dynamic programming algorithm for solving permutation problems. The algorithm is extended in later chapters to work in limited space. Dynamic programming also serves as a basis for the development of an algorithm for the ancestor relation problem.

2.1 Bayesian Networks

In this section, we focus on the aspects of Bayesian networks that are needed in this thesis; for a more thorough treatment, see, for example, recent textbooks by Koller and Friedman [66] or Neapolitan [75]. We also define three different computational problems related to structure discovery in Bayesian networks: the Osd problem, the Fp problem, and the ancestor relation problem.

2.1.1 Bayesian Network Representation

A Bayesian network consists of two parts: the structure and the parameters. The structure is represented by a directed acyclic graph (DAG) and the parameters determine the conditional probability distributions for each node.

A DAG is a pair (N, A), where N is the *node set* and $A \subseteq N \times N$ is the *arc set*. Note that a DAG is acyclic, that is, the arcs in A do not form directed cycles. A node u is said to be a *parent* of a node v if the arc set contains an arc from u to v, that is, $uv \in A$. The set of the parents of v are denoted by A_v. If u is a parent of v then v is a *child* of u. Further, a node u is said to be an *ancestor* of a node v in a DAG (N, A) if there is a directed path from u to v in A; denoted $u \rightsquigarrow v$. If u is an ancestor of v, then v is a *descendant* of u. When there is no ambiguity about the node set, we identify a DAG by its arc set. The cardinality of N is denoted by n.

Each node v corresponds to a random variable and the DAG expresses conditional independence assumptions between variables. Random variables u and v are said to be conditionally independent given a set S consisting of random variables if $Pr(u, v|S) = Pr(u|S)Pr(v|S)$. A DAG represents a joint distribution of the random variables if the joint distribution satisfies the *Markov condition*, that is, every variable is conditionally independent of its non-descendants given its parents. Now it remains to specify one such distribution. This can be done using local *conditional probability distributions (CPD)* which specify the distribution of a random variable given the variables corresponding to its parents A_v. CPDs are usually taken from a parametrized class of probability distributions, like discrete or Gaussian distributions. Thus, the CPD of variable v is determined by its parameters θ_v; the type and the number of parameters is specified by the particular class of probability distributions. The parameters of a Bayesian network are denoted by θ and it consists of the parameters of each CPD. Finally, a Bayesian network is a pair (A, θ).

Example 1. Let us consider a company whose employees have a tendency to be late to work. The company conducts a study on the factors which affect being late to work. Figure 2.1 shows a Bayesian network, that summarizes the findings. Generally, the people who are not on time at work have either overslept or their bus has been late. The main reason for oversleeping is forgetting to set the alarm. Let us use the shorthands a, b, o, and t, for nodes “Alarm on?”, “Bus Late?”, “Overslept?”, and “In Time?”, respectively.

For example, let us compute the probability that a person is at work on time when he has forgotten to set the alarm, that is, $Pr(t = \text{yes}|a = \text{no}) = Pr(t = \text{yes}, a = \text{no})/Pr(a = \text{no})$. To compute probabilities $Pr(t = \text{yes}, a = \text{no})$ and $Pr(a = \text{no})$ we need to marginalize out the variables whose values are not fixed. To this end, we get $Pr(t = \text{yes}, a = \text{no}) = 0.1 \times 0.9 \times 0.2 \times 0.1 + 0.1 \times 0.9 \times 0.8 \times 0.3 + 0.1 \times 0.1 \times 0.2 \times 0.2 + 0.1 \times 0.1 \times 0.8 \times 0.9 = 0.031$ and $Pr(a = \text{no}) = 0.1$, thus $Pr(t = \text{yes}|a = \text{no}) = 0.031/0.1 = 0.31$. In words, if a person has forgotten to set his alarm clock there is a 31 percent chance that he is at work on time. $\diamond$

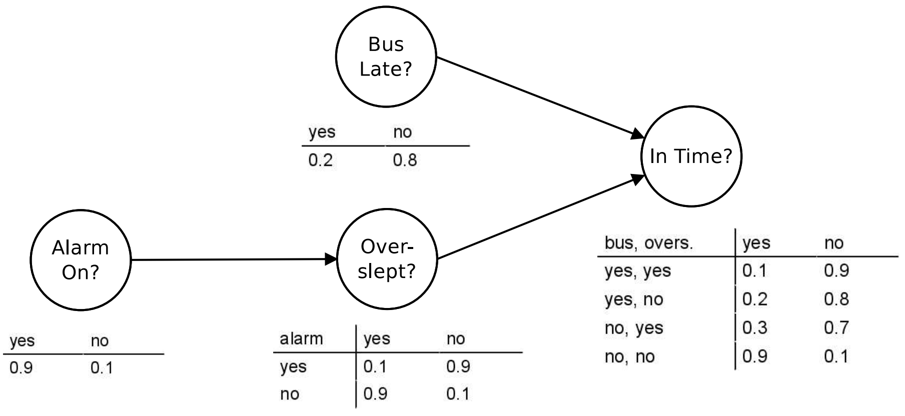


Figure 2.1: A Bayesian network illustrating factors affecting being late to work. Each node is associated with a conditional probability table which determines the probabilities of different values of the particular variable (columns) given each configuration of the values of its parents (rows).

The compactness of the Bayesian network representation can be illustrated by comparing the number of parameters needed with the joint distribution factorized according to a Bayesian network structure to the non-factorized joint distribution[1]. To this end, let us consider discrete variables and let r_v be the number of distinct values of variable v. The conditional distribution associated with node v is determined by $(r_v - 1)\prod_{u\in A_v} r_u$ parameters and thus the joint distribution can be represented with $\sum_{v\in N}(r_v - 1)\prod_{u\in A_v} r_u$ parameters. On the other hand, without a factorization one needs $\prod_{v\in N} r_v - 1$ parameters. For example, consider the joint distribution of 10 binary variables. Without a factorization one needs $2^{10} - 1 = 1023$ parameters. However, if the joint distribution can be represented by a Bayesian network with maximum indegree 3, the number of parameters needed is less than $10 \times 2^3 = 80$.

The *likelihood* of a Bayesian network, that is, the probability of data given a Bayesian network reflects how well the Bayesian network fits to the data. The data D are an $n \times m$-matrix, where the rows are associated with the nodes. Each node v is associated with m random variables $D_{v1}, D_{v2}, \ldots, D_{vm}$, which constitute the vth row, D_v of D. The m columns are often treated as observations, each column corresponding to one observation; observations are assumed to be independent and identically distributed. Given the data D on the node set N, the likelihood of a Bayesian

[1]The latter case is equivalent to factorization according to a complete DAG.

network (A, θ) can be factorized according to the DAG A as follows:

$$Pr(D|A, \theta) = \prod_{i=1}^{m} Pr(D_{\cdot i}|A, \theta) = \prod_{i=1}^{m} \prod_{v \in N} Pr(D_{vi}|D_{A_v i}, A_v, \theta_v).$$

The term $Pr(D_{vi}|D_{A_v i}, A_v, \theta_v)$ is called a *local likelihood.*

Sometimes the Bayesian network is unobserved or unknown to the modeler and one wants to learn it from the data. In Bayesian approach, one first sets up a *full probability model*, that is, a joint probability distribution of all observed and unobserved variables and then calculates the *posterior distribution* of the unobserved variables given the observed ones [43]. Thus, the unknown values, the DAG A and the parameters θ, can be included as variables in the probability model. To this end, we have a joint probability distribution of the data, the DAG, and the parameters. Based on the chain rule of probability calculus, the joint probability distribution factorizes as

$$Pr(D, A, \theta) = Pr(A)Pr(\theta|A)Pr(D|A, \theta).$$

The terms $Pr(A)$ and $Pr(\theta|A)$ are called a structure prior and a parameter prior, respectively. Next, we will discuss common assumptions concerning the priors.

Consider the *structure prior* $Pr(A)$. In the Bayesian approach, a prior distribution reflects the beliefs of the modeler about uncertainty of the value of a variable. For a structure prior, one commonly assumes a modular prior [95, 96, 105], which often enables convenient computation. A prior $Pr(A)$ is *modular* if it can be written as $Pr(A) = \prod_{v \in N} q_v(A_v)$, where q_v is a nonnegative function. A common modular prior is the *uniform prior* over all DAGs, that is, $q_v(A_v) \propto 1$. In some cases using a modular prior in structure discovery, however, can lead to a significant increase in running time as we will see in Section 2.3.2.

Sometimes it is more convenient to assume an *order-modular prior* [39, 65][2]. To this end, we introduce a new random variable, linear order L on

[2]We note that an order-modular prior often assigns different probabilities to the different members of a Markov equivalence class and favors networks that have several topological orderings. Although many researchers (see, for example, Silander [95] or Tian and He [105]) seem to consider the uniform prior as the default or "true" prior, from the Bayesian point of view assuming an order-modular prior is neither an advantage nor a disadvantage, besides the computational advantage. Indeed, an order-modular prior may sometimes better reflect the modeler's subjective beliefs. One should also notice that generally we are only able to learn structures up to a Markov equivalence class and the uniform prior over DAGs is not uniform over equivalence classes as pointed by Friedman and Koller [39]. For a more thorough discussion about different priors, see, for example, Angelopoulos and Cussens [3].

N and define a joint prior of L and a DAG A, $Pr(L, A)$. A *linear order* L on base-set N is a subset of $N \times N$ such that for all $x, y, z \in N$ it holds that

1. $xx \in L$ (reflexivity),
2. $xy \in L$ and $yx \in L$ imply $x = y$ (antisymmetry),
3. $xy \in L$ and $yz \in L$ imply $xz \in L$ (transitivity), and
4. $xy \in L$ or $yx \in L$ (totality).

If $xy \in L$ we say that x *precedes* y. We write L_v for the set of nodes that precede v in L and say that L and A are *compatible* if $A_v \subseteq L_v$ for all v, that is, L is a topological ordering of A. The joint prior is defined as

$$Pr(L, A) = \begin{cases} 0 & \text{if } L \text{ is not compatible with } A, \\ \prod_{v \in N} \rho_v(L_v) q_v(A_v) & \text{otherwise,} \end{cases}$$

where ρ_v and q_v are nonnegative functions. The prior $Pr(A)$ is obtained by marginalizing the joint prior over linear orders, that is,

$$Pr(A) = \sum_L Pr(L, A).$$

Note that if $\rho_v(L_v)$ and $q_v(A_v)$ are constant functions then the prior probability of a DAG is proportional to the number of its topological orderings.

Often one wants to restrict the number of possible parent sets of a node. To this end, one can specify a set of possible parent sets, $\mathcal{F}_v$, for each node v. To guarantee that $Pr(A) = 0$ if $A_v \notin \mathcal{F}_v$ for some v, it is sufficient (for both modular and order-modular priors) to specify that $q_v(A_v) = 0$ if $A_v \notin \mathcal{F}_v$. We will return to handling parent sets in Section 3.2.1.

The *parameter prior* $Pr(\theta|A)$ determines the probability of the parameters given a DAG. It is usually assumed (see, for example, Friedman and Koller [39]) that the prior admits *global parameter independence*, that is, $Pr(\theta|A) = \prod_{v \in N} Pr(\theta_v|A)$ and *parameter modularity*, that is, given two DAGs A and A' in which $A_v = A'_v$, then $Pr(\theta_v|A) = Pr(\theta_v|A')$. Combining global parameter independence and parameter modularity yields $Pr(\theta|A) = \prod_{v \in N} Pr(\theta_v|A_v)$.

As we in this thesis are concerned with structure discovery in Bayesian networks, the parameters of the conditional probability distributions play merely a role of nuisance variables. To this end, we integrate the parameters out and obtain *marginal likelihood* $Pr(D|A)$ of A. Heckerman et al. [47][3] show that if the parameter prior satisfies global parameter independence and parameter modularity then

$$\begin{aligned} Pr(D|A) &= \int_{\theta} Pr(D|A,\theta)Pr(\theta|A,)\mathrm{d}\theta \\ &= \prod_{v\in N} \int_{\theta_v} \prod_{i=1}^{m} Pr(D_{vi}|D_{A_v i}, A_v, \theta_v)Pr(\theta_v|A_v)\mathrm{d}\theta_v \\ &= \prod_{v\in N} Pr(D_v|D_{A_v}, A_v). \end{aligned}$$

We call $Pr(D_v|D_{A_v}, A_v)$ the *local marginal likelihood*. The local marginal likelihood expresses how well the data on the node set $A_v \cup \{v\}$ fit to some class of conditional probability distributions specified by the modeler. For discrete variables, a common choice is to use a Bayesian score [13, 19, 47], which has a closed-form expression.

In the Bayesian approach, we are interested in the posterior probability $Pr(A|D)$ of the DAG A given the data D, that is,

$$Pr(A|D) = \frac{Pr(D|A)Pr(A)}{Pr(D)},$$

where $Pr(D)$ is the marginal probability of the data. When the data D are given, the posterior probability is proportional to the unnormalized posterior density, that is, $Pr(A|D) \propto Pr(D|A)Pr(A)$. Thus, assuming a modular structure prior

$$Pr(A|D) \propto \prod_{v\in N} Pr(D_v|D_{A_v}, A_v)q_v(A_v).$$

Often it is convenient to operate with the logarithm of the unnormalized posterior. To this end, we define the score of a DAG as

$$s(A) = \sum_{v\in N} s_v(A_v),$$

where $s_v(A_v) = \log Pr(D_v|D_{A_v}, A_v) + \log q_v(A_v)$ is called a local score.

[3] Actually, Heckerman et al. [47] derive the existence of the parameters θ given which the data columns are conditionally independent from the assumption of (infinitely) exchangeable data; an interested reader may refer to, for example, Bernardo [7] for more information about this alternative route to establish a fully specified (Bayesian network) model.

A function that assigns the score to a DAG based on the data is called a *scoring criterion.* A scoring criterion that factorizes according to the underlying DAG is said to be *decomposable.* For our purposes the scoring criterion can be considered a black-box function, as long as it is decomposable.

Next, we consider the causal interpretation of a Bayesian network. A Bayesian network is called *causal* if the arcs are interpreted as direct cause–effect relations, that is, an arc from node u to node v denotes that u is a direct cause of v. When a Bayesian network serves only as a representation of a probability distribution, knowing the causal relations does not add any value. From the knowledge discovery point of view, however, all information about causal relations between variables can be highly interesting. To this end, we discuss the concept of *identifiability* that describes to what extent one can learn the structure of a causal Bayesian network from observational data.

To consider identifiability, it is essential to define Markov equivalence. Two DAGs are said to be *Markov equivalent* if they describe the same set of conditional independence statements. To see whether two DAGs are Markov equivalent, we need to define a skeleton and a v-structure. The *skeleton* of a DAG A is an undirected graph that is obtained by replacing all directed arcs $uv \in A$ with undirected edges between u and v. Nodes s, t, and u form a *v-structure* in a DAG if there is an arc from s to u and from t to u and there is no arc between s and t. It is well-known that two networks belong to the same Markov equivalence class if and only if they have the same skeleton and the same set of v-structures [109]. A scoring criterion is said to be *score equivalent* if two Markov equivalent DAGs always have the same score. Thus, a score-equivalent score cannot distinguish two Markov equivalent DAGs based on observational data only, that is, the DAGs are nonidentifiable. This is often desired when a DAG is primarily interpreted as a set of independence constraints of some distribution. Using a score-equivalent score we are able to learn Markov equivalence classes but the most likely network within a class is determined by the prior. There are also non-score equivalent scores. A notable example is K2 score [19] which is said to yield good results in practice despite the lack of score equivalence [15].

Due to nonidentifiability, causal Bayesian networks cannot usually be learned from observational data[4]. However, if we are able to intervene on

[4]Full causal models can be learned, however, if the assumptions are suitable for that purpose. For example, linear causal models with non-Gaussian noise with no undetermined parameters can be learned from observational data; see, for example, Shimizu et al. [94].

some variables, that is, to set them to user-specified values, we can infer the directions of arcs whose direction is not specified in the particular Markov equivalence class. It is also possible to infer causal relations by averaging over all DAGs; see the feature probability problem and the ancestor relation problem defined in the next section.

The following example illustrates the Markov equivalence.

Example 2. Consider the structure of the Bayesian network in Figure 2.1. The DAG has one v-structure which consists of nodes b, o, and t (t in the middle). The DAG is Markov equivalent with the DAG $\{bt, oa, ot\}$ (and with no other DAG). Note that if the arcs are interpreted to be causal, the DAGs $\{bt, ao, ot\}$ and $\{bt, oa, ot\}$ are not equivalent as a is a cause of o in the former one but o is a cause of a in the latter one. $\diamond$

2.1.2 Computational Problems

We will subsequently tackle three different structure discovery problems. To this end, it is essential to formally define the problems.

A key task in the score-based approach is to find a maximum-a-posteriori (MAP) DAG, or equivalently, a DAG that maximizes the score. For our purposes it is convenient to define two variants of this problem with slightly different inputs. We will discuss the issues concerning the explicit and implicit input in detail in Section 3.2.1. Here $\mathcal{F}_v$ is the set of possible parent sets of node v.

Definition 2.1 (Optimal structure discovery (OSD) problem with explicit input) *Given the values of local score $s_v(Y)$ for $v \in N$ and $Y \in \mathcal{F}_v$, compute $\max_A s(A)$, where A runs through the DAGs on N.*

Definition 2.2 (Optimal structure discovery (OSD) problem with implicit input) *Given the data D on nodes N and a decomposable scoring criterion $s_v(Y)$, which evaluates to 0 if $Y \notin \mathcal{F}_v$, compute $\max_A s(A)$, where A runs through the DAGs on N.*

Remark that in OSD the goal is to compute the score of an optimal DAG. In practice we are often more interested in an optimal DAG itself. It turns out (see Section 2.3.1) that an optimal DAG can be constructed with essentially no extra computational burden.

Another remark is that the scoring criterion does not have to be based on the local marginal likelihood. The OSD problem remains reasonable even if a scoring criterion does not conveniently admit a structure prior.

For example, maximum likelihood, minimum description length [88], and Bayesian information criterion (BIC) [93] can be used as the scoring criterion.

The size of the search space of OSD, that is, the number of DAGs on n nodes, grows superexponentially with respect to n [89]: Later we will see that, thanks to the decomposable score, OSD can be solved significantly faster than by an exhaustive search.

Choosing just one DAG, even if it is optimal, can seem quite arbitrary: often there are several almost equally probable DAGs and with large node sets, even the most probable DAGs tend to be very improbable. To circumvent this problem, one can, for example, take an average over k best DAGs [106]. Another possibility is to find useful summarizations of the DAGs. To this end, one can take a full Bayesian model averaging approach (see, for example, Hoeting et al. [52]) and compute posterior probabilities of features of interest from the data. For Bayesian networks, natural candidates for such a summary feature are subgraphs, like arcs.

We associate such a structural feature f with an indicator function $f(A)$ which evaluates to 1 if A has the feature f and 0 otherwise. For computational convenience, it is often practical to consider modular features. A feature f is *modular* if the indicator function factorizes as

$$f(A) = \prod_{v \in N} f_v(A_v),$$

where $f_v(A_v)$ is a local indicator function. Note that modular features are local in the sense that for the presence of a modular feature, it is necessary and sufficient that n independent local conditions hold. For example, the indicator for an arc uv is represented by letting $f_v(A_v) = 1$ if $u \in A_v$ and $f_v(A_v) = 0$ otherwise and $f_w(A_w) = 1$ for all $w \neq v$ and all A_w. For another example, the indicator for a v-structure with arcs from nodes s and t to a node u is represented by letting $f_u(A_u) = 1$ if $s, t \in A_u$ and $f_u(A_u) = 0$ otherwise and $f_s(A_s) = 1$ if $t \notin A_s$ and $f_s(A_s) = 0$ otherwise and $f_t(A_t) = 1$ if $s \notin A_t$ and $f_t(A_t) = 0$ otherwise and $f_w(A_w) = 1$ for all other nodes w and all A_w.

Consider the computation of the posterior probability of a modular feature. We notice that $Pr(f|D) = Pr(f,D)/Pr(D)$ and

$$\begin{aligned} Pr(f,D) &= \sum_A Pr(f,D|A)Pr(A) \\ &= \sum_A Pr(f|A)Pr(D|A)Pr(A) \\ &= \sum_A f(A)Pr(D|A)Pr(A). \end{aligned} \tag{2.1}$$

Here, the first equality is by the law of total probability, the second one by the independence of f and D given A and the third one by the fact that $Pr(f|A) = f(A)$.

Recall that the marginal likelihood $Pr(D|A)$ decomposes into local terms $Pr(D_v|D_{A_v}, A_v)$ whose values can be computed from the data efficiently. Furthermore, for computational convenience (see Section 2.3.2) we assume here an order-modular structure prior. Now we are ready to define two variants of the feature probability problem.

Definition 2.3 (Feature probability (FP) problem with explicit input) *Given the values of local likelihood $Pr(D_v|D_{A_v}, A_v)$ for $v \in N$ and $A_v \in \mathcal{F}_v$, an order-modular structure prior $Pr(A)$ and a modular structural feature f, compute the posterior probability $Pr(f|D)$.*

Definition 2.4 (Feature probability (FP) problem with implicit input) *Given the data D on nodes N, local marginal likelihoods $Pr(D_v|D_{A_v}, A_v)$, an order-modular structure prior $Pr(A)$ and a modular structural feature f, compute the posterior probability $Pr(f|D)$.*

The ancestor relation problem, that is, computing the posterior of a directed path between two nodes is an extension of the FP problem to non-modular features, namely ancestor relations. The non-modularity means that the indicator function $f(A)$ of a non-modular feature does not factorize into a product of local indicator functions. The problem is defined as follows.

Definition 2.5 (Ancestor relation problem) *Given the data D on nodes N, a local likelihood function $Pr(D_v|D_{A_v}, A_v)$, an order-modular structure prior $Pr(A)$ and an ancestor relation $u \rightsquigarrow v$, $u, v \in N$, compute the posterior probability $Pr(u \rightsquigarrow v|D)$.*

2.2 Permutation Problems

Consider the OSD problem. The goal is to find a maximum of the sum of local scores under a global acyclicity constraint. To guarantee that the DAG is acyclic, one may fix a linear order and for each node choose the best parent set from the preceding nodes. To solve the OSD problem one has to consider all the linear orders on the nodes. Next, we introduce a class of computational problems called permutation problems, which are similar to the OSD problem in the sense that they can be solved in a similar fashion.

For another example of permutation problems, consider the traveling salesman problem (TSP). In TSP, a traveling salesman is given n cities and he tries to find the shortest route that visits every city exactly once and returns to the starting point. Every route can be seen as a permutation of the cities, that is, the order in which the cities are visited determines the route. The total length of the route, or the cost of the permutation, can be decomposed to a sum of local costs: the cost of moving from the ith city to the $(i+1)$th city is the distance between those particular cities. Now the problem is essentially to minimize the cost over the permutations of cities. It is well-known that TSP can be solved using a dynamic programming algorithm [6, 50]; we will return to dynamic programming in Section 2.3.

More generally, we define a class of problems called *permutation* or *sequencing problems* which ask for a permutation on a n-element set that minimizes a given cost function. The cost function decomposes into local terms, where the local cost of an element depends only on the preceding elements. Besides TSP, examples of such a problems include the feedback arc set problem, the cutwidth problem, the treewidth problem, and the scheduling problem. In Section 2.3 we will show that all permutation problems can be solved using a generic dynamic programming algorithm.

Some of the aforementioned permutation problems are not minimization problems. Therefore, it is convenient to present the permutation problems in a more general setting. To this end, we start by introducing a commutative semiring; for more, see, for example, Aji and McEliece [1] or Koivisto [61]. Recall that a binary operation $\circ$ on set R is *associative* if $(x \circ y) \circ z = x \circ (y \circ z)$ for all $x, y, z \in R$ and *commutative* if $x \circ y = y \circ x$ for all $x, y \in R$.

A *commutative semiring* is a set R together with two binary operations $\oplus$ and $\odot$ called addition and multiplication, which satisfy the following axioms:

1. The operation $\oplus$ is associative and commutative, and there is an additive identity element called "0" or zero element such that $s \oplus 0 = s$

for all $s \in R$.

2. The operation $\odot$ is associative and commutative, and there is a multiplicative identity called "1" or one-element such that $s \odot 1 = s$ for all $s \in R$.

3. The distributive law holds, that is, for all $s, t, u \in R$,

$$(s \odot t) \oplus (s \odot u) = s \odot (t \oplus u).$$

The semiring is denoted as a triple $(R, \oplus, \odot)$.

We can clearly see that ordinary addition and multiplication on real numbers form a commutative semiring. Another example of a semiring is the min–sum semiring on non-negative numbers, that is, $([0,\infty), \min, +)$, in which the Tsp problem was defined. Minimization is associative and commutative and it has a zero-element ∞ such that $\min(s, \infty) = s$ for all $s \in [0,\infty)$. Addition is associative and commutative and it has a one-element 0 such that $s + 0 = s$ for all $s \in [0,\infty)$. Finally, the distributive law holds as $\min(s+t, s+u) = s + \min(t,u)$ for all $s, t, u \in [0,\infty)$.

The semiring $(R, \oplus, \odot)$ is *idempotent* if for all $x \in R$, $x \oplus x = x$. For example, semirings in which the $\oplus$ operation is maximization or minimization are idempotent semirings. We also note that the Osd problem is defined in an idempotent semiring.

Let us return to formulating a permutation problem. A *permutation* $\sigma(M)$ of an m-element set M is a sequence $\sigma_1\sigma_2\cdots\sigma_m$, where $\sigma_i \in M$ and $\sigma_i \neq \sigma_j$ if $i \neq j$. When there is no ambiguity about the set M, we denote a permutation by σ. Sometimes it is convenient to represent a linear order as a permutation. A permutation σ and a linear order L on set M are in a one-to-one relationship if and only if it holds that $\sigma_i\sigma_j \in L$ if and only if $i \leq j$. We can clearly see that for every permutation there exists a corresponding linear order. Therefore, we use the terms permutation and linear order interchangeably and choose whichever is more convenient.

A permutation problem is defined in an arbitrary semiring. To this end, assume that the problem is defined in a semiring $(R, \oplus, \odot)$. We assume that the cost $c(\sigma)$ of a permutation σ decomposes into a product of n local costs:

$$c(\sigma) = \bigodot_{j=1}^{n} c\big(\{\sigma_1, \sigma_2, \ldots, \sigma_j\}, \sigma_{j-d+1}\cdots\sigma_{j-1}\sigma_j\big) .$$

The local cost of the jth elements depends on the sequence of d preceding elements and, possibly the set of all the j preceding elements. Here, if $d > j$ we read $\sigma_{j-d+1}\cdots\sigma_{j-1}\sigma_j$ as $\sigma_1\cdots\sigma_{j-1}\sigma_j$, and if $d = 0$ the sequence is void. A void sequence is denoted by $\emptyset$. We note that the local costs are specified by the problem input, which is some data on set N. The input can be, for example, a graph or a data matrix. In permutation problems, the task is to compute the sum

$$\bigoplus_{\sigma} c(\sigma),$$

where σ runs over the permutations of N. We note that permutation problems are so-called sum-product problems. For a constant $d \geq 0$, we call any problem of this form a *permutation problem of degree d*. We are especially interested in permutation problems, whose local cost function can be evaluated in polynomial time with respect to the size of the input. These problems are called *polynomial local time permutation problems.*

Next, we present some examples of polynomial local time permutation problems. A reader who is interested in permutation problems may refer to Fomin and Kratsch [35] though their definition of a permutation problem differs slightly from the one presented above.

The traveling salesman problem (TSP). The traveling salesman problem is a permutation problem of degree 2 in the min–sum semiring on real numbers, with $c(Y, v) = 0$ for $|Y| = 1$, and $c(Y, uv)$, for $|Y| > 1$, equaling the weight of edge uv in the input graph with vertex set N, indifferent of Y. Strictly speaking, this computes the weight of the minimum weight of the Hamiltonian paths, not cycles. This can be fixed by adding the weight of the edge from v to the starting node to local costs $c(N, v)$ for all nodes v.

The feedback arc set problem (FEEDBACK). Given a directed graph A, find the size (or weight) of the smallest (or lightest) arc set A' such that $A \setminus A'$ is acyclic. The feedback arc set problem is a permutation problem of degree 1 in the min–sum semiring, with $c(Y, v)$ equaling the number (or total weight) of arcs from $Y \setminus \{v\}$ to v in the input graph.

The cutwidth problem (CUTWIDTH). Given an undirected graph $G = (V, E)$ and a linear order L, the cutwidth of the linear order is the maximum number of arcs that a line inserted between two consecutive nodes cuts. The cutwidth of graph G is the minimum cutwidth over all possible linear orders. The cutwidth problem is a permutation problem of degree 0 in the min–max semiring, with $c(Y)$ equaling the number of edges with one endpoint in Y and another in $V \setminus Y$.

The treewidth problem (TREEWIDTH). A tree decomposition of an undirected graph $G = (V, E)$ is a pair (T, X) where $X = \{X_1, X_2, \ldots, X_k\}$ is a family of subsets of V and T is a tree whose nodes are the subsets X_i such that (i) $\bigcup_i X_i = V$, (ii) if $uv \in E$, then $u \in X_i$ and $v \in X_i$ for at least one i, and (iii) if $v \in X_i$ and $v \in X_j$, then v is also a member of every set along the path between X_i and X_j. The width of a tree decomposition is the size of the largest set X_i minus one. The treewidth of a graph G is the smallest width over all of its tree decompositions. The treewidth problem is a permutation problem of degree 1 in the min–max semiring, with $c(Y, v)$ equaling the number of vertices in $V \setminus Y$ that have a neighbor in the unique component of the induced subgraph $G[Y]$ containing v; see, for example, Bodlaender et al. [11].

The scheduling problem (Scheduling). Given n jobs $J_1, J_2, \ldots, J_n$ such that the execution of job J_i takes time t_i, there is an associated cost $d_i(t)$ for finishing job J_i at time t, the jobs are executed on a single machine and the execution of a job cannot be interrupted until it is finished, the task is to find the order of jobs that minimizes the total cost. The scheduling problem[5] is a permutation problem of degree 1 in the min–sum semiring, with cost $c(Y, v)$ equaling $d_v(t)$, where $t = \sum_{u \in Y} t_u$.

In all the presented problems, the local costs can be computed efficiently, that is, in polynomial time. However, finding the globally optimal permutation is more difficult. In summary, Tsp [41], Feedback [56], Cutwidth [42], Treewidth [4], and Scheduling [70] are all NP-hard.

It should be noted that the formulation of a polynomial local time permutation problem is quite loose and does not require that a permutation problem has anything to do with the cost of a permutation. Especially, it is possible to formulate a problem in a such way that one local cost function solves the problem and the others do nothing. Thus, all problems, which can be solved in polynomial time for example, can also be formulated as polynomial local time permutation problems. However, all time bounds that we will present are exponential and hence they are not particularly interesting for problems that are known to be solvable in polynomial time. The results hold, of course, for the "easier" permutation problems, too. Thus, we hope that whenever we consider polynomial local time permutation problems, the reader perceives them as problems such as Tsp, Feedback, Treewidth, Cutwidth, and Scheduling, which all have a natural interpretation for the cost of a permutation.

As the permutation problems operate in semirings, it is convenient to analyze their time and space requirements in terms of semiring operations. The time requirement is the total number of semiring additions and multiplications needed during the execution of an algorithm. The space requirement is the maximum number of values stored at any point during

[5]This problem is sometimes also called sequencing problem. This is not to be confused with the alternative name of the permutation problem.

execution. It is also convenient to assume that the local costs are either given or they can be computed in polynomial time and space using a black-box function whenever needed. This holds for all of the aforementioned problems. Next, we will formulate the OSD and FP problems as permutation problems. We will later see that they are permutation problems with a local cost, whose computation can take exponential time. We will also shortly discuss the ancestor relation problem although we do not know whether it can be efficiently formulated as a permutation problem.

2.2.1 The OSD Problem

Recall that in the OSD problem the goal is to maximize the sum

$$s(A) = \sum_{v \in N} s_v(A_v),$$

where A runs over the DAGs on N.

Next, we formulate the OSD problem as a permutation problem. We notice that due to the acyclicity every DAG has at least one node, called a *sink*, that has no outgoing arcs. Thus, the nodes of every DAG A can be topologically ordered, that is, if $uv \in A$, then u precedes v. In particular, an optimal DAG can be topologically ordered. Given a topological order L of A, node v is a sink of the subgraph induced by the set L_v. Thus, given a permutation (or linear order) σ, the score of an optimal DAG compatible with σ can be found by choosing for each node the best parents from its predecessors (excluding the node itself)[6]. To this end, define

$$\hat{s}_v(Y) = \max_{X \subseteq Y} s_v(X). \tag{2.2}$$

Intuitively, $\hat{s}_v(Y)$ is the highest local score when the parents of a node v are chosen from Y. Now, the score of an optimal DAG compatible with σ can be written as $s(\sigma) = \sum_{j=1}^{n} \hat{s}_{\sigma_j}\big(\{\sigma_1, \sigma_2, \ldots, \sigma_{j-1}\}\big)$. The score of an optimal DAG can be found by maximizing the sum of local scores over all orders, that is, by computing $\max_\sigma s(\sigma)$; in Section 2.3 we will show how to construct the DAG that maximizes the score. We observe the following.

Observation 2.6 *The* OSD *problem is a permutation problem of degree 1 in a max–sum semiring.*

[6]Note that by the definition of a linear order, every node precedes itself. However, due to the acyclicity constraint, a node cannot be its own parent.

2.2.2 The Fp Problem

In the Fp problem the goal is to compute the posterior probability of a modular feature f given data D, that is, $Pr(f|D)$. To this end, it is essential to compute the joint probability $Pr(f, D)$. We notice that

$$\begin{aligned} Pr(f, D) &= \sum_{A} f(A) Pr(D|A) Pr(A) \\ &= \sum_{A} \sum_{L \supseteq A} f(A) Pr(D|A) Pr(L, A) \\ &= \sum_{L} \sum_{A \subseteq L} \prod_{v \in N} f_v(A_v) Pr(D_v|D_{A_v}, A_v) \rho_v(L_v) q_v(A_v) \\ &= \sum_{L} \prod_{v \in N} \rho_v(L_v) \sum_{A_v \subseteq L_v} f_v(A_v) Pr(D_v|D_{A_v}, A_v) q_v(A_v). \end{aligned} \tag{2.3}$$

Here, the first equality is the equation (2.1), the second equality holds by the definition of an order-modular prior and by the fact that an order-modular prior vanishes when the DAG is not compatible with the order, the third one by the decomposability of the likelihood, the modularity of the feature and the prior, and the commutativeness of the addition, and the fourth one by distributiveness of multiplication over addition.

Now, let us define a local score β for each v and A_v as

$$\beta_v(A_v) = f_v(A_v) Pr(D_v|D_{A_v}, A_v) q_v(A_v).$$

Further, for each v and set L_v we define a sum

$$\alpha_v(L_v) = \rho_v(L_v) \sum_{A_v \subseteq L_v} \beta_v(A_v).$$

The sum on the right is known as the *zeta transform*[7] of β_v. We will return to the computation of the zeta transform in Section 2.3.2. Now, we are ready to express the $Pr(f, D)$ as

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

where $\alpha_v(L_v)$ depends only on the set of preceding nodes. To finalize the computation of $Pr(f|D)$, we notice that the probability $Pr(D)$ can be computed like $Pr(f, D)$ but the feature f is a trivial indicator function that evaluates everywhere as 1. Based on the equation (2.4), we observe the following.

[7] In some papers the zeta transform is referred to as the Möbius transform. In this thesis, however, we use the terminology by Rota [90].

Observation 2.7 *The* Fp *problem is a permutation problem of degree 1 in a sum–product semiring.*

2.2.3 The Ancestor Relation Problem

Unlike the Osd and Fp problems, we are not aware of any efficient way to formulate the ancestor relation problem as a permutation problem, even with an order-modular prior. The difference compared to Osd and Fp is the fact that the indicator function for an ancestor relation, which is a non-modular feature, cannot be factorized as simply as the indicator function of a modular feature.

2.3 Dynamic Programming

Next, we present a standard way to solve a permutation problem. A naïve algorithm would go through all different permutations one by one and thus taking time proportional to $n!$. However, we can do much better than that: a dynamic programming algorithm that goes through all 2^n node subsets will do. Such algorithms have been known for decades for permutation problems such as Tsp [6, 50], Feedback [69] and Treewidth [4] and recently such dynamic programming techniques have been shown to work also for Osd [78, 96, 98] and Fp [65].

Let us consider TSP on cities N. Let $w(u, v)$ be the distance from a city u to a city v. Without any loss of generality, we can choose a city $s \in N$ to be the starting point of the route. We define $g(Y, v)$ for all $Y \subseteq N \setminus \{s\}$ and $v \in Y$ to be the length of the shortest route from s to v via all cities in Y. Now $g(Y, v)$ can be computed by dynamic programming according to the following recurrences:

$$\begin{aligned}
g(Y,v) &= w(s,v) \qquad \text{if } |Y| = 1,\\
g(Y,v) &= \min_{u \in Y\setminus\{v\}} \Big\{ g(Y \setminus \{v\}, u) + w(u,v) \Big\} \qquad \text{if } 1 < |Y| < n-1,\\
g(Y,v) &= \min_{u \in Y\setminus\{v\}} \Big\{ g(Y \setminus \{v\}, u) + w(u,v) + w(v,s) \Big\} \quad \text{if } |Y| = n-1.
\end{aligned}$$

The first equation clearly holds for routes with one stage. Then we start building routes with more stages. For every set Y and a city v we find a city u, the second last city in the route, that minimizes the total route length (the second equation). Finally, in the third equation we add the distance from the last city to the starting point. The length of the shortest tour is the minimum of $g(N, v)$ over $v \in N$.

The algorithm computes and stores the route lengths for each subsets Y and a city $v \in Y$, that is, for $O(n2^n)$ pairs (Y, v). For each pair the computation takes $O(n)$ time. Thus, the total time and space requirements of the algorithm are $O(n^2 2^n)$ and $O(n2^n)$, respectively.

The dynamic programming algorithm can be generalized to permutation problems in general. Let $\sigma(Y)$ denote a permutation of elements in the set Y. In what follows we use the shorthand $l = |Y|$. For any $Y \subseteq N$ and distinct elements $v_1, v_2, \ldots, v_{d-1} \in Y$ we define $g(Y, v_{d-1} \cdots v_2 v_1)$ as

$$g(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{\substack{\sigma(Y) \\ \sigma_{l-d+2} \cdots \sigma_{l-1} \sigma_l = v_{d-1} \cdots v_2 v_1}} \bigodot_{j=1}^{l} c(\{\sigma_1, \sigma_2, \ldots, \sigma_j\}, \sigma_{j-d+1} \cdots \sigma_{j-1} \sigma_j). \quad (2.5)$$

Note that the summation is over all permutations of the elements in Y ending with $v_{d-1} \cdots v_2 v_1$. Further, the sum of $c(\sigma)$ over all permutations σ equals

$$\bigoplus_{\substack{v_1, v_2, \ldots, v_{d-1} \in N \\ v_i \neq v_j \text{ if } i \neq j}} g(N, v_{d-1} \cdots v_2 v_1).$$

Because multiplication distributes over addition in a semiring, we get the following dynamic programming equations. For permutation problems of degree 0, we have

$$\begin{aligned} g(\emptyset) &= 1 \\ g(Y) &= \bigoplus_{u \in Y} g(Y \setminus \{u\}) \odot c(Y) \quad \text{for } l \geq 1. \end{aligned}$$

A straightforward dynamic programming algorithm computes the sum in $O(n2^n)$ time and $O(2^n)$ space.

For permutation problems of degree $d > 0$, we have

$$\begin{aligned} g(\emptyset, \emptyset) &= 1, \\ g(Y, v_{d-1} \cdots v_2 v_1) &= \bigoplus_{v_d \in Y \setminus \{v_1, v_2, \ldots, v_{d-1}\}} g(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_2) \\ &\quad \odot c(Y, v_d v_{d-1} \cdots v_2 v_1) \quad \text{for } l \geq 1, \end{aligned}$$

where the latter recurrence is computed for all nonempty $Y \subseteq N$ and $v_i \in Y$ with $v_i \neq v_j$ if $i \neq j$. This yields a straightforward dynamic programming algorithm that computes $g(N, v_{d-1} \cdots v_2 v_1)$ for all $v_1, v_2, \ldots, v_{d-1} \in N$ with $v_i \neq v_j$ if $i \neq j$ and hence the sum of $c(\sigma)$ over all permutations on N, in $O^*(2^n)$ time and space.

The polynomial factors in the time requirement depend on two things: the degree of the permutation problem and the time requirement of computing the local costs. For each set Y, there is at most $\binom{l}{d-1}(d-1)! = O(l^{d-1})$ different sequences $v_{d-1} \cdots v_2 v_1$ and thus $g(Y, v_{d-1} \cdots v_2 v_1)$ is computed and stored for at most $O(n^{d-1})$ times per set. As the computation of $g(Y, v_{d-1} \cdots v_2 v_1)$ for a given set and sequence takes $O(n)$ time, the polynomial factor for a permutation problem of degree $d > 0$, while counting semiring operations, is $O(n^d)$ for time and $O(n^{d-1})$ for space. The following theorems summarize the bounds for time and space.

Theorem 2.8 *Permutation problems of degree d can be solved in $O(n^{d'}2^n)$ time and $O(n^{d'-1}2^n)$ space, where $d' = \max\{1, d\}$, assuming that the local costs are precomputed and can be evaluated in constant time.*

Theorem 2.9 *Polynomial local time permutation problems of degree d, where d is a constant, can be solved in $O^*(2^n)$ time and space.*

These bounds are the best we know for most of the classical permutation problems we presented in Section 2.2. As an exception, Fomin and Villanger [36] have presented an algorithm for TREEWIDTH that runs in $O^*(1.7549^n)$ time and space.

Next, we will consider the OSD and FP problems. Here, the difference compared to the polynomial local time permutation problems is that the computation of local costs can take up to exponential time. We will show, however, that compared to the polynomial local time permutation problems the computation of local costs in OSD and FP increases the time requirement only by a factor linear in n.

2.3.1 The OSD Problem

The dynamic programming algorithm for permutation problems in general applies to the OSD problem: we go through all the node sets tabulating the intermediate results for the sets of the first i nodes of the order, for $i = 0, 1, \ldots, n$. Let $\hat{s}_v(Y)$ be the score of the best parent set of a node v when its parents have to be chosen from the set $Y \subseteq N \setminus \{v\}$; see the equation (2.2). We define $g(\emptyset) = 0$ and for nonempty sets $Y \subseteq N$ recursively

$$g(Y) = \max_{v \in Y} \Big\{ g(Y \setminus \{v\}) + \hat{s}_v(Y \setminus \{v\}) \Big\}. \tag{2.6}$$

In words, $g(Y \setminus \{v\}) + \hat{s}_v(Y \setminus \{v\})$ is the maximum score of the DAGs on a node set Y when a node v is the sink, that is, the parents of v are selected from $Y \setminus \{v\}$.

We notice that the naïve computation of $\hat{s}_v(Y)$ for a fixed Y and v requires $O(2^{|Y|})$ basic operations. Thus, the total number of basic operations for computing $\hat{s}_v(Y)$ for all Y is proportional to $n \sum_{k=0}^{n-1} \binom{n-1}{k} 2^k = n3^{n-1}$ (the equality holds by the binomial theorem; see the equation (A.1) in Appendix A), which is larger than the $O^*(2^n)$ required by the dynamic programming algorithm given precomputed local scores. However, the following observation, which states that the best parent set of v chosen from Y is either Y itself or the best parent set chosen from $Y \setminus \{u\}$ for some $u \in Y$, helps us to reduce the time requirement.

Lemma 2.10 (Ott and Miyano [79]) *If $v \in N$ and $Y \subseteq N \setminus \{v\}$, then*

$$\hat{s}_v(Y) = \max \Big\{ s_v(Y), \max_{u \in Y} \hat{s}_v(Y \setminus \{u\}) \Big\}.$$

Proof. By the definition of $\hat{s}_v(Y)$, $\max_{u \in Y} \hat{s}_v(Y \setminus \{u\})$ equals the maximum over the union of $2^{Y \setminus \{v\}}$ where v runs over the elements in Y. We observe that now $\hat{s}_v(Y)$ equals the maximum over the union Y and $\bigcup_{v \in Y} 2^{Y \setminus \{v\}}$, which is the maximum over 2^Y. This, for one, is the definition of $\hat{s}_v(Y)$. $\square$

Using this recurrence[8], the computation of $\hat{s}_v(Y)$ for a fixed v and Y takes no more than n comparisons, yielding the total time requirement $O(n^2 2^n)$ and the total space requirement $O(n2^n)$ for computing $\hat{s}_v(Y)$ for all v and Y.

Assuming that the values $\hat{s}_v(Y)$ have been computed and stored, the value $g(N)$ can be computed in $O(n2^n)$ time and $O(2^n)$ space. Thereby, the total time and space requirements of the algorithm are $O(n^2 2^n)$ and $O(n2^n)$, respectively. This is almost optimal since the input, that is, the local scores, might already contain $n2^{n-1}$ values.

If all the intermediate results are kept in memory, the space requirement of the previous algorithm is $O(n2^n)$. However, we notice that the computation of $g(Y)$ and $\hat{s}_v(Y)$ requires results only from the sets of size $|Y| - 1$. Therefore, we can compute both functions simultaneously, and proceed levelwise, that is, increasing cardinality of Y. While computing scores at level ℓ we need to keep values for only $O\Big(\binom{n}{\ell} + \binom{n}{\ell-1}\Big)$ sets in memory. This reduces the space requirement to $O(\sqrt{n}2^n)$ [6, 79].

[8]Another way to compute the values of $\hat{s}_v$ would be to use a generalized version of Yates's algorithm; see, for example, Koivisto [61] and Koivisto and Sood [65].

Above, we computed the score of an optimal DAG. Next, we show how to find the DAG itself. The construction of an optimal DAG is quite straightforward; see Algorithm 1. In algorithm descriptions, we use the bracket notation to emphasize the program variables that are used while running the algorithm, that is, for example $\hat{s}_v[Y]$ corresponds to the target value $\hat{s}_v(Y)$. We also assume that the arg max operation returns a single arbitrary element that maximizes the expression in question. First, when computing $\hat{s}_v[Y]$ for sets $Y \subseteq N \setminus \{v\}$, for every v and Y we keep track of the best parent set of v when the parents are chosen from Y, that is, $bps_v[Y \setminus \{v\}] = \arg\max_{X \subseteq Y} s_v(X)$. Further, while computing the recurrence (2.6) we keep track of the sink nodes. Once we have found the score of an optimal network we construct the corresponding DAG A by backtracking. First, we initialize the DAG A on node set N to be an empty graph. We start by setting $Y \leftarrow N$ and find the sink v of Y. Then we assign the parent set $bps_v[Y \setminus \{v\}]$ to node v in graph A. Then we move the set $Y \leftarrow N \setminus \{v\}$ and find its set and the corresponding best parent set. We continue this kind of procedure until we reach the empty set. At this point, the DAG A is an optimal DAG.

Algorithm 1 Dynamic programming for Osd.

```
Input: Local scores $s_v(Y)$ for all $v \in N$ and $Y \subseteq N \setminus \{v\}$.
Output: An optimal DAG $A$.
 1: for all $Y \subseteq N$ in increasing cardinality do
 2:     for all $v \in Y$ do
 3:         $\hat{s}_v[Y] \leftarrow \max\Big\{s_v(Y), \max_{u \in Y} \hat{s}_v[Y \setminus \{u\}]\Big\}$.
 4:         $bps_v[Y] \leftarrow \arg\max\Big\{s_v(Y), \max_{u \in Y} \hat{s}_v[Y \setminus \{u\}]\Big\}$.
 5:     end for
 6:     $g[Y] \leftarrow \max_{v \in Y}\Big\{g[Y \setminus \{v\}] + \hat{s}_v[Y \setminus \{v\}]\Big\}$.
 7:     $sink[Y] \leftarrow \arg\max_{v \in Y}\Big\{g[Y \setminus \{v\}] + \hat{s}_v[Y \setminus \{v\}]\Big\}$.
 8: end for
 9: $A$ is an empty graph.
10: $Y \leftarrow N$
11: while $Y \neq \emptyset$ do
12:     $v \leftarrow sink[Y]$
13:     Add arcs from $bps_v[Y \setminus \{v\}]$ to $v$ to $A$.
14:     $Y \leftarrow Y \setminus \{v\}$.
15: end while
16: return $A$
```

2.3.2 The FP Problem

The probability $Pr(f, D)$ given the values of $\alpha_v(Y)$ can be computed straightforwardly by dynamic programming in similar fashion as for OSD: we define the recurrence $g(\emptyset) = 1$ and

$$g(Y) = \sum_{v \in Y} g(Y \setminus \{v\})\alpha_v(Y \setminus \{v\}) \tag{2.7}$$

for every nonempty set $Y \subseteq N$. Finally, $g(N)$ equals $Pr(f, D)$. Here, g is called a *forward function* as it computes the contribution of the set Y to the target probability assuming that Y are the first $|Y|$ nodes in the order. Later in Section 4.4.2 we will introduce a related backward function.

The difficulty lies in the computation of $\alpha_v(Y)$. As it is a sum over all subsets of Y, the computation for fixed v and Y takes $O(2^{|Y|})$ time and the total time for all Y is $O(n3^n)$. However, this can be computed faster as we notice that α_v is essentially the zeta transform of β_v and there is a known fast algorithm for zeta transform; next, we will discuss the algorithm.

Let h be a function from the subsets of an n-element set N to the real numbers. The *zeta transform* of h is another function $(h\zeta)$ defined by

$$(h\zeta)(Y) = \sum_{X \subseteq Y} h(X), \quad Y \subseteq N.$$

Next, we will show that given h, the zeta transform for all $Y \subseteq N$ can be computed in $O(n2^n)$ time and $O(2^n)$ space. The *fast zeta transform* algorithm is based on an idea presented by Yates in 1937 [111], restated by Knuth [59] and later rediscovered at least by Kennes and Smets [57, 58]. The fast zeta transform has been used to develop faster algorithms for several combinatorial problems such as graph coloring [63] and Steiner tree [76] and combinatorial tools like the fast subset convolution [9].

Algorithm 2 computes the zeta transform over N. Here we assume, without any loss of generality that $N = \{1, 2, \ldots, n\}$. The idea of the algorithm is to conduct the "big" zeta transform in n phases doing one "little" transform at a time.

Theorem 2.11 *Algorithm 2 works correctly.*

Proof. We show by induction on j that the computed function satisfies

$$h_j(Y) = \sum_{X \in (Y)_j} h(X),$$

Algorithm 2 The fast zeta transform.

```
Input: $h(X)$ for $X \subseteq N$.
Output: $(h\zeta)(Y)$ for $Y \subseteq N$.
 1: for $Y \subseteq N$ do
 2:   Let $h_0(Y) \leftarrow h(Y)$.
 3: end for
 4: for $j \leftarrow 1, \ldots, n$ do
 5:   for $Y \subseteq N$ do
 6:     if $j \in Y$ then
 7:       Let $h_j(Y) \leftarrow h_{j-1}(Y) + h_{j-1}(Y \setminus \{j\})$.
 8:     else
 9:       Let $h_j(Y) \leftarrow h_{j-1}(Y)$.
10:     end if
11:   end for
12: end for
13: return Zeta transforms $h_n(Y)$ for all $Y \subseteq N$.
```

where we use the shorthand

$$(Y)_j = \{X \subseteq Y : Y \cap \{j+1, j+2, \ldots, n\} \subseteq X\};$$

This will suffice, since $(Y)_n$ consists of the subsets of Y.

We proceed by induction on j and the size of the set Y. (i) Trivially $(Y)_0 = \{Y\}$. Therefore, $h_0(Y) = h(Y) = \sum_{X \in (Y)_0} h(X)$, as claimed. (ii) Assume then that $h_{j-1}(Y) = \sum_{X \in (Y)_{j-1}} h(X)$.

First, consider the case $j \notin Y$. Now $(Y)_j = (Y)_{j-1}$ because $Y \cap \{j+1, j+2, \ldots, n\} = Y \cap \{j, j+1, \ldots, n\}$. Thus $h_j(Y) = h_{j-1}(Y)$, as correctly computed at line 9.

Second, consider the case $j \in Y$. To prove the correctness of line 7, it suffices, by the induction assumption, to show that $(Y)_{j-1}$ and $(Y \setminus \{j\})_{j-1}$ are disjoint and their union is $(Y)_j$. To this end, it suffices to observe that $(Y)_{j-1}$ consists of all subsets $X \supseteq Y \cap \{j, j+1, \ldots, n\}$ of Y that do contain j, whereas $(Y \setminus \{j\})_{j-1}$ consists of all subsets $X \supseteq Y \cap \{j, j+1, \ldots, n\}$ of Y that do not contain j. □

The following example illustrates the fast zeta transform.

Example 3. The fast zeta transform of a function $g(Y)$ for subsets of $N = \{1, 2, 3\}$ is illustrated in Figure 2.2. Each column corresponds to a subset of N and the boxes correspond to the values of h_j. The first row corresponds to the initialization of the values $h_0(Y)$ to $g(Y)$ and the other rows to the computations of h_j for $j = 1, 2, 3$. At each phase the value of h_j for a set Y is obtained by summing the values that correspond to the tails of the arcs pointing to $h_j(Y)$. The bottom line contains the values $h_n(Y)$, that is, the zeta transform of g at Y. Figure 2.2 shows that there is exactly one directed path from each subset of Y to Y, that is, all subsets contribute to the sum exactly once. Also there is no directed path to Y from any set that is not a subset of Y, thus only the subsets of Y contribute to the sum. $\diamond$

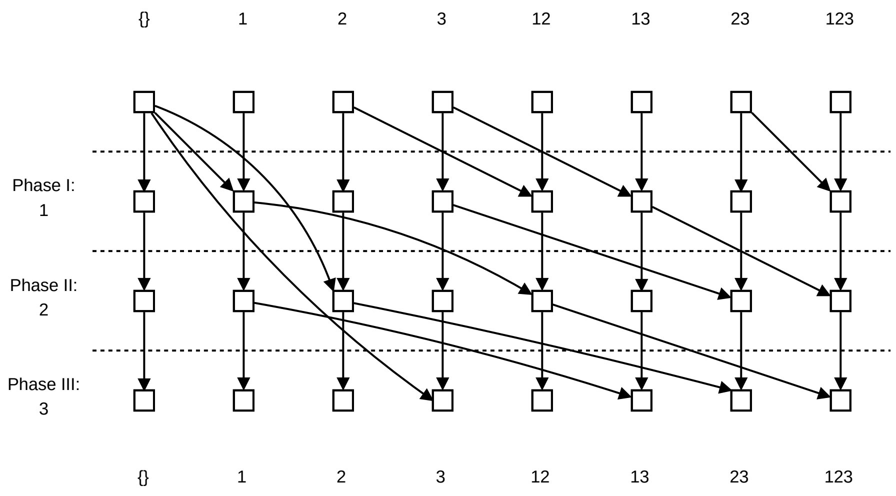


Figure 2.2: The fast zeta transform on the set $N = \{1, 2, 3\}$.

It should be noted that the recurrence (2.7) applies only when we have an order-modular prior for the structure. There are dynamic programming algorithms that compute posterior probabilities of structural features with uniform prior over DAGs [55, 105] but their time requirement is $O^*(3^n)$. The extra time required is due to the fact that the algorithms use inclusion-exclusion to prevent the same DAG from contributing to the sum more than once.

Chapter 3

Space–Time Tradeoffs

We continue with an introduction to space–time tradeoffs in learning the structure of Bayesian networks. First, we introduce two simple schemes, namely the two-bucket scheme and the divide-and-conquer scheme, for permutation problems in general. Then, we adapt these schemes for OSD and FP. We end this chapter by defining the notion "optimality" of a scheme in terms of the time–space product.

3.1 Permutation Problems

In this section, we introduce two schemes to solve permutation problems with less space. Both schemes work under an assumption that the local costs can be evaluated in polynomial time. These schemes are later in Section 3.2 adapted for structure discovery problems, where the evaluation of local costs can take exponential time.

3.1.1 Two-Bucket Scheme

The two-bucket scheme is a simple scheme to solve permutation problems in less than $O^*(2^n)$ space. The scheme is based on an idea to guess the s first elements of the linear order and solve the subproblems for the first s and the last $n - s$ elements independently. More formally, this approach is justified by the following observation. Fix an integer $n/2 \leq s \leq n$. Now, there exists a partition of N to N_0 and N_1 such that N_0 are the first s elements in a linear order and N_1 are the last $n - s$ elements. For simplicity, we assume that $n - s \geq d$, where d is the degree of the permutation problem in question. We use a shorthand $l = |Y|$. We also assume that whenever the notation $v_1, \ldots, v_d \in X$ is used, $v_i \neq v_j$ if $i \neq j$. Based on the observation, one can solve a permutation problem by trying out all possible partitions $\{N_0, N_1\}$ and solving the recurrences $g_0(\emptyset, \emptyset) = 1$,

$$g_0(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{v_d \in Y \setminus \{v_1, \ldots, v_{d-1}\}} \Big\{ g_0(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_2) \odot c(Y, v_d v_{d-1} \cdots v_2 v_1) \Big\}, \tag{3.1}$$

for $\emptyset \subset Y \subseteq N_0$, $v_1, v_2, \ldots, v_{d-1} \in Y$,

$$g_1(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{v_d \in N_0 \setminus \{v_2, \ldots, v_{d-1}\}} \Big\{ g_0(N_0, v_d v_{d-1} \cdots v_2) \odot c(N_0 \cup Y, v_d v_{d-1} \cdots v_2 v_1) \Big\}, \tag{3.2}$$

for $Y \subseteq N_1$, $l = 1$, $v_1 \in Y$, $v_2, v_3, \ldots, v_{d-1} \in N_0$,

$$g_1(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{v_d \in N_0 \setminus \{v_{l+1}, \ldots, v_{d-1}\}} \Big\{ g_1(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_2) \odot c(N_0 \cup Y, v_d v_{d-1} \cdots v_2 v_1) \Big\}, \tag{3.3}$$

for $Y \subseteq N_1$, $1 < l < d$, $v_{l+1}, \ldots, v_{d-1} \in N_0$, $v_1, \ldots, v_l \in Y$ and

$$g_1(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{v_d \in Y \setminus \{v_1, \ldots, v_{d-1}\}} \Big\{ g_1(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_2) \odot c(N_0 \cup Y, v_d v_{d-1} \cdots v_2 v_1) \Big\}, \tag{3.4}$$

for $Y \subseteq N_1$, $l \geq d$, and $v_1, v_2, \ldots, v_{d-1} \in N_1$; the total cost is obtained as the sum $g(N) = \oplus_{v_1, \ldots, v_{d-1} \in N_1} g'_1(N_1, v_{d-1} \cdots v_2 v_1)$, where $g'_1(N_1, v_{d-1} \cdots v_2 v_1)$ is the sum of $g_1(N_1, v_{d-1} \cdots v_2 v_1)$ over all partitions $\{N_0, N_1\}$. Here, the recurrence g_0 is of the same form as standard dynamic programming. The recurrence g_1, however, differs from the standard form. In the recurrences (3.2) and (3.3) part of the sequence $v_{d-1} \cdots v_2 v_1$ consists of items from the set N_0. The recurrence g_1 does not have a value for an empty set; instead in the recurrence (3.2) one takes the values from a suitable $g_0(N_0, v_{d-1} \cdots v_2 v_1)$.

Let us analyze the time and space requirement of the algorithm. We notice that the two subproblems are independent given the partition $\{N_0, N_1\}$, and thus can be solved separately. Assuming the local costs can be computed in polynomial time, applying the exact dynamic programming algorithm from Section 2.3 solves the first subproblem, that is, computes g_0

in $O^*(2^s)$ time and space. Similarly, the second subproblem, that is, the values of g_1 are computed in $O^*(2^{n-s})$ time and space. There are $\binom{n}{s}$ different partitions $\{N_0, N_1\}$ and thus in total the problem can be solved in $O^*(\binom{n}{s}2^s)$ time and $O^*(2^s)$ space. We get the following bounds for time and space.

Theorem 3.1 *Polynomial local time permutation problems of bounded degree can be solved in $O^*(\binom{n}{s}2^s)$ time and $O^*(2^s)$ space.*

Next, we present some examples of these bounds for different values of s. For example, setting $s = (4/5)n$ yields $O^*(2.872^n)$ time and $O^*(1.742^n)$ space. On the other hand, setting $s = (3/5)n$ yields $O^*(2.972^n)$ time and $O^*(1.516^n)$ space.

3.1.2 Divide and Conquer Scheme

We further develop the partitioning idea from the previous section by applying it recursively. This scheme is based on a divide-and-conquer approach introduced by Savitch [91]. The approach has later been applied to TSP [8, 46] and other problems [11, 108]. In what follows, we loosely follow the presentation by Fomin and Kratsch [35] although we present the divide-and-conquer scheme in a far more general form.

Here the idea is to create subproblems using the partitioning idea of the two-bucket scheme and solve the subproblems by applying the partitioning idea again. Assume that we have a partition $\{N_0, N_1\}$ where all elements in N_0 precede all elements in N_1. Now N_0 can be partitioned to $\{N_{00}, N_{01}\}$ and N_1 to $\{N_{10}, N_{11}\}$ such that all elements in N_{00} precede all elements in N_{01} and all elements in N_{10} precede all elements in N_{11}. We define function $g(N_1, N_0, w_{d-1} \cdots w_2 w_1, v_{d-1} \cdots v_2 v_1)$ to be the score over all permutations on N_1 ending with $v_{d-1} \cdots v_2 v_1$ given that exactly the elements of N_0 precede N_1 and $w_{d-1} \cdots w_2 w_1$ is the sequence of $d-1$ last nodes in N_0. The function $g(N_1, N_0, w_{d-1} \cdots w_2 w_1, v_{d-1} \cdots v_2 v_1)$ equals to the sum of

$$\bigoplus_{x_1, x_2, \ldots, x_{d-1} \in N_{10}} g(N_{10}, N_0, w_1 w_2 \cdots w_{d-1}, x_{d-1} \cdots x_2 x_1) \\ \odot g(N_{11}, N_0 \cup N_{10}, x_{d-1} \cdots x_2 x_1, v_{d-1} \cdots v_2 v_1)$$

over all partitions $\{N_{10}, N_{11}\}$. In this section, assume that whenever the notation $v_1, \ldots, v_l \in X$ is used, $v_i \neq v_j$ if $i \neq j$; we also use a shorthand $l = |Y|$. The permutation problem is solved by computing $\bigoplus_{v_1, v_2, \ldots, v_{d-1} \in N} g(N, \emptyset, \emptyset, v_{d-1} \cdots v_2 v_1)$. In general one can apply the recursion to depth t and then solve the subproblems by dynamic programming. The value $g(N_1, N_0, w_{d-1} \cdots w_2 w_1, v_{d-1} \cdots v_2 v1)$ can be computed by dynamic programming recurrence $g'(\emptyset, \emptyset) = 1$ and

$$\begin{aligned} g'(Y, v_{d-1} \cdots v_2 v_1) &= g'(Y \setminus \{v_1\}, v_l \cdots v_3 v_2) \\ &\quad \odot c(Y \cup N_0, w_{d-l-1} \cdots w_2 w_1 v_l \cdots v_2 v_1) \end{aligned}$$

for nonempty $Y \subseteq N_1$ with $l < d$ and $v_1, \ldots, v_l \in Y$ and

$$\begin{aligned} g'(Y, v_{d-1} \cdots v_2 v_1) &= \bigoplus_{v_d \in Y \setminus \{v_1, v_2, \ldots, v_{d-1}\}} g'(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_3 v_2) \\ &\quad \odot c(Y \cup N_0, v_d v_{d-1} \cdots v_2 v_1) \end{aligned}$$

for $Y \subseteq N_1$ with $l \geq d$ and $v_1, v_2, \ldots, v_{d+1} \in Y$. The value $g'(N_1, v_{d-1} \cdots v_2 v_1)$ equals to $g(N_1, N_0, w_{d-1} \cdots w_2 w_1, v_{d-1} \cdots v_2 v_1)$.

For an analysis of the time and space requirements, it is convenient to assume a balanced scheme: in every step of the recursion, the element set in question is partitioned into two sets of about equal size. Further, for simplicity we assume that the number of elements, n, is a power of 2. Then, at depth $t \geq 0$ of the recursion, the element set of each subproblem is of size $s = n/2^t$. The value of g can be computed using the recurrence g' in $O^*(2^s)$ time and space. For every set of size s, there is at most $\binom{s}{s/2} \leq 2^{s-1}$ possible partitions (the inequality holds by a well-known bound; see the inequality (A.2) in Appendix A). For each partition of size s, the recurrence $g(N_1, N_0, w_{d-1} \cdots w_2 w_1, v_{d-1} \cdots v_2 v1)$ is called at most $2\binom{s}{d-1}(d-1)! \leq 2s^{d-1}(d-1)!$ times. Thus, the recurrences are called at most $n^{d-1}(d-1)!2^n(n/2)^{d-1}(d-1)!2^{n/2}(n/4)^{d-1}(d-1)!2^{n/4} \cdots (2s)^{d-1}(d-1)!2^{2s} = (n^t/2^t)(d-1)!^t 2^{2n-2s}$ times. As the recursion depth is at most $\log n$, we get the following.

Theorem 3.2 *Polynomial local time permutation problems of degree d can be solved in $O^*(2^{2n-s} n^{(d-1)(\log n - \log s)}(d-1)!^{\log n - \log s})$ time and $O^*(2^s)$ space for any $s = n/2, n/4, n/8, \ldots, 1$.*

As a theoretically interesting special case with a polynomial space requirement we have the following.

Corollary 3.3 *Polynomial local time permutation problems of degree d can be solved in $O^*(4^n n^{(d-1)\log n}(d-1)!^{\log n})$ time and polynomial space.*

3.2 Structure Discovery in Bayesian Networks

So far, we assumed that the local costs can always be evaluated in polynomial time. This is not, however, the case with the Osd and Fp problems.

Luckily, it turns out that one can, under some assumptions about possible parent sets, overcome the complications caused by the computation of the local score and solve both OSD and FP using the two-bucket and divide-and-conquer schemes. To this end, it is essential to be precise about the input of these problems. Therefore, we first discuss the handling of the parent set with limited space.

3.2.1 On Handling Parent Sets with Limited Space

For a rigorous treatment of the structure discovery problems with limited space, we need to be explicit about the input of the problem, that is, the local scores $s_v(A_v)$ for OSD and $\beta_v(A_v)$ for FP. It is convenient to assume that the scores are nonzero (in the corresponding semiring) only for a family of *possible parent sets* denoted by $\mathcal{F}_v$; elsewhere we define that $s_v(A_v) = -\infty$ and $\beta_v(A_v) = 0$. Note that the zero scores are a direct consequence of setting the structure prior to zero when $A_v \notin \mathcal{F}_v$ as we did in Section 2.1.1.

The space requirement of the structure discovery problems is affected by the fact whether the input is represented explicitly or implicitly. In the former case, we assume that the input consists of a list of tuples $(v, A_v, s_v(A_v))$ or $(v, A_v, \beta_v(A_v))$. This enables finding the local score for a given pair (v, A_v) in $O(n)$ time[1]. However, the size of the input is $\sum_v |\mathcal{F}_v|$, which is going to be a lower bound of the space requirement for our algorithms. In the latter case, we assume that data is kept in memory as a $n \times m$ matrix, where n is the number of nodes and m is the number of observations, and the local scores are computed anew in $\delta(n)$ time whenever needed. Although it is often reasonable to assume that $\delta(n)$ is polynomial in n, this approach is usually slow in practice and therefore mainly of theoretical interest. It should be noticed that although we present $\delta(n)$ as a function of n it actually depends also on the number of samples in the data, m. In practice, m can be fairly large compared to n and cause the constant factor to be large as well. In this thesis we concentrate on the explicit input but consider the implicit input, too.

We are particularly interested in possible parent set families $\mathcal{F}_v$ that are *downward-closed*, that is, closed with respect to inclusion: if $Y \in \mathcal{F}_v$ and $X \subseteq Y$ then $X \in \mathcal{F}_v$. Natural examples of such families in Bayesian network context are (a) sets of at most size k for a fixed k and (b) sets that are subsets of given candidate parents [39, 86]. A useful property

[1]If the local scores are stored in n arrays (one for each $v \in N$) in lexicographic order, the set A_v can be found using binary search in $O(\log |\mathcal{F}_v|)$ time. As $|\mathcal{F}_v| \leq 2^{n-1}$, $\log |\mathcal{F}_v| \leq \log 2^{n-1} = (n-1)\log 2 = O(n)$.

of the downward-closed set families is that their members can be listed in (nearly) linear time. We will later make use of the following, slightly stronger observation. Define a set *interval* from X to Y as $[X, Y] = \{Z \subseteq Y : X \subseteq Z\}$.

Proposition 3.4 *Given a set N, a downward closed family $\mathcal{F} \subseteq 2^N$, whose membership can be determined in $\tau(n)$ time, and sets X and Y with $X \subseteq Y \subseteq N$, the members of $\mathcal{F} \cap [X, Y]$ can be listed in $O(|\mathcal{F} \cap [X, Y]| n\tau(n))$ time.*

Proof. If $X \notin \mathcal{F}$, then none of its superset is in $\mathcal{F}$. Otherwise, we can list the members of $\mathcal{F}$ in increasing cardinality. Let $\mathcal{F}_\ell$ consist of the sets $Z \in \mathcal{F}$ satisfying $X \subseteq Z \subseteq Y$ and $|Z| = \ell$. We go through all $Z \in \mathcal{F}_\ell$ in lexicographic order and for every $u \in Y \setminus Z$ such that u is larger than the maximal element of $Z \setminus X$ (in lexicographic order) we test whether $Z \cup \{u\}$ is a member of $\mathcal{F}$, and add it to $\mathcal{F}_{\ell+1}$ if it satisfies the condition. We notice that no member of $\mathcal{F}$ is listed more than once, and for every member of $\mathcal{F}$, the membership in $\mathcal{F}$ is tested no more than $n - 1$ times. Therefore, the total time requirement is $O(|\mathcal{F} \cap [X, Y]| n\tau(n))$. □

If $\mathcal{F}$ consists of the parent sets of cardinality of most k and we proceed levelwise as in the proof of Proposition 3.4, we always know the cardinality of sets in each level. In other words, the membership in $\mathcal{F}$ can be determined in constant time, which implies the following.

Proposition 3.5 *Given a set N, a family $\mathcal{F} \subseteq 2^N$ that consists of all sets of cardinality at most k, and sets X and Y with $X \subseteq Y \subseteq N$, the members of $\mathcal{F} \cap [X, Y]$ can be listed in $O(|\mathcal{F} \cap [X, Y]| n)$ time.*

3.2.2 The Osd Problem

We are ready to adapt the two-bucket scheme and the divide-and-conquer scheme for the Osd problem. The recurrences are mostly in the same form as for the permutation problems in general but the computation of the local scores differs. Also, here it is convenient to formulate the two-bucket scheme with two completely independent recurrences. Based on the recurrences (3.1) and (3.2), one can find the score of an optimal DAG $\hat{A}$ by trying out all possible partitions $\{N_0, N_1\}$ and solving the recurrences

$$g_0(Y) = \max_{v \in Y} \Big\{ g_0(Y \setminus \{v\}) + \hat{s}_v(Y \setminus \{v\}) \Big\}, \tag{3.5}$$

for $\emptyset \subset Y \subseteq N_0$ with $g_0(\emptyset) = 0$, and

$$g_1(Y) = \max_{v \in Y} \Big\{ g_1(Y \setminus \{v\}) + \hat{s}_v(N_0 \cup Y \setminus \{v\}) \Big\}, \tag{3.6}$$

for $\emptyset \subset Y \subseteq N_1$ with $g_1(\emptyset) = 0$, where $\hat{s}_v(Y \setminus \{v\})$ is the highest score of a node v and its parents when the parents are chosen from $Y \setminus \{v\}$; see the equation (2.2). The score of $\hat{A}$ is obtained as the maximum of $g_0(N_0) + g_1(N_1)$ over all partitions $\{N_0, N_1\}$.

Next, we analyze the time and space requirement of the algorithm. Here the difference compared to some other permutation problems of degree 1, like Feedback, Treewidth, and Scheduling, is that the evaluation of the local score, $\hat{s}_v(Y)$, can take exponential time. Let us first consider the explicit input.

The two subproblems, g_0 and g_1, are independent given the partition $\{N_0, N_1\}$, and thus can be solved separately. Applying the exact dynamic programming algorithm from Section 2.3.1 solves the first subproblem, that is, computes g_0 in $O(2^s n^2)$ time and $O(2^s n)$ space. We note that evaluating a local score $s_v(Y)$ now takes $O(n)$ time. This, however, does not affect the total time requirement of the recurrence in Lemma 2.10.

Computation of g_1 is slightly more complicated, since evaluating the $\hat{s}_v(N_0 \cup Y)$ requires the consideration of all possible subsets of N_0 as parents of v, in addition to a subset from $Y \setminus \{v\}$. To this end, for all $X_1 \subseteq N_1 \setminus \{v\}$ we write

$$s'_v(X_1) = \max\{s_v(X) : X \cap N_1 = X_1, X \in \mathcal{F}_v\}. \tag{3.7}$$

The scores $s'_v(X_1)$ for all $X_1 \subseteq N_1$ can be computed using Algorithm 3.

Algorithm 3 Compute $s'_v(X_1)$.

```
Input: Scores $s_v(X)$ for all $X \in \mathcal{F}_v$.
Output: $s'_v[X_1]$ for all $X_1 \subseteq N_1$.
1: for $X_1 \subseteq N_1$ do
2:     $s'_v[X_1] \leftarrow -\infty$
3: end for
4: for $X \in \mathcal{F}_v$ do
5:     $X_1 \leftarrow X \cap N_1$
6:     $s'_v[X_1] \leftarrow \max\{s'_v[X_1], s_v(X)\}$
7: end for
8: return $s'_v[X_1]$ for all $X_1 \subseteq N_1$
```

Observe that the loop in line 1 is executed 2^{n-s} times. Listing the sets $X \in \mathcal{F}_v$ required in line 4 takes $O(Fn\tau(n))$, where F is the size of $\mathcal{F}_v$

and the membership in $\mathcal{F}_v$ can be evaluated in $\tau(n)$ time. The loop in line 4 is executed F times. The lines 5 and 6 can be computed in $O(n)$ time yielding the total time requirement $O((F\tau(n)+2^{n-s})n)$. Now because $\hat{s}_v(N_0 \cup X_1) = \max_{Y \subseteq X_1} s'_v(Y)$, the exact dynamic programming algorithm again applies to computing g_1, running in $O((F\tau(n)+2^{n-s})n^2+2^{n-s}n^2) = O((F\tau(n)+2^{n-s})n^2)$ time and $O(2^{n-s}n)$ space in total. Because there are $\binom{n}{s}$ different partitions $\{N_0, N_1\}$, we get the following result.

Proposition 3.6 *The* Osd *problem with explicit input can be solved in* $O(\binom{n}{s}(2^s+F\tau(n))n^2)$ *time and* $O((2^s+F)n)$ *space for any* $s = n/2, n/2+1, \ldots, n$ *provided that each node has at most* F *possible parent sets, which form a downward-closed set family, and the membership in the possible parent sets can be evaluated in* $\tau(n)$ *time.*

Now consider the case with implicit input. The only time that the local scores are needed is the computation of $s'_v(X_1)$. In this case the time requirement of Algorithm 3 is $O((F\tau(n)\delta(n) + 2^{n-s})n)$ yielding the following result.

Proposition 3.7 *The* Osd *problem with implicit input can be solved in* $O(\binom{n}{s}(2^s + F\tau(n)\delta(n))n^2)$ *time and* $O(2^s n)$ *space for any* $s = n/2, n/2+1, \ldots, n$ *provided that each node has at most* F *possible parent sets, which form a downward-closed set family, the membership in the possible parent sets can be evaluated in* $\tau(n)$ *time and a local score can be computed from data in* $\delta(n)$ *time.*

Let us consider the divide-and-conquer scheme. Osd is a permutation problem of degree 1 and thus the sequences are void. Thus, $g(N_1, N_0)$ is the sum of

$$\max\Big\{g(N_{10}, N_0) + g(N_{11}, N_0 + N_{10})\Big\}$$

over all partitions $\{N_{10}, N_{11}\}$, where N_0 consist of the nodes preceding N_1. One can apply the recursion to depth t and then solve the remaining problem $g(N_1, N_0)$ by the dynamic programming recurrence in which $g'(\emptyset) = 0$ and

$$g'(Y) = \max_{v \in Y}\Big\{g'(Y \setminus \{v\}) + \hat{s}_v((Y \setminus \{v\}) \cup N_0)\Big\}$$

for nonempty $Y \subseteq N_1$. The Osd problem for node set N is solved by computing $g(N, \emptyset)$.

The analysis of the time and space requirements follows the same pattern as in Section 3.1.2. Here, however, we have to include the input in our analysis. Again, assume a balanced scheme and that the number of nodes, n, is a power of 2. Then, at depth $t \geq 0$ of the recursion, the node set of each subproblem is of size $s = n/2^t$. Therefore, each subproblem can be solved in $O^*(2^s)$ time and space, assuming that the number of possible parent sets for each variable is polynomial in n. More precisely, assuming explicit input, if every node has at most F possible parent sets, which form a downward-closed set family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time, a subproblem can be solved in $O((2^s n + F\tau(n))n)$ time and $O((2^s + F)n)$ space. Because each subproblem of size $2s$ is divided into $2\binom{2s}{s} \leq 2^{2s}$ subproblems of size s, the total number of subproblems of size s to be solved is at most $2^n 2^{n/2} 2^{n/4} \cdots 2^{2s} = 2^{2n-2s}$. We get the following bounds.

Theorem 3.8 *The* Osd *problem with explicit input can be solved in $O(2^{2n-2s}(2^s + F\tau(n))n^2)$ time and $O((2^s + F)n)$ space for any $s = n/2, n/4, n/8, \ldots, 1$, provided that each node has at most F possible parent sets, which form a downward-closed set family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Corollary 3.9 *The* Osd *problem with explicit input can be solved in $O(4^n n^{k+2} \tau(n))$ time and $O(n^{k+1})$ space, provided that each node has at most $O(n^k)$ possible parent sets, which form a downward-closed set family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Since the divide-and-conquer scheme handles the local scores the same way as the two-bucket scheme, we get the following result for the implicit input.

Theorem 3.10 *The* Osd *problem with implicit input can be solved in $O(2^{2n-2s}(2^s + F\tau(n)\delta(n))n^2)$ time and $O(2^s n)$ space for any $s = n/2, n/4, n/8, \ldots, 1$, provided that each node has at most F possible parent sets, which form a downward-closed set family, the membership in the possible parent sets can be evaluated in $\tau(n)$ time and a local score can be computed from data in $\delta(n)$ time.*

The results in this section apply particularly in the scenario in which the number of parents is bounded. If we want to lift this condition, the running time grows substantially. Indeed, the number of possible parent sets can be 2^{n-1} yielding the total time requirement $O(2^{3n-s}n^2)$. By setting $s = 1$, we get a theoretically interesting (but in practice quite useless) algorithm that solves the Osd problem without any restrictions on parent sets in $O(8^n n^2)$ time and polynomial space.

3.2.3 The Fp Problem

The Fp problem admits the two-bucket scheme and the divide-and-conquer scheme in a similar fashion as the Osd problem. Let us analyze the two-bucket scheme. Now we consider all possible partitions $\{N_0, N_1\}$ of N and have the recurrences

$$g_0(Y) = \sum_{v \in Y} g_0(Y \setminus \{v\}) \alpha_v(Y \setminus \{v\}), \tag{3.8}$$

for $\emptyset \subset Y \subseteq N_0$ with $g_0(\emptyset) = 1$, and

$$g_1(Y) = \sum_{v \in Y} g_1(Y \setminus \{v\}) \alpha_v((Y \setminus \{v\}) \cup N_0), \tag{3.9}$$

for $\emptyset \subset Y \subseteq N_1$ with $g_1(\emptyset) = 1$; the score is obtained as the sum of $g_0(N_0)g_1(N_1)$ over all partitions $\{N_0, N_1\}$.

The values of $\alpha_v(Z)$ for $Z \subseteq N_0$ can be computed by the standard fast zeta transform algorithm. For sets $N_0 \subseteq Z \subseteq N$, however, we need to do some minor modifications. To this end, for all $X_1 \subseteq N_1 \setminus \{v\}$ we write

$$\alpha'_v(X_1) = \sum_{\substack{X \in \mathcal{F}_v \\ X \cap N_1 = X_1}} \beta_v(X),$$

which is analogous to the equation (3.7). The values $\alpha'_v(X_1)$ for all $X_1 \subseteq N_1$ can be computed by Algorithm 4, which is a modified version of Algorithm 3. The values of $\alpha_v(X_1)$ for $X_1 \subseteq N_1$ are obtained by computing the zeta transform of $\alpha'_v(X_1)$ over N_1.

A straightforward application of the presented techniques enables solving the Fp problem using the divide-and-conquer scheme. As the same argumentation applies for both Osd and Fp, we summarize the time and space bounds yielded by the two-bucket scheme and the divide-and-conquer scheme as follows.

Theorem 3.11 *The bounds presented in Proposition 3.6, Proposition 3.7, Theorem 3.8, Corollary 3.9, and Theorem 3.10 hold for the* Fp *problem if the word* Osd *is replaced by* Fp.

Algorithm 4 Compute $\alpha'_v(X_1)$.

Input: Scores $\beta_v(X)$ for all $X \in \mathcal{F}_v$.
Output: $\alpha'_v(X_1)$ for all $X_1 \subseteq N_1$.

```
1: for $X_1 \subseteq N_1$ do
2:     $\alpha'_v[X_1] \leftarrow 0$
3: end for
4: for $X \in \mathcal{F}_v$ do
5:     $X_1 \leftarrow X \cap N_1$
6:     $\alpha'_v[X_1] \leftarrow \alpha'_v[X_1] + \beta_v(X)$
7: end for
8: return $\alpha'_v[X_1]$ for all $X_1 \subseteq N_1$
```

3.3 Time–Space Product

So far, we have obtained several different time and space bounds for permutation problems. Recall that the dynamic programming algorithm runs in $O^*(2^n)$ time and space. The two-bucket scheme yields different time and space requirements depending on the parameter s. For example, setting $s = (4/5)n$ yields $O^*(2.872^n)$ time and $O^*(1.742^n)$ space. On the other hand, setting $s = (3/5)n$ yields $O^*(2.972^n)$ time and $O^*(1.516^n)$ space. Also the divide-and-conquer scheme yields several different time and space bounds. For example, Corollary 3.3 yields $O^*(4^n)$ time and polynomial space. Setting $s = n/4$ yields $O^*(3.364^n)$ time and $O^*(1.190^n)$ space. But how good or efficient are these bounds?

In order to analyze the “goodness” or “efficiency” of a space–time trade-off, we need a measure, preferably one that summarizes the efficiency into a single number. As a candidate for such a measure we consider the *time–space product* which we define as follows.

Definition 3.12 (Time–space product.) *The time–space product of an algorithm with the time requirement $O^*(T^n)$ and the space requirement $O^*(S^n)$ is the infimum of*

$$\theta = TS.$$

Later in Section 4.2 we extend this definition to handle the partial order schemes that will be presented in Chapter 4. We notice that this kind of efficiency measure is sensible only for exponential time algorithms as all polynomial time (and space) algorithms have the time–space product 1.

Our base case is the dynamic programming algorithm running in $O^*(2^n)$ time and space yielding the time–space product 4 and we will compare the space–time tradeoff schemes to this number. Intuitively, if an algorithm yields a time–space product less than 4, then whenever the space requirement (compared to the base case) is halved, the time requirement less than doubles.

The divide-and-conquer scheme also has time–space product 4 as it yields the bounds $O^*(2^{2n-s})$ for time and $O^*(2^s)$ for space. The two-bucket scheme, however, generally yields larger time–space products. Setting $s = (4/5)n$ yields $O^*(2.872^n)$ time, $O^*(1.742^n)$ space, and time–space product 5. On the other hand, setting $s = (3/5)n$ yields $O^*(2.972^n)$ time, $O^*(1.516^n)$ space, and time–space product 4.51. In general, this approach yields a smooth time–space tradeoff for space bounds from $O^*(2^{n/2})$ to $O^*(2^n)$. However, this is not the end of the story. In Chapter 4 we will generalize the two-bucket scheme and obtain time–space products smaller than 4.

Chapter 4

The Partial Order Approach

Linear orders have been a key ingredient in many algorithms for score-based structure discovery in Bayesian networks. Cooper and Herskovits [19] observed that given a linear order and allowing the nodes to have at most a constant number of parents, an optimal structure can be found in polynomial time. Unfortunately, in practice we usually do not know the correct order and we have to resort to other means. Linear orders have been employed in several heuristic algorithms. For example, Friedman and Koller [39] found out that an MCMC sampler that works in the space of linear orders mixes considerably faster than the one that works directly with the structures. Further, Teyssier and Koller [104] introduced an algorithm that searches an optimal ordering using greedy hill-climbing. Also the exact dynamic programming algorithms (Section 2.3) work in the space of linear orders.

In this chapter we present a partial order approach for permutation problems in general and for structure discovery in Bayesian networks in particular. The partial order approach can be viewed as an extension of the dynamic programming method in Section 2.3. It is also a generalization of the two-bucket scheme. We present a “sparse” dynamic programming algorithm that solves the structure discovery problems over DAGs that are compatible with a given partial order. The idea is to consider a family of partial orders that together cover all linear orders and solve the subproblem for each partial order separately. In the end, the results from the subproblems are combined to get a solution to the original problem.

We begin this chapter by presenting dynamic programming over partial orders for permutation problems in general. Then, we accommodate the algorithm to solve the Osd and Fp problems. In this chapter we present our results for partial orders in general; later in Chapter 5 we consider theoretical and practical issues concerning the choice of partial orders.

4.1 Partial Orders

In this section we introduce the partial order terminology needed in this thesis. For a more thorough representation of partial orders we refer the reader to Davey and Priestley [23].

A *partial order* P on a *base-set* M is a subset of $M \times M$ such that for all $x, y, z \in M$ it holds that

1. $xx \in P$ (reflexivity),
2. $xy \in P$ and $yx \in P$ imply $y = x$ (antisymmetry), and
3. $xy \in P$ and $yz \in P$ imply $xz \in P$ (transitivity).

If $xy \in P$ we say that x *precedes* y. A partial order P is a *linear order* if in addition $xy \in P$ or $yx \in P$ for all $x, y \in M$ (totality).

A linear order Q is a *linear extension* of a partial order P if $P \subseteq Q$. Recall that every permutation is in a one-to-one relationship with a linear order. Thus, we say that a permutation σ is a linear extension of a partial order P if the linear order corresponding to σ is a linear extension of P. For our purposes, it is useful to define the *trivial order* P on M as the "diagonal" partial order $\{xx : x \in M\}$. For a set $Y \subseteq M$, an *induced partial order* $P[Y]$ is the partial order $P \cap Y \times Y$.

A DAG A and a partial order P are said to be *compatible* with each other if there exists a partial order Q such that $P \subseteq Q$ and $A \subseteq Q$. In other words, some topological ordering of A is a linear extension of the partial order.

A set $\mathcal{P}$ is a *partial order family*—or a *POF* for short—on N if $\mathcal{P}$ consists of partial orders on N. Next, we define two types of POFs that will play a key role in our partial order approach.

Definition 4.1 (Cover) *A POF $\mathcal{P}$ on N is a cover of, or covers, the linear orders on N if every linear order on N is a linear extension of at least one partial order in $\mathcal{P}$.*

Definition 4.2 (Exact cover) *A POF $\mathcal{P}$ on N is an exact cover of, or exactly covers, the linear orders on N if every linear order on N is a linear extension of exactly one partial order in $\mathcal{P}$.*

An ideal (or a down-set, in operations research also a feasible set) of a partial order P is a set of elements that can "begin" some linear extension of P. Formally, an *ideal* I of a partial order P is a subset of elements such that $y \in I$ and $xy \in P$ implies $x \in I$. Another ideal I' of P is called a *subideal of* I if $I' \subseteq I$. Analogically, an ideal I' is a *superideal of* I if $I \subseteq I'$. We denote by $\mathcal{I}(P)$ the set of all the ideals of the partial order P.

Sometimes one wants to visualize the aforementioned concepts. Recall that a *partially ordered set* (poset) is a set M associated with a partial order P on M. It can be illustrated by a *Hasse diagram*. In a Hasse diagram every $x \in M$ is depicted by a point $p(x)$ in the Euclidean plane. Points $p(x)$ and $p(y)$, $x \neq y$, are connected to each other by an edge if x directly precedes y, that is, $xy \in P$ and there is no $z \in M$, $x \neq z \neq y$ such that $xz \in P$ and $zy \in P$. The points are laid out in a way that if $xy \in P$ then $p(x)$ is located lower than $p(y)$ in the diagram.

The ideals of a partial order can be illustrated by a *subset lattice*. A lattice is a partially ordered set where for every pair of elements there exists a least upper bound (join) and a greatest lower bound (meet). If ideals are ordered by $\subseteq$-relation, then $\cup$ is a join-operation and $\cap$ is a meet-operation[1]. The subset lattice can be illustrated by a Hasse diagram; see Example 4.

The following example illustrates these concepts.

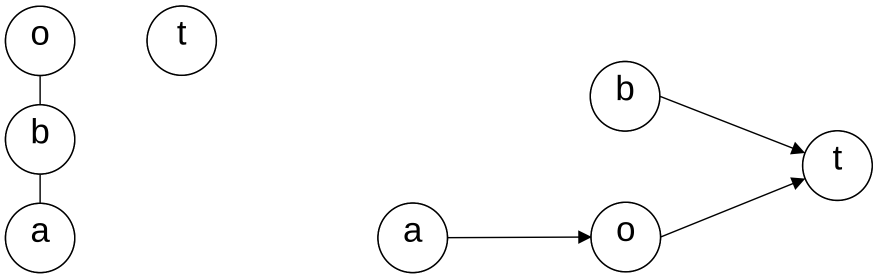
Example 4. Recall Example 1 with an employee who tends to be late. For an illustration of these concepts, consider the partial order $P = \{aa, ab, ao, bb, bo, oo, tt\}$ on the base-set $M = \{a, b, o, t\}$. A Hasse diagram of P is shown in Figure 4.1(a). The set of ideals $\mathcal{I}(P)$ consists of nine sets: $\emptyset$, $\{a\}$, $\{b\}$, $\{a, b\}$, $\{a, o\}$, $\{b, t\}$, $\{a, b, o\}$, $\{a, b, t\}$, and $\{a, b, o, t\}$; note that, for example, $\{o\}$ is not an ideal, because node a always precedes node o, and therefore o cannot be the first node in any linear extension of P. Subset lattices induced by the trivial order and P are shown in Figure 4.2.

Consider a DAG $A = \{ao, bt, ot\}$ on $N = \{a, b, o, t\}$ shown in Figure 4.1(b). The DAG A is compatible with the partial order P because by setting $Q = P$, we have $A \subseteq Q$ and $P \subseteq Q$. The partial order P has 4 linear extensions: *abot*, *abto*, *atbo*, and *tabo*. A permutation *abot* corresponds to a linear order $Q = \{aa, ab, ao, at, bb, bo, bt, oo, ot, tt\}$ and $A \subseteq Q$ and therefore A is compatible with P. $\diamond$

4.2 Dynamic Programming over Partial Orders

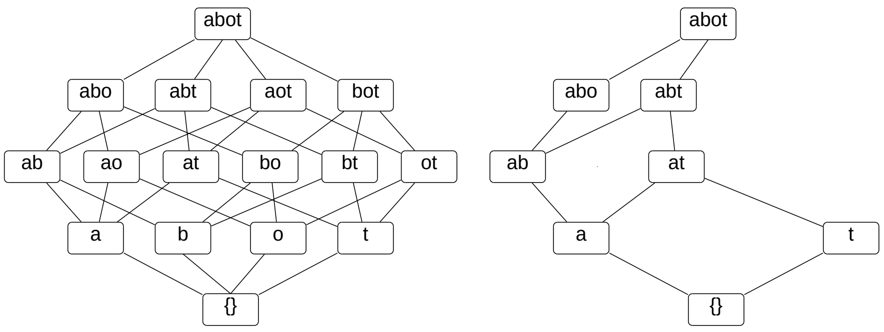
The early development of the dynamic programming algorithms that accommodate information about partial orders was motivated by the opera-

[1]Join and meet operations mean that if $X, Y \in \mathcal{I}(P)$ then both $X \cup Y \in \mathcal{I}(P)$ and $X \cap Y \in \mathcal{I}(P)$.



(a) (b)

Figure 4.1: (a) A Hasse diagram of the partial order $P = \{aa, ab, ao, bb, bo, oo, tt\}$. (b) A DAG with $N = \{a, b, o, t\}$ and $A = \{ao, bt, ot\}$.



(a) (b)

Figure 4.2: Subset lattices of the ideals of (a) the trivial order on $N = \{a, b, o, t\}$ and (b) the partial order $P = \{aa, ab, ao, bb, bo, oo, tt\}$.

tions research community's need to solve scheduling problems with precedence constraints, that is, when the jobs must be scheduled according to some partial order. Baker and Schrage [5, 92] introduced a labeling scheme to reduce the number of states in the dynamic programming. Later, Lawler [71] further developed Baker's and Schrage's work and gave an algorithm that solves a precedence-constrained scheduling problem in time and space proportional to the number of ideals of the partial order determined by the precedence constraints.

Next, we further develop the ideas by Baker, Schrage, and Lawler and generalize the dynamic programming algorithm presented in Section 2.3 to compute the sum of costs over all linear extensions of a given partial order P on N. In what follows we use a shorthand $l = |Y|$. For any ideal $Y \in \mathcal{I}(P)$ and elements $v_1, v_2 \ldots, v_{d-1} \in Y$, $v_i \neq v_k$ if $i \neq k$, define

$$g^P(Y, v_{d-1} \cdots v_2 v_1) = \bigoplus_{\sigma} \bigodot_{j=1}^{l} c(\{\sigma_1, \sigma_2, \ldots, \sigma_j\}, \sigma_{j-d+1} \cdots \sigma_{j-1}\sigma_j),$$

where the summation is over all linear extensions σ of the induced partial order $P[Y]$ with $\sigma_{l-d+2} \cdots \sigma_{l-1}\sigma_l = v_{d-1} \cdots v_2 v_1$. Here, if $d > j$ we read $\sigma_{j-d+1} \cdots \sigma_{j-1}\sigma_j$ as $\sigma_1 \cdots \sigma_{j-1}\sigma_j$, and if $d = 0$ the sequence is void. Note that if P is a trivial order, then g^P equals the g defined in the equation (2.5).

Consider $g^P(Y, v_{d-1} \cdots v_2 v_1)$. If $Y \setminus \{v_1\}$ is not an ideal of P, then $P[Y]$ does not have any linear extensions with v_1 as the last element. Thus, the sum is empty and $g^P(Y, v_{d-1} \cdots v_2 v_1) = 0$. Similarly, if $Y \setminus \{v_1, v_2, \ldots, v_k\}$ for $k \leq d-1$ is not an ideal of P then $g^P(Y, v_{d-1} \cdots v_2 v_1) = 0$. Suppose therefore that $Y \setminus \{v_1, v_2, \ldots, v_k\}$ is an ideal of P for all $k \leq d-1$. Then, any linear extension σ of $P[Y]$ with $\sigma_{l-d+1} \cdots \sigma_{l-1}\sigma_l = v_{d-1} \cdots v_2 v_1$ determines a linear extension $\sigma' = \sigma(Y \setminus \{v_1, v_2, \ldots, v_{d-1}\})$ of $P[Y \setminus \{v_1, v_2, \ldots, v_{d-1}\}]$ with some $v_d = \sigma'_{l-d+1}$. Thus, the recurrence takes the form

$$\begin{aligned} g^P(\emptyset, \emptyset) &= 1\,, & (4.1)\\ g^P(Y, v_{d-1} \cdots v_2 v_1) &= \bigoplus_{v_d \in Y \setminus \{v_1, v_2, \ldots v_{d-1}\}} g^P(Y \setminus \{v_1\}, v_d v_{d-1} \cdots v_2) \\ &\quad \odot c(Y, v_d v_{d-1} \cdots v_2 v_1) & (4.2) \end{aligned}$$

for $Y \setminus \{v_1, v_2, \ldots, v_k\} \in \mathcal{I}(P)$ for $0 \leq k \leq d$, and $l \geq 1$. That is, the formulas are the same as in the basic version but applied only for ideals Y of P where $v_k \in Y$ is a maximal element of $Y \setminus \{v_1, v_2, \ldots, v_{k-1}\}$. We get the following result.

Theorem 4.3 *For any partial order P on N it holds*

$$\bigoplus_{\substack{v_1, v_2, \ldots, v_{d-1} \in N \\ v_i \neq v_k \text{ if } i \neq k}} g^P(N, v_{d-1} \cdots v_2 v_1) = \bigoplus_{\sigma^P} c(\sigma^P),$$

where σ^P runs over linear extensions of P and $c(\sigma^P) = \odot_{j=1}^{n} c(\{\sigma_1^P, \sigma_2^P, \ldots, \sigma_j^P\}, \sigma_{j-d+1}^P \cdots \sigma_{j-1}^P \sigma_j^P)$.

Proof. We show by induction on the size of Y that $\bigoplus_{v_1, v_2, \ldots, v_{d-1} \in Y} g^P(Y, v_{d-1} \cdots v_2 v_1)$ equals the sum of costs $c(\sigma(Y))$ over the linear extensions of $P[Y]$, assuming Y is an ideal of P.

First, consider an arbitrary singleton $\{v_1\} \in \mathcal{I}(P)$. Clearly, there is exactly one permutation on $\{v_1\}$ and it is compatible with $P[\{v_1\}] = \{v_1v_1\}$; the score of the permutation is $1 \oplus c(\{v_1\}, v_1) = c(\{v_1\}, v_1)$. This is precisely what the recurrence (4.2) gives.

Suppose then that the recurrence (4.2) holds for all proper subideals of an ideal $Y \subseteq N$. Without any loss of generality, assume $Y = N$ for notational convenience. Now, write

$$\begin{aligned}
\bigoplus_{\sigma \in P[N]} c(\sigma) &= \bigoplus_{\sigma \in P[N]} \bigodot_{j=1}^{n} c(\{\sigma_1, \sigma_2, \ldots, \sigma_j\}, \sigma_{j-d+1} \cdots \sigma_{j-1}\sigma_j) \\
&= \bigoplus_{v_1, v_2, \ldots, v_{d-1} \in N} \quad \bigoplus_{v_d \in N \setminus \{v_1, v_2, \ldots, v_{d-1}\}} \Big\{ c(N, v_d v_{d-1} \cdots v_2 v_1) \\
&\quad \odot \bigoplus_{\substack{\sigma' \in P[N \setminus \{v_1\}] \\ \sigma'_{j-d+2} \cdots \sigma'_{j-1}\sigma'_j = v_d v_{d-1} \cdots v_2}} \\
&\quad \bigodot_{j=1}^{n-1} c(\{\sigma_1, \sigma_2, \ldots, \sigma_j\}, \sigma_{d-1} \cdots \sigma_{j-1}\sigma_j) \Big\} \\
&= \bigoplus_{v_1, v_2, \ldots, v_{d-1} \in N} \quad \bigoplus_{v_d \in N \setminus \{v_1, v_2, \ldots, v_{d-1}\}} \Big\{ c(N, v_d v_{d-1} \cdots v_2 v_1) \\
&\quad \odot g^{P[N \setminus \{v_1\}]}(N \setminus \{v_1\}, v_d v_{d-1} \cdots v_3 v_2) \Big\} \\
&= \bigoplus_{v_1, v_2, \ldots, v_{d-1} \in N} g^P(N, v_{d-1} \cdots v_2 v_1).
\end{aligned}$$

The first identity holds by the definitions of linear extension and decomposability; the second one by the distributive law and the fact that some elements $v_d, v_{d-1}, \ldots, v_2, v_1$ are the last d elements in permutation σ; the third one by the induction step; the fourth one by the definition of g^P and the fact that $g^P(Y) = g^{P[Y]}(Y)$ for $Y \in \mathcal{I}(P)$, since clearly a subset X of Y is an ideal of P if and only if X is an ideal of $P[Y]$. □

By applying recurrences (4.1) and (4.2) we get the following.

Proposition 4.4 *The sum of costs $c(\sigma)$ over all linear extensions σ of a given partial order P on N can be computed in $O^*(|\mathcal{I}(P)|)$ time and space assuming that the local costs can be evaluated in polynomial time.*

To extend the sum over all linear orders, we generally need to consider more than one partial order. Recall Definition 4.2 which states that a POF

$\mathcal{P}$ exactly covers all linear orders on N if every linear order is a linear extension of exactly one partial order in $\mathcal{P}$. Given such an exact cover, the sum over all linear orders can be computed by simply computing the sum separately for each partial order and finally taking the sum over the $|\mathcal{P}|$ results. We get the following theorem.

Theorem 4.5 *If $\mathcal{P}$ is a POF that exactly covers the linear orders on N, then*

$$\bigoplus_{P\in\mathcal{P}} \bigoplus_{v_1,v_2,\ldots,v_{d-1}\in N} g^P(N, v_{d-1}\cdots v_2 v_1) = \bigoplus_{\sigma} c(\sigma),$$

where σ runs through all linear orders on N.

Proof. It suffices to note that

$$\bigoplus_{P\in\mathcal{P}} \bigoplus_{v_1,v_2,\ldots,v_{d-1}\in N} g^P(N, v_{d-1}\cdots v_2 v_1) = \bigoplus_{P\in\mathcal{P}} \bigoplus_{\sigma^P} c(\sigma^P) = \bigoplus_{\sigma} c(\sigma),$$

where σ^P runs over the linear extensions of P. Here the first equality holds by Theorem 4.3 and the second one by the definition of an exact cover (Definition 4.2). □

Thus, by Proposition 4.4 and Theorem 4.5 we have the following.

Proposition 4.6 *Let $\mathcal{P}$ be a family of partial orders that exactly covers the linear orders on N. Then any polynomial local time permutations problem can be solved in $O^*(\sum_{P\in\mathcal{P}} |\mathcal{I}(P)|)$ time and $O^*(\max_{P\in\mathcal{P}} |\mathcal{I}(P)|)$ space.*

Note that if the problem is defined in an idempotent semiring, $\mathcal{P}$ does not have to be an exact cover. Any $\mathcal{P}$ that covers all linear orders will do. However, it is not known whether there exists a (non-exact) cover $\mathcal{P}$ such that the time requirement $\sum_{P\in\mathcal{P}} |\mathcal{I}(P)|$ is smaller than the time requirement of every exact cover whose space requirement equals $\max_{P\in\mathcal{P}} |\mathcal{I}(P)|$.

Now that we know how to compute the time and space requirements of dynamic programming over partial orders given a POF, we extend the time–space product to this setting. The *time–space product of a POF $\mathcal{P}$* on N is defined as the nth root of the product

$$\theta(\mathcal{P}) = \Big(\sum_{P\in\mathcal{P}} |\mathcal{I}(P)|\Big)\Big(\max_{P\in\mathcal{P}} |\mathcal{I}(P)|\Big). \tag{4.3}$$

Moreover, we define the *time–space product of a bounded degree permutation problem* to be the infimum of the time–space product of all algorithms that solve the problem in question. To upper bound the time–space product of the bounded degree permutation problems, consider a sequence of partial order families $(\mathcal{P}_n)$ such that each $\mathcal{P}_n$ exactly covers the linear orders on $\{1, 2, \ldots, n\}$. Then the time–space product of the bounded degree permutation problem is at most $\lim_n \theta(\mathcal{P}_n)^{1/n}$ (supposing the limit exists).

4.3 The Osd Problem

The dynamic programming over partial orders can be applied straightforwardly to the Osd problem since Osd is a permutation problem. By adapting the dynamic programming recurrences from the previous section, we define function g^P by $g^P(\emptyset) = 0$ and for nonempty $Y \in \mathcal{I}(P)$ recursively:

$$g^P(Y) = \max_{\substack{v \in Y \\ Y \setminus \{v\} \in \mathcal{I}(P)}} \Big\{ g^P(Y \setminus \{v\}) + \hat{s}_v(Y \setminus \{v\}) \Big\}, \tag{4.4}$$

where $\hat{s}_v(Y \setminus \{v\})$ is the highest local score when the parents of v are chosen from $Y \setminus \{v\}$. It follows by Theorem 4.3 that $g^P(N)$ equals the maximum score over the DAGs compatible with P. The DAG that maximizes the score can be constructed using an adapted version of Algorithm 1. Now, given a POF $\mathcal{P}$ that covers the linear orders on N, the Osd problem can be solved by computing $\max_{P \in \mathcal{P}} g^P(N)$.

By Proposition 4.4, the dynamic programming algorithm evaluates the function g^P along the recurrence (4.4) in $O^*(|\mathcal{I}(P)|)$ time and space, provided that the values $\hat{s}_v(Y \setminus \{v\})$ are precomputed. However, the computation of $\hat{s}_v(Y \setminus \{v\})$ is a problem of its own. In Section 2.3.1 we were able to use Lemma 2.10 and compute $\hat{s}_v(Y \setminus \{v\})$ for all $Y \subseteq N$ in essentially the same time as needed for solving Osd given the values $\hat{s}_v(Y \setminus \{v\})$. Here, however, the task is more involved due to the need to use only a limited amount of space. We will show that under some mild conditions, the values $\hat{s}_v(Y \setminus \{v\})$ can be computed in $O^*(|\mathcal{I}(P)|)$ time and space. First, however, consider a naïve computation where $\hat{s}_v(Y \setminus \{v\})$ given the values $s_v(Y \setminus \{v\})$ is computed from scratch for each $Y \subseteq N$. We get the following.

Lemma 4.7 *Given a partial order P on N and scores $s_v(Y \setminus \{v\})$ for $v \in N$ and $Y \setminus \{v\} \in \mathcal{F}_v$, a DAG A that is compatible with P and maximizes the score $s(A)$, can be found in $O(nF\tau(n)|\mathcal{I}(P)|)$ time and $O(n(F + |\mathcal{I}(P)|))$ space, provided that each node has at most F possible parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Now combined with Theorem 4.5, this gives the following summary.

Proposition 4.8 *Given a POF $\mathcal{P}$ that covers the linear orders on N and explicit input, the* OSD *problem can be solved in $O(nF\tau(n)\sum_{P\in\mathcal{P}}|\mathcal{I}(P)|)$ time and $O(n(F+\max_{P\in\mathcal{P}}|\mathcal{I}(P)|))$ space, provided that each node has at most F possible parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

However, we would like to improve the time and space bounds. Especially, we would like to remove the multiplicative factor F—which can be as large as 2^{n-1}—from the running time bound, like we did in Section 2.3.1 with Lemma 2.10 by reusing the scores computed earlier. Unfortunately, Lemma 2.10 does not apply directly in our general setting. However, under some general assumptions that bound the number of possible parent sets, the multiplicative factor can be turned into an additive factor. Next we address this issue by generalizing the idea of Lemma 2.10.

The idea is to arrange the computation of the recurrence (4.4) in such a way that some of the values of $\hat{s}_v(Y\setminus\{v\})$, the scores of the best parent set chosen from $Y\setminus\{v\}$, may be reused. An immediate attempt would be to do dynamic programming described in Lemma 2.10 and keep only two levels in memory at any given time, but this seems to take space proportional to $2^n/\sqrt{n}$ for each v [79]. Luckily, it turns out that we can use a sparse dynamic programming variant that is explained next.

The key insight is the following observation, which is stated in plain combinatorial terms.

Lemma 4.9 *Let X and Y be sets with $X\subseteq Y$. Let*

$$\begin{aligned}\mathcal{A} &= [X,Y],\\ \mathcal{B} &= \{Z\subseteq Y : x\notin Z \textit{ for some } x\in X\}.\end{aligned}$$

Then (i) $2^Y=\mathcal{A}\cup\mathcal{B}$ and (ii) $\mathcal{B}=\bigcup_{x\in X}2^{Y\setminus\{x\}}$.

Proof. (i) "$\supseteq$" is obvious. So, consider "$\subseteq$": Let $Z\subseteq Y$: If $X\subseteq Z$, then $Z\in\mathcal{A}$. Otherwise, there exists $x\in X$ such that $x\notin Z$, hence $Z\in\mathcal{B}$.

(ii) By definition, $Z\in\mathcal{B}$ if and only if $Z\subseteq Y\setminus\{x\}$ for some $x\in X$. □

Intuitively, $\mathcal{A}$ consists of the subsets of Y that contain X. The set $\mathcal{B}$, on the other hand, consists of the subsets of Y that do not contain all elements of X. Lemma 4.9(i) states that every subset of Y either contains X or does not contain X. Further, Lemma 4.9(ii) states that each subset of Y that does not contain X is a subset of at least one $Y\setminus\{x\}$ with $x\in X$.

In terms of the set functions s_v and $\hat{s}_v$, for an arbitrary v, the above lemma amounts to the following generalization of Lemma 2.10.

Lemma 4.10 *Let X and Y be subsets of $N \setminus \{v\}$ with $X \subseteq Y$. Then*

$$\hat{s}_v(Y) = \max\Big\{ \max_{X \subseteq Z \subseteq Y} s_v(Z), \max_{u \in X} \hat{s}_v(Y \setminus \{u\}) \Big\}.$$

Proof. By Lemma 4.9(ii) and the definition of $\hat{s}_v$, the maximum of $s_v(Z)$ given $Z \in \bigcup_{x \in X} 2^{Y \setminus \{x\}}$ equals $\max_{u \in X} \hat{s}_v(Y \setminus \{u\})$. Further, by Lemma 4.9(i), the larger of $\max_{X \subseteq Z \subseteq Y} s_v(Z)$ and $\max_{u \in X} \hat{s}_v(Y \setminus \{u\})$ equals $\max_{V \subseteq Y} s_v(V)$, which is the definition of $\hat{s}_v(Y)$. □

In words, Lemma 4.10 states that the maximum score for v given that its parents are chosen from Y is the maximum of the maximum over parent sets that contain X and the maximum over the parent sets that do not contain X. Note that Lemma 2.10 is a special case of Lemma 4.10, with $X = Y$.

Lemma 4.10 leaves us the freedom to choose a suitable node subset X for each set of interest Y. How to make the choice is guided by the fact that in the evaluation of g^P, the values of $\hat{s}_v(Y)$ are needed only for sets Y that belong to $\mathcal{I}(P)$; in what follows we consider $P \in \mathcal{P}$ fixed. By storing the values $\hat{s}_v(Y)$ only for sets $Y \in \mathcal{I}(P)$, we adhere to the space requirement (up to a polynomial factor) already needed for storing $g^P(Y)$ for $Y \in \mathcal{I}(P)$. Thus, the goal is to choose X such that $Y \setminus \{u\} \in \mathcal{I}(P)$ for all $u \in X$. To this end, let X consist of all such nodes in Y that have no strictly larger node in Y (with respect to P), that is, X consists of maximal elements of Y. Accordingly, for $Y \in \mathcal{I}(P)$ define the set

$$\check{Y} = \{u \in Y : uv \notin P \text{ for all } v \in Y \setminus \{u\}\}.$$

Furthermore, define the *tail* of Y (with respect to P) as the interval

$$\mathcal{T}_Y = [\check{Y}, Y].$$

The sets $\check{Y}$ and Y are called the *bottom* and the *top* of $\mathcal{T}_Y$, respectively.

Example 5. Let us elucidate the concept of tails. Suppose we are given a set $N = \{a, b, o, t\}$ and a partial order $P = \{aa, ab, ao, bb, bo, oo, tt\}$. Now, all ideals of P and their tails are shown in Figure 4.3. Consider, for example, a set $Y = \{a, b, o, t\}$. The elements o and t do not have any strictly larger element in Y, and thus $\check{Y} = \{o, t\}$. The tail $\mathcal{T}_Y$ consists of the sets $\{o, t\}$, $\{a, o, t\}$, $\{b, o, t\}$, and $\{a, b, o, t\}$; $\mathcal{T}_Y$ is colored in orange in Figure 4.3. ◇

The next two lemmas show that $\check{Y}$ indeed has the desired property (in a maximal sense), and that the tails of different ideals are pairwise disjoint and thus partition the subsets of N.

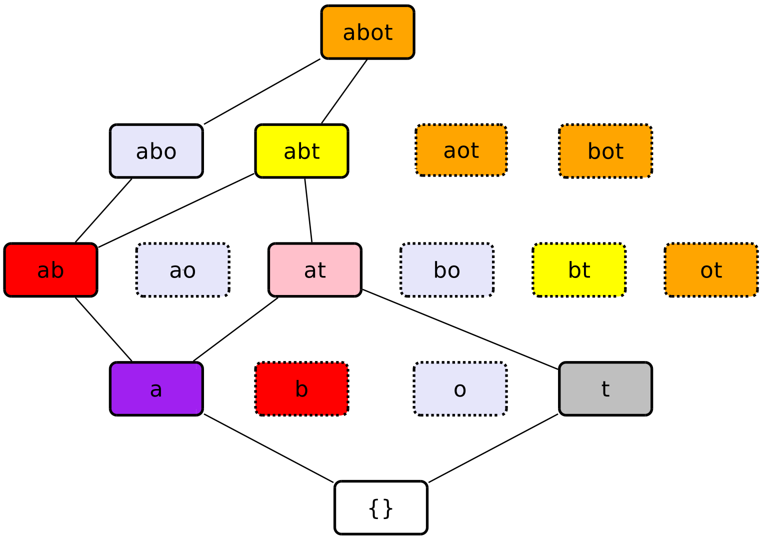


Figure 4.3: The tails of the ideals of a partial order $P = \{aa, ab, ao, bb, bo, oo, tt\}$ on $N = \{a, b, o, t\}$. Subsets of N that are surrounded by solid lines are ideals of P and subset with dashed lines are not ideals. An ideal and its tail are colored in the same color.

Lemma 4.11 *Let $Y \in \mathcal{I}(P)$ and $u \in Y$. Then $Y \setminus \{u\} \in \mathcal{I}(P)$ if and only if $u \in \check{Y}$.*

Proof. "If": Let $u \in \check{Y}$. Let $st \in P$. By the definition of $\mathcal{I}(P)$ we need to show that $t \in Y \setminus \{u\}$ implies $s \in Y \setminus \{u\}$. So, suppose $t \in Y \setminus \{u\}$, hence $t \in Y$. Now, since $Y \in \mathcal{I}(P)$, we must have $s \in Y$. It remains to show that $s \neq u$. But this holds because $ut \notin P$ by the definition of $\check{Y}$.

"Only if": Let $u \notin \check{Y}$. Then we have $uv \in P$ for some $v \in Y \setminus \{u\}$. But $u \notin Y \setminus \{u\}$ and $v \in Y \setminus \{u\}$, implying $Y \setminus \{u\} \notin \mathcal{I}(P)$ by the definition of $\mathcal{I}(P)$. □

Lemma 4.12 *Let Y and Y' be two distinct sets in $\mathcal{I}(P)$. Then the tails of Y and Y' are disjoint.*

Proof. Suppose to the contrary that $Z \in \mathcal{T}_Y \cap \mathcal{T}_{Y'}$. By symmetry we may assume that $Y \setminus Y'$ contains an element w. Thus $w \notin Z$, because $Z \subseteq Y'$. Because $\check{Y} \subseteq Z$, we have $w \notin \check{Y}$. By definition of $\check{Y}$ we conclude that for every $u \in Y \setminus \check{Y}$ there exists $v \in \check{Y}$ such that $uv \in P$. Therefore, in particular there exists $v \in \check{Y}$ such that $wv \in P$. Since $w \notin Y'$ and Y' is an ideal of P it follows by definition of an ideal that $v \notin Y'$. On the other hand, $v \in \check{Y}$ and $\check{Y} \subseteq Z \subseteq Y'$ implies that $v \in Y'$: contradiction. □

The results of the previous lemmas are illustrated in Figure 4.3. Indeed, every subset of $N = \{a, b, o, t\}$ is covered by the tails (Lemma 4.11) and the tails are disjoint (Lemma 4.12).

As a remark, notice that in the recurrence (3.7) the maximization is over the tail of X_1 with respect to partial order $P = \{xy \in N \times N : x = y \text{ or } (x \in N_0 \text{ and } y \in N_1)\}$. Thus, in Section 3.2.2 we actually employed Lemma 4.10.

We now merge the ingredients given above into an algorithm for evaluating g^P using the recurrence (4.4), for fixed $P \in \mathcal{P}$. In Algorithm 5 below, $g^P[Y]$ and $\hat{s}_v[Y]$ denote program variables that correspond to the respective target values $g^P(Y)$ and $\hat{s}_v(Y)$ to be computed. Also, recall that $\mathcal{F}_v$ denotes the family of possible parent sets for node v.

Algorithm 5 Compute g^P.

```
Input: Partial order $P$, local scores $s_v(Y)$ for all $v$ and $Y \in \mathcal{F}_v$.
Output: Maximum score of a DAG compatible with $P$.
 1: $g^P[\emptyset] \leftarrow 0$.
 2: for all $v \in N$ do
 3:    $\hat{s}_v[\emptyset] \leftarrow s_v(\emptyset)$.
 4: end for
 5: for all $Y \in \mathcal{I}(P)$, in increasing order of cardinality do
 6:    $g^P[Y] \leftarrow \max_{v \in \check{Y}} \left\{ g^P[Y \setminus \{v\}] + \hat{s}_v[Y \setminus \{v\}] \right\}$.
 7:    for all $v \in N$ do
 8:       $\hat{s}_v[Y] \leftarrow \max \left\{ \max_{Z \in \mathcal{T}_Y \cap \mathcal{F}_v} s_v(Z), \max_{u \in \check{Y}} \hat{s}_v[Y \setminus \{u\}] \right\}$.
 9:    end for
10: end for
```

Lemma 4.13 *Algorithm 5 correctly computes g^P, that is, $g^P[Y] = g^P(Y)$ for all $Y \in \mathcal{I}(P)$.*

Proof. By the definition of g^P in the recurrence (4.4) and by Lemma 4.10 it suffices to notice that, by Lemma 4.11, the condition "$v \in Y$ and $Y \setminus \{v\} \in \mathcal{I}(P)$" is equivalent to "$v \in \check{Y}$", given that $Y \in \mathcal{I}(P)$. Note that maximizing over $\mathcal{T}_Y \cap \mathcal{F}_v$ is maximizing over $\mathcal{T}_Y$, since, by convention, $s_v(Z) = -\infty$ for $Z \notin \mathcal{F}_v$. □

Given a POF $\mathcal{P}$ that covers the linear orders on N, the Osd problem can be solved by running Algorithm 5 for each $P \in \mathcal{P}$. The time and space requirements are bounded as follows.

Theorem 4.14 *Given a POF $\mathcal{P}$ that covers the linear orders on N and explicit input, the* OSD *problem can be solved in $O\big(\big[\sum_{P\in\mathcal{P}}(|\mathcal{I}(P)| + F\tau(n))\big]n^2\big)$ time and $O\big(\big[\max_{P\in\mathcal{P}}|\mathcal{I}(P)| + F\big]n\big)$ space, provided that each node has at most F possible parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Proof. By Theorem 4.5 and Lemma 4.13 it suffices to run Algorithm 5 for each $P \in \mathcal{P}$.

The time requirement of Algorithm 5 is dominated by the `for`-loop on line 5. Given Y, the set $\check{Y}$ can be constructed in $O(n^2)$ time by removing from Y each element that has a successor in Y. Thus, the contribution of line 6 in the total time requirement is $O(|\mathcal{I}(P)|n^2)$.

We then analyze the time requirement of the line 8, for fixed v. By Proposition 3.4, the maximization of the local scores over $\mathcal{T}_Y \cap \mathcal{F}_v$ can be done in $O(|\mathcal{T}_Y \cap \mathcal{F}_v|n\tau(n))$ time. Since, by Lemma 4.12, families $\mathcal{T}_Y$ are disjoint for different $Y \in \mathcal{I}(P)$, the total contribution to the time requirement is proportional to $|\mathcal{F}_v| \le F$, for each v. Because the loop in line 7 is executed $|\mathcal{I}(P)|$ times and line 8 n times for every execution of line 7, the total time requirement of line 8 is $O(|\mathcal{I}(P)|n^2 + F\tau(n)n^2)$.

Combining the time bounds of lines 6 and 8 and summing over all members of $\mathcal{P}$ yields the bound $O([\sum_{P\in\mathcal{P}}(|\mathcal{I}(P)| + F\tau(n))]n^2)$.

The space requirement for a given partial order P is $O(|\mathcal{I}(P)|n)$, since, by Lemma 4.11, the values $g^P[Y]$ and $\hat{s}_v[Y]$ are needed only for $Y \in \mathcal{I}(P)$. Therefore, the total space requirement is $O([\max_{P\in\mathcal{P}}|\mathcal{I}(P)| + F]n)$. □

Notice that the results above apply to any POF $\mathcal{P}$ that covers the linear orders on N. Later in Chapter 5 we will address the issue of choosing an "efficient" POF. It should also be noted that if the number of possible parent sets F is larger than $|\mathcal{I}(P)|$, the space requirement is dominated by the input size. Thus, when $|\mathcal{I}(P)|$ is small and one wants to use at most $O^*(|\mathcal{I}(P)|)$ space, it is essential to restrict the number of possible parent sets; we will discuss this issue in more detail in Section 5.4. Another way to address this issue is to solve the OSD problem with implicit input. We see that the results in Theorem 4.14 are easily adapted for implicit input by adding the factor $\delta(n)$ whenever a local score is needed and removing the size of the input from the space requirement. We get the following.

Theorem 4.15 *Given a POF $\mathcal{P}$ that covers the linear orders on N and implicit input, the* Osd *problem can be solved in $O\big(\big[\sum_{P\in\mathcal{P}}(|\mathcal{I}(P)| + F\delta(n)\tau(n))\big]n^2\big)$ time and $O\big(\big[\max_{P\in\mathcal{P}}|\mathcal{I}(P)|\big]n\big)$ space, provided that each node has at most F possible parent sets, which form a downward-closed family, the membership in the possible parent sets can be evaluated in $\tau(n)$ time and a local score can be computed from data in $\delta(n)$ time.*

4.4 The Fp Problem

We continue with dynamic programming over partial orders by adapting it to the Fp problem. Adapting the dynamic programming algorithm can be done straightforwardly in similar fashion as for the Osd problem in the previous section. Computing the local scores in limited space is, however, a more complicated task.

Given a partial order P we get a recursion $g^P(\emptyset) = 1$ and for every nonempty ideal Y of P, let

$$g^P(Y) = \sum_{v\in Y : Y\setminus\{v\}\in\mathcal{I}(P)} \alpha_v(Y\setminus\{v\})g^P(Y\setminus\{v\}),$$

where $\alpha_v(Y\setminus\{v\})$ is the zeta transform of the local scores $\beta_v(Y\setminus\{v\})$. Given an exact cover $\mathcal{P}$, by Theorem 4.5 the Fp problem can be solved by computing $\sum_{P\in\mathcal{P}} g^P(N)$.

The following theorem summarizes the time and space bounds for Fp. Note that in the proof of the theorem we will use a result about computation of a fast sparse zeta transform; this result will be proved in Section 4.4.1.

Theorem 4.16 *Given a POF $\mathcal{P}$ that exactly covers the linear orders on N and explicit input, the* Fp *problem can be solved in $O\big(\sum_{P\in\mathcal{P}}\big[|\mathcal{I}(P)| + F\tau(n)\big]n^2\big)$ time and $O\big(\big[\max_{P\in\mathcal{P}}|\mathcal{I}(P)| + F\big]n\big)$ space, provided that each node has a nonzero score in at most F parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Proof. Suppose we precompute the values $\alpha_v(Y\setminus\{v\})$, for $Y\in\mathcal{I}(P)$. By Theorem 4.20, the fast sparse zeta transform computes $\alpha_v(Y\setminus\{v\})$ for a fixed v and for all $Y\in\mathcal{I}(P)$ in $O((|\mathcal{I}(P)|+F\tau(n))n)$ time and $O(|\mathcal{I}(P)|+F)$ space, provided that each node has a nonzero score in at most F parent sets, which form a downward-closed family and a membership on the family can be determined in $\tau(n)$ time. Thus, computing and storing values of $\alpha_v(Y\setminus\{v\})$ for all $v\in N$ and $Y\in\mathcal{I}(P)$ requires $O((|\mathcal{I}(P)| + F\tau(n))n^2)$ time and $O((|\mathcal{I}(P)| + F)n)$ space in total.

By Proposition 4.4, g^P can be computed in $O^*(|\mathcal{I}(P)|)$ time and space given that local scores can be evaluated in polynomial time. Since the local scores are precomputed and can be accessed in constant time, computation of $g^P(N)$ takes $O(n|\mathcal{I}(P)|)$ time and $O(|\mathcal{I}(P)|)$ space. Thus, combining these time and space requirements with the time required to precompute the values $\alpha_v(Y \setminus \{v\})$, we have the time and space requirements $O((|\mathcal{I}(P)| + F\tau(n))n^2)$ and $O((|\mathcal{I}(P)| + F)n)$, respectively.

The $g^P(N)$ has to be computed for all $P \in \mathcal{P}$. Thus, in total the problem can be solved in $O(\sum_{P \in \mathcal{P}}[|\mathcal{I}(P)| + F\tau(n)]n^2)$ time and $O([\max_{P \in \mathcal{P}} |\mathcal{I}(P)| + F]n)$ space. □

This theorem states the time and space requirement of computing the posterior probability of any fixed arc set. Later, in Section 4.4.2, we will show that the posterior probabilities of all $n(n-1)$ arcs can be computed simultaneously without increasing the time and space requirement (more than a small constant).

4.4.1 Fast Sparse Zeta Transform

Motivated by Theorem 4.16, we consider a sparse zeta transform, in which the values of the zeta transform $(h\zeta)(Y)$ are needed only for the ideals of a partial order P. The goal is to compute the zeta transform for all $Y \in \mathcal{I}(P)$ in time proportional to $n|\mathcal{I}(P)|$. In general, this is not possible as $h(X)$, $X \subseteq N$ might already have 2^n different values. However, we will show that the goal can be achieved assuming $h(X)$ is nonzero on at most $|\mathcal{I}(P)|$ sets, which form a downward-closed family.

Previously, there has been some work on zeta transforms in a sparse setting. Koivisto [62] has shown that k-truncated zeta transform [65], that is, a zeta transform in which only sets of size k or less contribute to the sum, can be computed in $O(k2^n)$ time. As the time requirement is still proportional to 2^n, this approach is not feasible for our purposes. Further, Björklund et al. [10] have shown that a zeta transform for sets belonging to a set family $\mathcal{F}$ of subsets of N can be computed using a trimmed zeta transform algorithm in $O^*(|{\uparrow}\mathcal{F}|)$ time, where ${\uparrow}\mathcal{F}$ is the upper closure of $\mathcal{F}$, that is, ${\uparrow}\mathcal{F}$ consists of the supersets of the members of $\mathcal{F}$. Unfortunately, $\emptyset$ is an ideal of any P, which yields the size of 2^n for the upper closure of the set of ideals of any partial order P.

Since the existing methods are not feasible for our purposes, we present a fast sparse zeta transform algorithm, which evaluates the zeta transform only at the ideals of the given partial order. Note, that this generalization subsumes the fast zeta transform algorithm when P is the trivial partial order $\{xx : x \in N\}$. The fast sparse zeta transform algorithm consists of two nested summations. The outer summation works on ideals $Y \in \mathcal{I}(P)$ and the inner summation gathers the terms needed for each Y.

We split the zeta transform in nested summations as follows. Let

$$h'(Y) = \sum_{X \in \mathcal{T}_Y} h(X)\ , Y \in \mathcal{I}(P).$$

Then, by Lemmas 4.11 and 4.12,

$$(h\zeta)(Y) = \sum_{X \subseteq Y : X \in \mathcal{I}(P)} h'(X)$$

for all $Y \in \mathcal{I}(P)$. Accordingly, we will compute $(h\zeta)$ in two phases. First, given h we evaluate h' at all ideals of P; second, given h' we evaluate $(h\zeta)$ at all ideals of P. The first phase is computationally straightforward as the tails $\mathcal{T}_Y$ are disjoint, and thus $h'(Y)$ can be computed independently for each Y. The second phase is less straightforward.

To compute the second phase, we modify Algorithm 2. A natural attempt to modify the fast zeta transform to operate on ideals of a given partial order P would be to change the `for`-loop at line 5 to run over sets $Y \in \mathcal{I}(P)$ and the condition at line 6 to form "$j \in Y$ and $Y \setminus \{j\} \in \mathcal{I}(P)$". However, the next example shows that this modification is not sufficient to guarantee the correct result.

Example 6. Consider a set $N = \{1, 2, 3\}$ and a partial order $P = \{11, 22, 32, 33\}$. The set of ideals $\mathcal{I}(P)$ consists of the sets $\emptyset$, $\{1\}$, $\{3\}$, $\{1, 3\}$, $\{2, 3\}$, and $\{1, 2, 3\}$. Figure 4.4 shows the computations conducted by a zeta transform algorithm modified as described in the previous paragraph. We can see that the result is incorrect. Clearly, $\emptyset$ and $\{1\}$ are subsets of $\{1, 2, 3\}$ but the values $\beta_v(\emptyset)$ and $\beta_v(\{1\})$ do not contribute to $\alpha_v(\{1, 2, 3\})$ (There is no directed path from the boxes corresponding to sets $\emptyset$ and $\{1\}$ at the topmost level to the box corresponding to $\{1, 2, 3\}$ at the bottommost level). Further, the value $\beta_v(\emptyset)$ does not contribute to $\alpha_v(\{2, 3\})$. ◇

The previous example confirms that we need to further modify Algorithm 2. Fortunately, a simple trick renders the algorithm to work correctly. While Algorithm 2 splits the transform into a sequence of n transforms in an arbitrary order, here we need a particular order. To this end, let σ be a permutation on N such that if $\sigma_i\sigma_j \in P$ then $i < j$; in other words, σ is a linear extension of P. Now the fast sparse zeta transform can be computed by Algorithm 6.

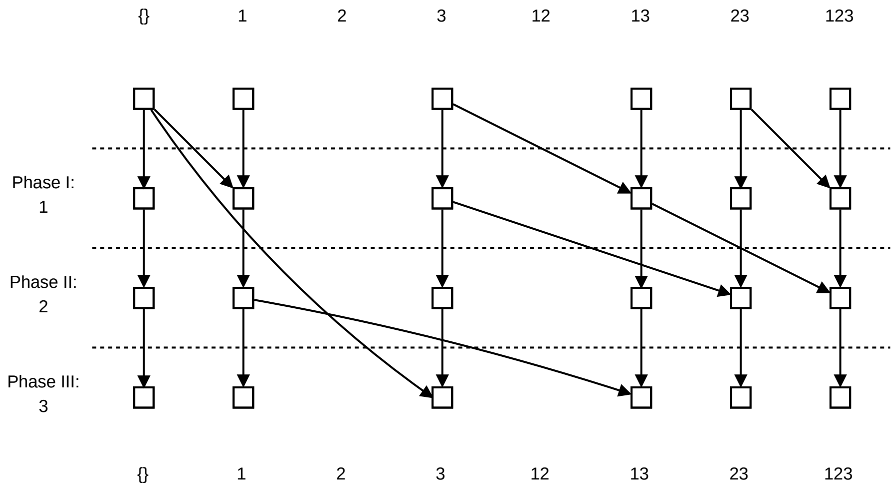


Figure 4.4: Incorrect sparse zeta transform given partial order $P = \{11, 22, 32, 33\}$.

Lemma 4.17 *Algorithm 6 works correctly.*

Proof. Assume that σ is a linear extension of P. We show by induction on j that if Y is an ideal of P, then the computed function satisfies

$$h_j(Y) = \sum_{X \in (Y)_j} h'(X),$$

where we use the shorthand

$$(Y)_j = \{I \in \mathcal{I}(P) : Y \cap \{\sigma_{j+1}, \sigma_{j+2}, \ldots, \sigma_n\} \subseteq I \subseteq Y\};$$

This will suffice, since $(Y)_n$ consists of all the subideals of Y.

We proceed by induction on j and the size of the ideal Y. (i) Trivially $(Y)_0 = \{Y\}$. Therefore, $h_0(Y) = h'(Y) = \sum_{X \in (Y)_0} h'(X)$, as claimed. (ii) Assume then that $h_{j-1}(Y) = \sum_{X \in (Y)_{j-1}} h'(X)$.

First, consider the case $\sigma_j \notin \check{Y}$. Now, if $\sigma_j \notin Y$, then $(Y)_j = (Y)_{j-1}$ because $Y \cap \{\sigma_{j+1}, \ldots, \sigma_n\} = Y \cap \{\sigma_j, \ldots, \sigma_n\}$. Thus $h_j(Y) = h_{j-1}(Y)$, as correctly computed at line 10.

If, on the other hand, $\sigma_j \in Y$, then $\sigma_j \notin \check{Y}$ implies that σ_j is not maximal in Y. This means that there exists another maximal element

Algorithm 6 The fast sparse zeta transform.

```
Input: partial order $P$, $h'(X)$ for $X \in \mathcal{I}(P)$.
Output: $(h\zeta)(Y)$ for $Y \in \mathcal{I}(P)$.
 1: Find a permutation $\sigma$ that is a linear extension of $P$
 2: for $Y \in \mathcal{I}(P)$ do
 3:    $h_0[Y] \leftarrow h'(Y)$.
 4: end for
 5: for $j \leftarrow 1, \ldots, n$ do
 6:    for $Y \in \mathcal{I}(P)$ do
 7:       if $\sigma_j \in \check{Y}$ then
 8:          $h_j[Y] \leftarrow h_{j-1}[Y] + h_{j-1}[Y \setminus \{\sigma_j\}]$.
 9:       else
10:          $h_j[Y] \leftarrow h_{j-1}[Y]$.
11:       end if
12:    end for
13: end for
14: return $h_n[Y]$ for all $Y \in \mathcal{I}(P)$.
```

$\sigma_i \in \check{Y}$ with $\sigma_j\sigma_i \in P$. Because σ is a linear extension of P, we have $j < i$. Now suppose $(Y)_j$ contains an ideal I that is not contained in $(Y)_{j-1}$. Then it must be that $\sigma_j \notin I$. However, $\sigma_i \in I$ because $\sigma_i \in Y$ and $i > j$. This is a contradiction, for $\sigma_j\sigma_i \in P$ and I is an ideal. Thus $(Y)_j = (Y)_{j-1}$, and so $h_j(Y) = h_{j-1}(Y)$, as correctly computed at line 10.

Second, consider the case $\sigma_j \in \check{Y}$, thus $\sigma_j \in Y$. Then $Y \setminus \{\sigma_j\}$ is an ideal. To prove the correctness of line 8, it suffices, by the induction assumption, to show that $(Y)_{j-1}$ and $(Y \setminus \{\sigma_j\})_{j-1}$ are disjoint and their union is $(Y)_j$. To this end, it suffices to observe that $(Y)_{j-1}$ consists of all subideals $I \supseteq Y \cap \{\sigma_j, \sigma_{j+1}, \ldots, \sigma_n\}$ of Y that do contain σ_j, whereas $(Y \setminus \{\sigma_j\})_{j-1}$ consists of all subideals $I \supseteq (Y \setminus \{\sigma_j\}) \cap \{\sigma_j, \sigma_{j+1}, \ldots, \sigma_n\}$ of Y that do not contain σ_j. □

The following example illustrates Algorithm 6.

Example 7. Let us again consider a set $N = \{1, 2, 3\}$ and a partial order $P = \{11, 22, 32, 33\}$. Figure 4.5 shows the summation when the sparse zeta transform is computed by Algorithm 6. The figure clearly shows that, for any ideals $X \subseteq Y$, there is exactly one directed path from the input value of X to the output level of Y. This means that each subideal contributes to the summation on its superideal exactly once and that the algorithm works correctly. ◇

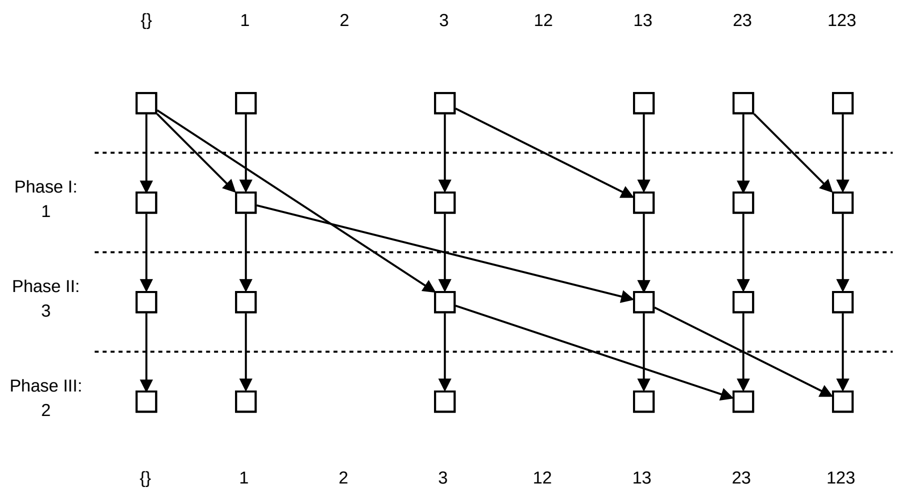


Figure 4.5: The fast sparse zeta transform given partial order $P = \{11, 22, 32, 33\}$ computed by Algorithm 6.

Let us analyze the time and space requirement of Algorithm 6 under the assumption that the values of the input function can be accessed in constant time. At line 1, the permutation σ can be found by topologically sorting the DAG $P \backslash \{xx : x \in N\}$. Topological sorting can be done in $O(n^2)$ time [60, p. 261–268]. The `for`-loop at lines 2-4 can clearly be computed in $O(|\mathcal{I}(P)|)$ time. In the `for`-loop starting at line 6, at most $|\mathcal{I}(P)|$ additions and substitutions are performed in each of the n phases. Assuming that the evaluation of $\sigma_j \in \check{Y}$ given $\check{Y}$ can be done in constant time, we get the total time requirement is $O(n|\mathcal{I}(P)|)$. However, constructing $\check{Y}$ for all sets $Y \in \mathcal{I}(P)$ seems to take $O(n|\mathcal{I}(P)|)$ time in total. Next we show that the sets $\check{Y}$ do not have to be constructed explicitly.

To this end, for every σ_j define the set

$$\begin{aligned} V_j \quad = \quad & \{v \in N : \sigma_j v \in P \text{ and } u \in N \text{ such that } \sigma_j u \in P \text{ and } uv \in P \\ & \text{does not exists}\}. \end{aligned}$$

Intuitively, V_j consists of the nodes that are "one step larger" than σ_j. The sets V_j can be constructed straightforwardly. First, one computes the transitive reduction[2] of the DAG obtained from P by removing self-loops.

[2]The transitive reduction of a DAG A is the minimal DAG A' such that the transi-

Then, the set V_j consists of the children of j in the transitive reduction. The transitive reduction of a DAG can be determined in $O(n^2)$ time [97]. Given the transitive reduction, each V_j can be determined in $O(n)$ time and thus V_j for all $j \in N$ can be computed in $O(n^2)$ time (and space).

Notice that $\sigma_j \in \check{Y}$ for $Y \in \mathcal{I}(P)$ if and only if $\sigma_j \in Y$ and $v \notin Y$ for any $v \in V_j$. We conclude that to determine $\sigma_j \in \check{Y}$ it is sufficient to do at most $|V_j| + 1$ set membership tests, that is, for each ideal Y we first test whether $\sigma_j \in Y$ and if the answer is affirmative we test whether $v \notin Y$ for all $v \in V_j$ in some predetermined order. Once we find a negative test we stop as we know that $\sigma_j \notin \check{Y}$; if all tests are affirmative, we know that $\sigma_j \in \check{Y}$. Unfortunately, $|V_j|$ can be as large as $n-1$ and thus we could need $O(n)$ tests. Next lemma shows that on average we need at most a constant number of tests per ideal.

Lemma 4.18 *Given a partial order P on N, $\sigma_j \in N$ and the set V_j, a test whether $\sigma_j \in \check{Y}$ can be tested for all $Y \in \mathcal{I}(P)$ in $O(|\mathcal{I}(P)|)$ time.*

Proof. More than one set membership test is needed only for ideals Y that contain σ_j and all of its predecessors in P; denote the set of σ_j and all of its predecessors by W_{σ_j}. So let us consider ideals $Y \in \mathcal{I}(P)$ such that $W_{\sigma_j} \subseteq Y$. Let the sequence $v_1 v_2 \ldots v_{|V_j|}$, where $v_i \in V_j$, be the order, in which the tests on members of V_j are conducted. Next we show by induction on the members of the sequence that the probability of a negative test result given the previous tests are positive is at least $1/2$ for each test. Now if Y with $W_{\sigma_j} \subseteq Y$ is an ideal, then also $Y \cup \{v_1\}$ is an ideal. We also note that if $Y \cup X$, $v_1 \notin X$ is an ideal, then also $Y \cup X \cup \{v_1\}$ is an ideal. Therefore, the test on v_1 is negative at for at least half of the considered sets. Now suppose that tests for i first members of V have been affirmative, that is, we are considering ideals Y such that $W_{\sigma_j} \subseteq Y \subseteq N \setminus \{v_1, v_2, \ldots, v_i\}$. Again, if Y is an ideal then also $Y \cup \{v_{i+1}\}$ is an ideal and the test on v_{i+1} is negative for at least half of the considered sets. Thus, for sets $W_{\sigma_j} \subseteq Y$ we have to conduct on average at most $\sum_{i=1}^{|V_j|} i(1/2)^i < (1/2)/(1-1/2)^2 = 2$ tests on members of V_j. Thus, in total we conduct at most $3|\mathcal{I}(P)|$ set membership tests. □

The following theorem summarizes the bounds for the time and space requirement.

tive closure of A equals to the transitive closure of A'. For finite DAGs, the transitive reduction is unique.

Theorem 4.19 *Algorithm 6 evaluates the sparse zeta transform on a partial order P in $O(|\mathcal{I}(P)|n)$ time and $O(|\mathcal{I}(P)|)$ space assuming that the values $h'(Y)$ for $Y \in \mathcal{I}(P)$ can be evaluated in constant time.*

Let us now analyze the time requirement of the first phase, that is, computing $h'(Y)$ given a partial order P for all $Y \in \mathcal{I}(P)$. In the first phase, h is evaluated on every subset of N if no assumptions are made. However, assume that h vanishes, that is, equals zero at all sets that are not in $\mathcal{F}$, where $\mathcal{F}$ is some downward-closed set family with F sets. Then by Proposition 3.4, we can compute the first phase in $O(Fn\tau(n))$ time. Thus the total running time is determined by the larger of the two bounds as follows.

Theorem 4.20 *Suppose $h(X)$ vanishes if $X \notin \mathcal{F}$, where $\mathcal{F}$ is a downward-closed set family with at most F members and a membership of $\mathcal{F}$ can be determined in $\tau(n)$ time. Then the zeta transform $(h\zeta)(Y)$ can be computed for all ideals Y of a partial order P in $O((|\mathcal{I}(P)| + F\tau(n))n)$ time and $O(|\mathcal{I}(P)| + F)$ space.*

Note that if $\mathcal{F}$ consists of sets of size at most k the time requirement becomes $O\left((|\mathcal{I}(P)| + \sum_{i=0}^{k}\binom{n}{i})n\right)$.

We end this section by considering a "dual" of the above-described restricted zeta transform. This transform turns out to be handy in Section 4.4.2. Let P be a partial order on N and h a function from subsets of N to reals. We define the *up-zeta transform* of h by

$$(g\zeta')(Y) = \sum_{X \supseteq Y} g(X) \,, Y \subseteq N.$$

The following lemma shows that the computation of the up-zeta transform reduces to the computation of the ordinary zeta transform. We denote by $\bar{X} = N \setminus X$ for any $X \subseteq N$.

Lemma 4.21 *Given a set function h,*

$$(h\zeta)(Y) = (g\zeta')(\bar{Y}),$$

where g is defined by $g(\bar{Y}) = h(Y)$.

Proof. It suffices to notice that

$$\begin{aligned}
(h\zeta)(Y) &= \sum_{X \subseteq Y} h(X) \\
&= \sum_{X \subseteq Y} g(N \setminus X) \\
&= \sum_{X \supseteq N \setminus Y} g(X) \\
&= (g\zeta')(\bar{Y}).
\end{aligned}$$

The first identity holds by the definition of the zeta transform; the second by the definition of g; the third one by the fact that X is a superset of $N \setminus Y$ if and only if $N \setminus X$ is a subset of Y; the fourth one by the definitions of the up-zeta transform and $\bar{X}$. □

Further, we define the *sparse up-zeta transform* of h by

$$(g{\zeta'}^P)(Y) = \sum_{X \supseteq Y : X \in \mathcal{I}(P)} g(X) \, , Y \subseteq N.$$

In words, the summation is over the superideals of Y. Note that $(g{\zeta'}^P)(Y)$ is defined also for sets Y that are not ideals of P. To compute the sparse up-zeta transform, we define a set

$$\hat{Y} = \{x \in N : xy \in P \text{ for some } y \in Y\}.$$

The following lemma shows that $\hat{Y}$ is an ideal and that the sparse up-zeta transform of Y equals the sparse up-zeta transform of $\hat{Y}$.

Lemma 4.22 *Given a partial order P, (i) the set $\hat{Y}$ is an ideal of P and (ii) $(g{\zeta'}^P)(Y) = (g{\zeta'}^P)(\hat{Y})$.*

Proof. (i) By the definition of an ideal, $y \in \hat{Y}$ and $xy \in P$ implies $x \in \hat{Y}$. Assume that such a y exists. If $y \in Y$, then by the definition of $\hat{Y}$, indeed $x \in \hat{Y}$. On the other hand, if $y \notin Y$ then, by transitivity and the definition of $\hat{Y}$, there exists $z \in Y$ such that $yz \in P$ and $xz \in P$ which implies that $x \in \hat{Y}$.

(ii) It is sufficient to show that if $X \supseteq Y$ is an ideal of P, then necessarily $X \supseteq \hat{Y}$. Suppose on the contrary that there exists $X \in \mathcal{I}(P)$ with $Y \subseteq X \subset \hat{Y}$. Then, by the definition of an ideal, $y \in Y$ and $xy \in P$ implies that $x \in X$; but this is the definition of $\hat{Y}$: contradiction. We conclude that $(g{\zeta'}^P)(Y) = (g{\zeta'}^P)(\hat{Y})$. □

Note that $\hat{Y} = X$ for all $Y \in \mathcal{T}_X$. By Lemma 4.22, it is sufficient to compute $(g\zeta'^P)$ over the ideals of P only.

The fast sparse up-zeta transform is analogous to the fast sparse (down-)zeta transform. Rather than modifying the construction we resort to the notion of "complementary symmetry". To this end, we denote by $\bar{P} = \{xy : yx \in P\}$ for any partial order P on N. We get the following lemma.

Lemma 4.23 *X is an ideal of P if and only if $\bar{X}$ is an ideal of $\bar{P}$.*

Proof. "If": Suppose on the contrary that $\bar{X}$ is an ideal of $\bar{P}$ but X is not an ideal of P. Now by the definition of an ideal, there must exist $x \in X$ such that $yx \in P$ and $y \notin X$. But by the definitions of $\bar{X}$ and $\bar{P}$, $y \in \bar{X}$, $xy \in \bar{P}$ and $x \notin \bar{X}$, that is, $\bar{X}$ is not an ideal of $\bar{P}$: contradiction.

"Only if": Suppose on the contrary that X is an ideal of P but $\bar{X}$ is not an ideal of $\bar{P}$. Now by the definition of an ideal, there must exist $x \in \bar{X}$ such that $yx \in \bar{P}$ and $y \notin \bar{X}$. But by the definitions of $\bar{X}$ and $\bar{P}$, $y \in X$, $xy \in P$ and $x \notin X$, that is, X is not an ideal of P: contradiction. □

The following theorem states the fast sparse up-zeta transform can be computed with the fast sparse (down-)zeta transform.

Theorem 4.24 *Given a set function g and a partial order P, $(g\zeta'^P)(Y) = h_n[N \setminus \hat{Y}]$, where $h_n[N \setminus \hat{Y}]$ is the output of Algorithm 6 ran with a input consisting of a partial order $\bar{P}$ and $h'(Y) = g(N \setminus Y)$ for $Y \in \mathcal{I}(\bar{P})$.*

Proof. By Lemma 4.17, $h_n[N \setminus Y]$ is the sum of the values $g(N \setminus Y)$ over the subideals of $N \setminus Y$ with respect to a partial order $\bar{P}$. By Lemmas 4.21 and 4.23, the sum of the values $g(N \setminus Y)$ over the subideals of $N \setminus Y$ with respect to a partial order $\bar{P}$ equals the sparse up-zeta transform of g evaluated at $\hat{Y}$. Finally, by Lemma 4.22(ii), the sparse up-zeta transform of g evaluated at $\hat{Y}$ equals $(g\zeta'^P)(Y)$. □

4.4.2 Posterior Probabilities of All Arcs

In Section 4.4 we showed how to compute the posterior probability of any arc set. Maybe the most important arc sets are single arcs as it is often handy to present the posterior probabilities for all $n(n-1)$ arcs. A straightforward method would compute the probabilities independently for every arc. However, a faster forward–backward algorithm [62] avoids the multiplicative factor $n(n-1)$ in running time altogether. Next, we adapt the forward–backward algorithm to run with limited space.

Let P be a partial order on N. Recall that a forward function g^P satisfies the recurrence $g^P(\emptyset) = 1$ and

$$g^P(Y) = \sum_{v \in Y : Y \setminus \{v\} \in \mathcal{I}(P)} \alpha_v(Y \setminus \{v\}) g^P(Y \setminus \{v\}),$$

where Y is an ideal of P. Analogously, we define a *backward function* b^P by $b^P(\emptyset) = 1$ and

$$b^P(X) = \sum_{v \in X : N \setminus (X \setminus \{v\}) \in \mathcal{I}(P)} \alpha_v(N \setminus X) b^P(X \setminus \{v\}),$$

where X is nonempty and $N \setminus X$ is an ideal of P. Intuitively, the backward function computes the contribution of the nodes in X given that they are the $|X|$ last nodes in the order. Then we combine the forward and backward functions into

$$\gamma_v^P(A_v) = \sum_Y q_v(Y) g^P(Y) b^P(N \setminus Y \setminus \{v\}),$$

where A_v is a subset of $N \setminus \{v\}$ and Y runs over all ideals of P with $A_v \subseteq Y \subseteq N \setminus \{v\}$. In words, if a node v has parents A_v, then $\gamma_v(A_v)$ computes the contribution of the other nodes. Naturally, a node set $Y \supseteq A_v$ must precede v and then the rest $N \setminus Y \setminus \{v\}$ must follow v.

We now express the marginal posterior probability of an arc uv as a weighted average of terms $\gamma_v^P(A_v)$. To this end, we observe that

$$\sum_{L \supseteq P} \prod_{t \in N} \alpha_t(L_t) = \sum_{A_v \subseteq N \setminus \{v\}} \beta_v(A_v) \gamma_v^P(A_v).$$

We observe that $\beta_v(A_v)\gamma_v^P(A_v)$ is the contribution of all DAGs compatible with P in which A_v (and no one else) are the parents of v. Thus, if $\mathcal{P}$ is a POF that exactly covers the linear orders on N, then, by the equation (2.4), the joint probability of data D and a feature f can be written as

$$Pr(D, f) = \sum_{P \in \mathcal{P}} \sum_{A_v \subseteq N \setminus \{v\}} \beta_v(A_v) \gamma_v^P(A_v). \tag{4.5}$$

Notice that under this assumption, the choice of an arc uv affects only the terms $\beta_v(A_v)$. Clearly, $\beta_v(A_v)$ is nonzero only when $u \in A_v$. The terms $\gamma_v^P(A_v)$ can thus be precomputed assuming the trivial feature $f \equiv 1$.

Let us analyze the time requirement of the computation of the posterior probabilities of arcs using the equation (4.5). First, note that for a fixed partial order P, the values $\alpha_v(Y \setminus \{v\})$ for all nodes v and ideals $Y \setminus \{v\}$ of P can be computed using the fast sparse zeta transform in $O(n^2|\mathcal{I}(P)|)$ time. Second, note that the forward and backward functions can be computed using a straightforward recursion in $O(n|\mathcal{I}(P)|)$ time. Third, the values of $\gamma_v^P(Y)$ for a fixed v and all ideals Y of P can be computed using the fast sparse up-zeta transform in $O(n|\mathcal{I}(P)|)$ time and thus for all v in $O(n^2|\mathcal{I}(P)|)$ time. Finally, given set Y, set $\hat{Y}$ can be computed in $O(n^2)$ time and thus given the values $\gamma_v^P(Y)$ for all ideals Y of P, the inner sum in equation (4.5) can be computed in $O(F\tau(n)n^2)$ time assuming that the score $\beta_v(A_v)$ vanishes if $A_v \notin \mathcal{F}_v$. Thus, for fixed P, the total time requirement is $O(n^2|\mathcal{I}(P)|)$ provided that $F \leq \mathcal{I}|(P)|$. Clearly all steps can be computed in $O(n|\mathcal{I}(P)|)$ space. Next theorem concludes the previous analysis.

Theorem 4.25 *Given a POF $\mathcal{P}$ that exactly covers the linear orders on N and explicit input, the* FP *problem for all arc features can be solved in $O(n^2\tau(n)\sum_{P\in\mathcal{P}}|\mathcal{I}(P)|)$ time and $O(n \max_{P\in\mathcal{P}}|\mathcal{I}(P)|)$ space, provided that the local score $\beta_v(A_v)$ vanishes for all $A_v \notin \mathcal{F}_v$, $\mathcal{F}_v$ consists of at most F sets, which form a downward-closed family, the membership in $\mathcal{F}_v$ can be evaluated in $\tau(n)$ time and $F \leq |\mathcal{I}(P)|$ for all $v \in N$ and $P \in \mathcal{P}$.*

Recall that Theorem 4.16 stated that the posterior probability of any modular structural feature (including a single arc) can be computed in $O(n^2\tau(n)\sum_{P\in\mathcal{P}}|\mathcal{I}(P)|)$ time and $O(n^2\sum_{P\in\mathcal{P}}|\mathcal{I}(P)|)$ space; thus the posterior probability of all $n(n-1)$ arcs can be computed in essentially the same time and space as the posterior probability of a single arc.

Chapter 5

Parallel Bucket Orders

In the previous chapter we presented the partial order approach for solving permutation problems. The time and space requirements were presented in terms of the size of a POF $\mathcal{P}$ and the number of ideals of partial orders $P \in \mathcal{P}$. In order to trade space against time "efficiently", the choice of $\mathcal{P}$ plays a crucial role. In this chapter we address this issue and concentrate on a particular class of partial orders called parallel bucket orders.

We start this chapter by introducing a simple example of parallel bucket orders called a pairwise scheme. The idea of the pairwise scheme is to choose p distinct pairs of elements and fix a linear order within each pair. The pairwise scheme illustrates many characteristics of the parallel bucket orders and hence seems to be an intuitively good starting point for our investigation of parallel bucket order schemes. Then in Section 5.2 we formally define parallel bucket orders and give some elementary results. In Section 5.3 we present a certain parallel bucket scheme which we call the $13 * 13$ scheme and show that it yields an optimal time–space product (among parallel bucket orders). Finally, we investigate the prospects of the parallel bucket orders in practice in Section 5.4.

5.1 Pairwise Scheme

Formally, the pairwise scheme is represented as follows. Pick $2p \leq n$ distinct elements $u_1, u_2, \ldots, u_p, v_1, v_2, \ldots, v_p \in N$ and fix a partial order P such that either $u_i v_i \in P$ or $v_i u_i \in P$ for all $1 \leq i \leq p$. For an analysis of the number of ideals, consider a fixed partial order P such that $uv \in P$. A set containing neither u nor v, a set containing u but not v, and a set containing both u and v can be ideals of P. However, no set that contains v but not u can be an ideal of P. As all p pairs are independent and any subset of the remaining $n - 2p$ elements can be part of an ideal, the total number of ideals is $3^p 2^{n-2p}$. Further, we note that for each linear order L either $u_i v_i \in L$ or $v_i u_i \in L$ for all i. To cover the linear orders on N we need to consider combinations for each pair, that is, in total 2^p different partial orders. These results can be summarized in the following proposition.

Proposition 5.1 *Polynomial local time permutation problems of bounded degree can be solved in $O^*(2^n(3/2)^p)$ time and $O^*(2^n(3/4)^p)$ space, where $p = 0, 1, \ldots, n/2$.*

Next, we will consider the Osd and Fp problems. Based on Theorems 4.14 and 4.16, we conclude as follows.

Proposition 5.2 *Given explicit input, both the* Osd *problem and the* Fp *problem can be solved in $O([2^n(3/2)^p + F\tau(n)]n^2)$ time and $O([2^n(3/4)^p + F]n)$ space, where $p = 0, 1, \ldots, n/2$, provided that each node has at most F possible parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

In order to guarantee that the number of possible parent sets does not dominate the time and space requirement we need to restrict the number of possible parent sets. By allowing at most k parents for each node, each node has at most $\sum_{i=0}^{k} \binom{n-1}{i}$ possible parent sets. It is well-known (see inequality (A.3) in Appendix A) that $\sum_{i=0}^{\mu n} \binom{n-1}{i} \leq 2^{H(\mu)n}$, where $H(\mu)$ is the binary entropy function and $\mu \leq 1/2$. Now we compare this bound to the space requirement $3^p 2^{n-2p}$ in the most stringent case, at $p = n/2$. We solve μ in $2^{H(\mu)n} \leq 3^{n/2}$, which is equivalent to $H(\mu)n \leq (n/2)\log_2 3$, that is, $H(\mu) \leq (1/2)\log_2 3$. Solving the inequality numerically yields $\mu \leq 0.238$. We summarize as follows.

Proposition 5.3 *Given explicit input, both the* Osd *problem and the* Fp *problem can be solved in $O([2^n(3/2)^p]n^2)$ time and $O([2^n(3/4)^p]n)$ space, where $p = 0, 1, \ldots, n/2$, provided that each node has at most $0.238n$ parents.*

The following example illustrates the pairwise scheme. Here it is convenient to measure the time and space requirements in terms of the number of ideals.

Example 8. Again, consider node set $N = \{a, b, o, t\}$. Now we can have the pairwise scheme with either one or two pairs. Figure 5.1 shows examples of partial orders on N with a linear order for (a) one pair ($p = 1$) and (b) two pairs ($p = 2$) fixed. The corresponding subset lattices are shown in

Figure 5.2. Recall that when dynamic programming over the trivial order which is equivalent to the standard dynamic programming algorithm, we get the time and space requirement of 16 (sets); see Figure 4.2(a). For the pairwise scheme with one fixed pair, we get 12 ideals and thus the space requirement 12; see Figure 5.2(a). As we need to consider 2 such partial orders to cover the linear orders, the total time requirement is 24. For the pairwise scheme with two pairs, we get 9 ideals; see Figure 5.2(b). The cover size is $2 \times 2 = 4$ and thus the total time requirement is 36. $\diamond$

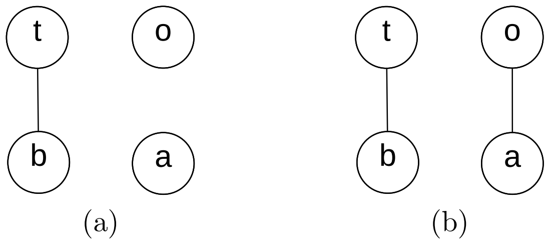


Figure 5.1: Hasse diagrams of partial orders (a) $P_1 = \{aa, bb, bt, oo, tt\}$ and (b) $P_2 = \{aa, ao, bb, bt, oo, tt\}$.

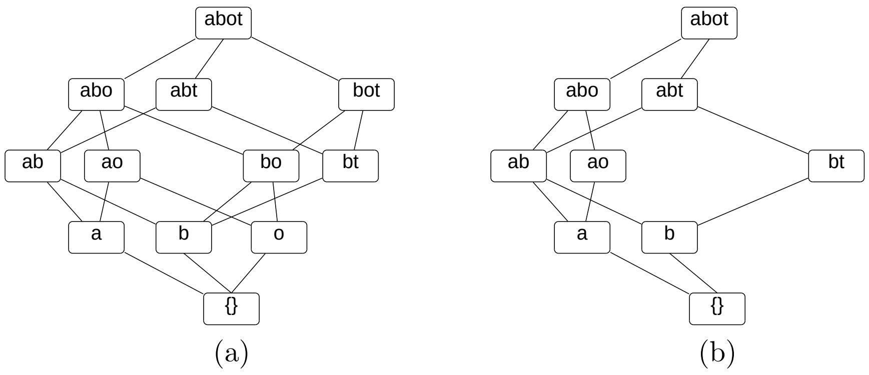


Figure 5.2: Lattices of the ideals of partial orders (a) $P_1 = \{aa, bb, bt, oo, tt\}$ and (b) $P_2 = \{aa, ao, bb, bt, oo, tt\}$.

The time–space product of the pairwise scheme is $\theta = (2^n(3/2)^p 2^n(3/4)^p)^{1/n} = 4(9/8)^{p/n}$. We see that the time–space product is the smaller the less space is saved, that is, when p is small. However, the term $(9/8)^{p/n}$ is always at least 1 and thus the time–space product is always at least 4. The time–space product of the pairwise scheme is maximized when $p = n/2$; the maximum is $4(9/8)^{1/2} \approx 4.2426$ and thus the time–space product is always lower than 4.243. Next we generalize the pairwise scheme and show that the time–space product can be further lowered.

5.2 Parallel Bucket Orders: An Introduction

We will consider parallel bucket orders which are defined as follows. A partial order P is a *parallel composition* of partial orders $P_1, P_2, \ldots, P_k$ if the P_i are pairwise disjoint and their union is P, that is, $\{P_1, P_2, \ldots, P_k\}$ is a partition of P. Given P, the partition becomes unique if each *component* P_i is required to be connected, that is, P_i does not further partition into two nonempty partial orders. A partial order P is a *series composition* of partial orders P_1 and P_2 if $uv \in P$ if and only if $uv \in P_1$, $uv \in P_2$, or $u \in P_1$ and $v \in P_2$. A partial order B on a base-set M is a *bucket order* if M can be partitioned into nonempty sets $B_1, B_2, \ldots, B_\ell$, called *buckets*, such that $xy \in B$ if and only if $x = y$ or $x \in B_i$ and $y \in B_j$ for some $i < j$; the bucket sequence is unique. The bucket order is denoted by $B = B_1 B_2 \ldots B_\ell$ and is said to be of length ℓ and type $|B_1| * |B_2| * \cdots * |B_\ell|$. A partial order P is a *parallel bucket order* if P is a parallel composition of bucket orders.

Example 9. We have already seen several bucket orders and parallel bucket orders. For example, a linear order is a bucket order with n buckets of size 1. The partial orders induced by the two-bucket scheme (Section 3.1.1) are bucket orders with two buckets whose sizes are s and $n - s$. Also the trivial order is a bucket order; all elements are in the same bucket. The pairwise scheme introduced in the previous section operates on parallel bucket orders; every pair is a bucket order with two buckets, each of size 1.

Further, Figure 5.3 shows Hasse diagrams of the bucket order $B = \{a\}\{b, c\}\{d\}$ and the parallel composition of bucket orders $Q_1 = \{h, i\}\{a, b\}$ and $Q_2 = \{j\}\{e, f, g\}\{c, d\}$. A parallel bucket order can be illustrated by a Hasse diagram, in which parallel compositions do not share any lines and the nodes inside each bucket are placed in parallel on one level and a line is drawn between every pair of nodes in consecutive levels. $\diamond$

Bucket orders and parallel bucket orders are a special case of *series–parallel* partial orders. Indeed, the order on each single bucket is a trivial

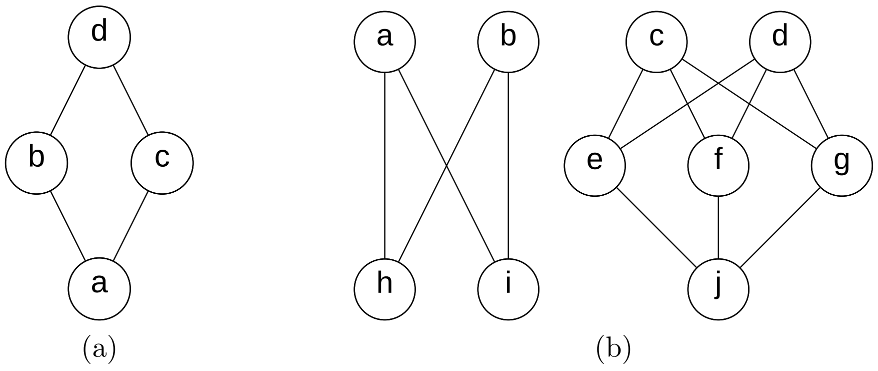


Figure 5.3: (a) A bucket order $B = \{a\}\{b,c\}\{d\}$ and (b) a parallel composition P of bucket orders $B_1 = \{h,i\}\{a,b\}$ and $B_2 = \{j\}\{e,f,g\}\{c,d\}$.

order, which is a parallel composition of singletons. Further, each bucket order is a series composition of buckets. Finally, parallel bucket orders are parallel compositions of bucket orders. Simple rules are known for calculating the number of ideals of a series–parallel partial order; see, for example, Equations 3.1 and 3.2 in Steiner [102] and references therein. The following two lemmas are mere applications of these rules.

Lemma 5.4 *The number of ideals of a bucket order* $B = B_1 B_2 \ldots B_\ell$ *is given by* $|\mathcal{I}(B)| = 1 - \ell + 2^{|B_1|} + 2^{|B_2|} + \cdots + 2^{|B_\ell|}$.

Lemma 5.5 *Let* P *be the parallel composition of partial orders* $P_1, P_2, \ldots, P_k$. *Then the number of ideals of* P *is given by* $|\mathcal{I}(P)| = |\mathcal{I}(P_1)||\mathcal{I}(P_2)| \cdots |\mathcal{I}(P_k)|$.

To construct a POF that exactly covers the linear orders on N and consists of parallel bucket orders, we introduce a notion of reordering. We say that two bucket orders are *reorderings* of each other if they have the same base-set and are of the same type (and length). Further, we say that two parallel bucket orders are reorderings of each other if their connected components can be labeled as $P_1, P_2, \ldots, P_k$ and $Q_1, Q_2, \ldots, Q_k$ such that P_i is a reordering of Q_i for all i. If P is a parallel bucket order, we denote by $\mathcal{P}(P)$ the family of the partial orders that are reorderings of P. We call $\mathcal{P}(P)$ the *equivalence class* of P (with respect to the reordering relation[1]).

[1]The reordering relation is an equivalence relation as it is obviously reflexive, symmetric and transitive.

Next, we show that $\mathcal{P}(P)$ exactly covers the linear orders on N.

Lemma 5.6 *Let P be a parallel bucket order. Then $\mathcal{P}(P)$ exactly covers the linear orders on the base-set of P.*

Proof. Let $P_1, P_2, \ldots, P_k$ be the connected components of P with base-sets $N_1, N_2, \ldots, N_k$, respectively. Let $\sigma = \sigma_1\sigma_2\cdots\sigma_n$ be a linear order on the base-set of P. It suffices to show that there is a unique partial order Q equivalent to P such that σ is an extension of Q.

For each $i = 1, 2, \ldots, k$, we construct a bucket order Q_i on N_i as follows. Let $m_1 * m_2 * \cdots * m_\ell$ be the type of P_i. For $j = 1, 2, \ldots, \ell$ denote $s_j = m_1 + m_2 + \cdots + m_j$, $s_0 = 0$ and $m = s_\ell$. Let $\sigma' = \sigma'_1\sigma'_2\cdots\sigma'_m$ be the induced order $\sigma[N_i]$. Now, let $Q_i = C_1C_2\cdots C_\ell$ with $C_j = \{\sigma'_t : s_{j-1} < t \le s_j\}$. Note that, on one hand, σ' is an extension of Q_i, and on the other hand, any other reordering of Q_i must contain a pair xy with $yx \in \sigma'$. Thus, Q_i is unique.

Finally, let Q be the parallel composition of the bucket orders Q_i. Note that σ is an extension of Q, since if $xy \in Q$ then $xy \in Q_i$ for some i, and hence, $xy \in \sigma$. □

By basic combinatorial arguments we find the number of reorderings of a given parallel bucket order:

Lemma 5.7 *The number of reorderings of a bucket order of type $m_1 * m_2 * \cdots * m_\ell$ is given by $(m_1 + m_2 + \cdots + m_\ell)!/(m_1!m_2!\cdots m_\ell!)$.*

Lemma 5.8 *The number of reorderings of the parallel composition of bucket orders $P_1, P_2, \ldots, P_k$ is given by $p_1p_2\cdots p_k$, where p_i is the number of reorderings of P_i.*

When P is a parallel composition of p bucket orders of type $m_1 * m_2 * \cdots * m_\ell$ we find it convenient to denote the POF $\mathcal{P}(P)$ by $(m_1 * m_2 * \cdots * m_\ell)^p$; this notation is explicit about the combinatorial structure of the POF while ignoring the arbitrariness of the labeling of the elements of N. When the size of the base-set, n, is clear from the context, we may extend the notation $(m_1 * m_2 * \cdots * m_\ell)^p$ to refer to a family $\mathcal{P}(P)$, where P is the parallel composition of p bucket orders of type $(m_1 * m_2 * \cdots * m_\ell)^p$ and one trivial order on the remaining $n - (m_1 + m_2 + \cdots + m_\ell)p$ elements.

Example 10. Consider the bucket order $B = \{a\}\{b,c\}\{d\}$ from the previous example. By Lemma 5.4 it has $1 - 3 + 2^1 + 2^2 + 2^1 = 6$ ideals. Further, B generates an exact cover which by Lemma 5.7 consists of $(1 + 2 + 1)!/(1!2!1!) = 12$ reorderings.

Next, consider the parallel composition P of bucket orders $P_1 = \{h, i\}\{a, b\}$ and $P_2 = \{j\}\{e, f, g\}\{c, d\}$. Bucket order P_1 has $1 - 2 + 2^2 + 2^2 = 7$ and P_2 has $1 - 3 + 2^1 + 2^3 + 2^2 = 12$ ideals. Thus, by Lemma 5.5, P has $7 \times 12 = 84$ ideals. Bucket order P_1 has $4!/(2!2!) = 6$ and P_2 has $6!/(1!3!2!) = 60$ reorderings and thus, by Lemma 5.8, P has in total $6 \times 60 = 360$ reorderings. $\diamond$

Equipped with Lemmas 5.4–5.8, we can calculate the time–space product of the equivalence class of any parallel bucket order. Let P be a parallel composition of bucket orders $P_1, P_2, \ldots, P_k$. Then we have

$$\begin{aligned}\sum_{Q \in \mathcal{P}(P)} |\mathcal{I}(Q)| &= |\mathcal{P}(P)||\mathcal{I}(P)| = \prod_{i=1}^{k} |\mathcal{P}(P_i)||\mathcal{I}(P_i)|,\\ \max_{Q \in \mathcal{P}(P)} |\mathcal{I}(Q)| &= |\mathcal{I}(P)| = \prod_{i=1}^{k} |\mathcal{I}(P_i)|,\end{aligned}$$

and thus

$$\theta(\mathcal{P}(P)) = |\mathcal{P}(P)||\mathcal{I}(P)|^2 = \prod_{i=1}^{k} |\mathcal{P}(P_i)||\mathcal{I}(P_i)|^2.$$

By a *parallel bucket order scheme* we refer to a *set* of reorderings of some parallel bucket order, parameterized by one or more parameters. When the size of the base-set, n, is the only parameter, the scheme is *degenerate*, being just a sequence of POFs, $(\mathcal{P}_n)$, that induces a single algorithm for permutation problems, with some asymptotic time and space complexity bounds. A scheme becomes more interesting when there are additional free parameters, varying the values of which yields, respectively, varying asymptotic time and space complexity bounds. An example of such a bucket order scheme is the two-bucket scheme of Section 3.1.1, which corresponds to POFs $(s * (n - s))^1$; here, the size of the first bucket, s, parameterizes the scheme. Other examples are the pairwise scheme, corresponding to POFs $(1 * 1)^p$, with p as the parameter, and the *generalized two-bucket scheme* defined by POFs $(\lceil m/2 \rceil * \lfloor m/2 \rfloor)^{\lfloor n/m \rfloor}$, with m as the parameter. In the sequel we will identify a bucket order scheme with the form of the corresponding POFs; for example, we may talk about the scheme $(5 * 5)^p$.

In the next section, we will find the parallel bucket order scheme that is optimal with respect to the time–space product.

5.3 On Optimal Time–Space Product

We continue our search for "efficient" POFs by showing that one can achieve a time–space product less than 4 by using parallel bucket orders. As we have seen in the previous section, there is a closed-form expression for the time–space product of the equivalence class of a parallel bucket order. Thus, one can numerically show that out of all parallel bucket order schemes, the $13 * 13$ scheme, that is, the scheme corresponding to a POF $(13 * 13)^{\lfloor n/26 \rfloor}$ yields an optimal time–space product. Next, we will prove this.

For natural numbers n and k with $26k \leq n$, use a shorthand $\mathcal{P}_{n,k}$ for a scheme corresponding to a POF $(13 * 13)^k$. We get the following result.

Theorem 5.9 *Polynomial local time permutation problems can be solved in $O^*((2\kappa^{1/26})^n)$ time and $O^*((2\lambda^{1/26})^n)$ space, where*

$$\begin{aligned} \kappa &= \binom{26}{13}(2^{14}-1)/2^{26} < 2.539055 \times 10^3 , \\ \lambda &= (2^{14}-1)/2^{26} < 2.441258 \times 10^{-4} . \end{aligned}$$

Proof. By Proposition 4.6, permutation problems can be solved in $O^*(\sum_{P \in \mathcal{P}} |\mathcal{I}(P)|)$ time and $O^*(\max_{P \in \mathcal{P}} |\mathcal{I}(P)|)$ space, where $\mathcal{P}$ is an exact cover. By Lemma 5.6, the equivalence class of a parallel bucket order is an exact cover and thus $\mathcal{P}_{n,k}$ is an exact cover.

By applying Lemmas 5.4 and 5.5 for $\mathcal{P}_{n,k}$, we get $\max_{Q \in \mathcal{P}_{n,k}} |\mathcal{I}(Q)| = \lambda^k 2^n$ and further applying Lemmas 5.7 and 5.8 we get $\sum_{Q \in \mathcal{P}_{n,k}} |\mathcal{I}(Q)| = \kappa^k 2^n$.

Moreover, set $k = \lfloor n/26 \rfloor$. Notice that $\kappa^{\lfloor n/26 \rfloor} 2^n \leq \kappa^{n/26} 2^n = (2\kappa^{1/26})^n$ and thus permutation problems can be solved in $O^*((2\kappa^{1/26})^n)$ time. Furthermore, notice that for any $\epsilon > 0$, we have $\lambda^{\lfloor n/26 \rfloor} 2^n \leq \lambda^{n/26-1} 2^n = (\lambda^{1/26}\lambda^{-1/n})^n 2^n < (2\lambda^{1/26})^n + \epsilon$ for sufficiently large n and therefore permutation problems can be solved in $O^*((2\lambda^{1/26})^n)$ space. □

We note that the time and space requirements in the previous theorem are $O^*(2.71^n)$ and $O^*(1.46^n)$, respectively, yielding the time–space product less than 3.9272. This time–space product is lower than the time–space products of the schemes presented earlier. This is also the first time that we encounter a time–space product less than 4. We summarize the time–space product by the following theorem that is immediate by Theorem 5.9.

Theorem 5.10 *Polynomial local time permutation problems have a parallel bucket order scheme with a time–space product $4(\kappa\lambda)^{1/26} < 3.93$.*

Let us show that the $13 * 13$ scheme (Theorems 5.9 and 5.10) is optimal in the sense that it minimizes the time–space product within the class of all parallel bucket order schemes. To get started, let P be a parallel composition of k bucket orders on $N = \{1, 2, \ldots, n\}$. To lower-bound the product $\theta(\mathcal{P}(P))$, define $\psi(m)$ as the minimum of $|\mathcal{P}(B)||\mathcal{I}(B)|^2$ over all bucket orders B on m elements. Before we calculate $\psi(m)$ below, we first note the bound

$$\begin{aligned} \theta(\mathcal{P}(P)) &\geq \prod_{i=1}^{k} \psi(|N_k|) \\ &\geq \min\left\{\psi(m)^{n/m} : m = 1, 2, \ldots, n\right\} . \end{aligned} \tag{5.1}$$

Here the first inequality follows by the definitions of $\theta(\mathcal{P}(P))$ and ψ and the second by $\psi(m) > 0$ and the following elementary observation.

Lemma 5.11 *Let $s_1, s_2, \ldots, s_k \geq 1$ be numbers that sum up to s, and let $\phi(r) > 0$ for any r. Then $\phi(s_1)\phi(s_2)\cdots\phi(s_k) \geq \min\left\{\phi(s_i)^{s/s_i} : i = 1, 2, \ldots, k\right\}$.*

Proof. Suppose the contrary. Then $\prod_i \phi(s_i)$ is less than $\phi(s_j)^{s/s_j}$ for all $j = 1, 2, \ldots, k$ and thus $\prod_i \phi(s_i)^{s_j}$ is less than $\phi(s_j)^s$ for all $j = 1, 2, \ldots, k$. Taking products on both sides yields the contradiction that $\prod_j \prod_i \phi(s_i)^{s_j} = \prod_i \phi(s_i)^s$ is less than $\prod_j \phi(s_j)^s$. □

As $\psi(m)^{1/m}$ lower bounds the time–space product of any bucket order on m elements, it remains to calculate $\psi(m)$ and show that $\psi(m)^{1/m}$ is minimized at $m = 26$. We begin by calculating $|\mathcal{P}(B)||\mathcal{I}(B)|^2$ for a bucket order $B = B_1 B_2 \cdots B_\ell$. If each B_j consists of m_j elements, then $|\mathcal{P}(B)||\mathcal{I}(B)|^2$ is given by

$$\begin{aligned} \theta(m_1, m_2, \ldots, m_\ell) = &\frac{(m_1 + m_2 + \cdots + m_\ell)!}{m_1! m_2! \cdots m_\ell!} \\ &\times (1 - \ell + 2^{m_1} + 2^{m_2} + \cdots + 2^{m_\ell})^2 . \end{aligned}$$

We next show that $\theta(m_1, m_2, \ldots, m_\ell)$ is minimized subject to $m_1 + m_2 + \cdots + m_\ell = m$ either at $\ell = 1$ with $m_1 = m$ or at $\ell = 2$ with $m_1 = \lceil m/2 \rceil$ and $m_2 = \lfloor m/2 \rfloor$; this will allow us to express $\psi(m)$ as $\min\{4^n, \theta(\lceil m/2 \rceil, \lfloor m/2 \rfloor)\}$.

We consider first the case $\ell = 2$ and show that $\theta(m_1, m_2)$ is minimized when m_1 and m_2 are as close to each other as possible, formalized in the following “balancing lemma”.

Lemma 5.12 *If m_1 and m_2 are positive integers with $m_1 + m_2 = m$, then $\theta(m_1, m_2) \geq \theta(\lceil m/2 \rceil, \lfloor m/2 \rfloor)$.*

Proof. We consider even and odd values of m separately.

Suppose $m = 2a$ is even. Let $c \leq a - 1$ be a positive integer. We will show that $\theta(a + c, a - c)/\theta(a, a) \geq 1$. To this end, observe first that

$$(2^{a+c} + 2^{a-c} - 1)/(2^a + 2^a - 1) \geq 2^{c-1} .$$

Thus,

$$\frac{\theta(a + c, a - c)}{\theta(a, a)} \geq \frac{a!a!}{(a + c)!(a - c)!} 4^{c-1} =: \rho(a, c) .$$

Next, note that $\rho(a, c)$ grows with a for any fixed c; to see this, observe that the ratio $\rho(a, c)/\rho(a-1, c)$ equals $a^2/[(a+c)(a-c)] > 1$ for $0 < c < a$. Thus, for any fixed c it would suffice to show that $\rho(c+1, c) \geq 1$. What we, in fact, can do is to show that $\rho(c+1, c)$ grows with c and that $\rho(3, 2) > 1$, which leaves the case $c = 1$ open for a moment. Here, the former claim is proven by

$$\frac{\rho(c + 1, c)}{\rho(c, c - 1)} = \frac{4(c + 1)^2}{(2c + 1)2c} > \frac{4(c + 1)^2}{(2c + 2)^2} = 1 ,$$

and the latter claim by calculation: $\rho(3, 2) = 6/5 > 1$. Finally, the case of $c = 1$ is handled by

$$\begin{aligned} \frac{\theta(a + 1, a - 1)}{\theta(a, a)} &= \frac{a}{a + 1}\Big(\frac{5 \cdot 2^{a-1} - 1}{4 \cdot 2^{a-1} - 1}\Big)^2 \\ &> \frac{2}{2 + 1}\Big(\frac{5}{4}\Big)^2 \\ &= \frac{50}{48} > 1 . \end{aligned}$$

Suppose then that $m = 2a + 1$ is odd. Again, let $c \leq a - 1$ be a positive integer. To show that $\theta(a + 1 + c, a - c)/\theta(a + 1, a) \geq 1$ we will repeat the line of argumentation given above for even m. Observe

$$(2^{a+1+c} + 2^{a-c} - 1)/(2^{a+1} + 2^a - 1) \geq 2^{c+1}/3 .$$

Thus,

$$\begin{aligned} \frac{\theta(a + 1 + c, a - c)}{\theta(a + 1, a)} &\geq \frac{(a + 1)!a!}{(a + 1 + c)!(a - c)!} \frac{4^{c+1}}{9} \\ &=: \rho'(a, c) . \end{aligned}$$

Next, note that $\rho'(a,c)$ grows with a for any fixed c; to see this, observe that the ratio $\rho'(a,c)/\rho'(a-1,c)$ equals $(a+1)a/[(a+1+c)(a-c)] > 1$ for $0 < c < a$. Thus, for any fixed c it would suffice to show that $\rho'(c+1,c) \geq 1$. What we, in fact, can do is to show that $\rho'(c+1,c)$ grows with c and that $\rho'(3,2) > 1$, which leaves the case $c = 1$ open for a moment. Here, the former claim is proved by

$$\frac{\rho'(c+1,c)}{\rho'(c,c-1)} = \frac{4(c+2)(c+1)}{(2c+2)(2c+1)} > \frac{4(c+2)(c+1)}{(2c+4)(2c+2)} = 1 ,$$

and the latter by calculation: $\rho'(3,2) = 64/45 > 1$. Finally, the case of $c = 1$ is handled by

$$\begin{aligned}\frac{\theta(a+2,a-1)}{\theta(a+1,a)} &= \frac{a}{a+2}\left(\frac{9\cdot 2^{a-1}-1}{6\cdot 2^{a-1}-1}\right)^2 \\ &> \frac{2}{2+2}\left(\frac{3}{2}\right)^2 \\ &= \frac{18}{16} > 1 .\end{aligned}$$

□

At first glance, one might think that the uniform distribution should minimize $\theta(m_1, m_2, \ldots, m_\ell)$ also for $\ell > 2$. However, this is in fact not the case. A counter-example is $\theta(3,3,3) \approx 4.536 > 4.421 \approx \theta(4,4,1)$. Therefore, the proof technique we used for Lemma 5.12 or other convexity arguments seems not applicable. Instead, we are able to prove the following "shortening lemma", which states that for any bucket order of length $\ell+1 \geq 3$ there is another bucket order of length $\ell \geq 2$ that yields a smaller time–space product.

Lemma 5.13 *Let $\ell \geq 2$ and let $m_1 \geq m_2 \geq \cdots \geq m_{\ell+1} \geq 0$ and $c_1 \geq c_2 \geq \cdots \geq c_\ell \geq 0$ be integers such that $m_{\ell+1} = c_1 + c_2 + \cdots + c_\ell$ and $c_1 - c_\ell \leq 1$. Then $\theta(m_1, m_2, \ldots, m_{\ell+1}) > \theta(m_1 + c_1, m_2 + c_2, \ldots, m_\ell + c_\ell)$.*

Proof. Put $a_i := m_i + c_i$ for $i = 1, 2, \ldots, \ell$. Also, denote $b := m_{\ell+1}$ for brevity. We will show that $\theta(a_1, a_2, \ldots, a_\ell)/\theta(m_1, m_2, \ldots, m_\ell, b) < 1$.

We begin with the case $b = 1$. Then it suffices to show that

$$\frac{\theta(m_1+1, m_2, \ldots, m_\ell)}{\theta(m_1, m_2, \ldots, m_\ell, 1)} = \frac{1}{m_1+1}\left(\frac{2^{m_1+1} + 2^{m_2} + \cdots + 2^{m_\ell} + 1 - \ell}{2^{m_1} + 2^{m_2} + \cdots + 2^{m_\ell} + 2 - \ell}\right)^2 < 1 .$$

To see that this holds, we consider a few cases to show that the squared term is always less than $m_1 + 1$. Because the squared term is always less than 4, we are done for $m_1 \geq 3$. Now, if $m_1 = 2$, then the squared term is at most $[(2^3 + 2^1 - 1)/(2^2 + 2^1 + 2^1 - 2)]^2 = (9/6)^2 = 9/4 < 3$. Finally, if $m_1 = 1$, then the squared term is at most $[(2^2+2^1-1)/(2^1+2^1+2^1-2)]^2 = (5/4)^2 = 25/16 < 2$.

Then, for any $b \geq 1$, we notice the bound

$$\frac{m_1!m_2!\cdots m_\ell!b!}{a_1!a_2!\cdots a_\ell!} \leq \frac{b!}{(b+1)^b} \tag{5.2}$$

that holds because the denominator contains the factorials in the numerator, except for $b!$, plus b other terms all greater or equal to $b+1$.

Next suppose $2 \leq b \leq \ell$. Under this assumption $a_i = m_i + 1$ for $i = 1, 2, \ldots, b$ and $a_i = m_i$ for $i = b+1, b+2, \ldots, \ell$. So we find that

$$2^{a_1} + 2^{a_2} + \cdots + 2^{a_\ell} + 1 - \ell \leq 2\big(2^{m_1} + 2^{m_2} + \cdots + 2^{m_\ell} + 2^b - \ell\big) \; . \tag{5.3}$$

To see this, subtract the terms on the left from the ones on the right to get

$$2^{m_{b+1}} + 2^{m_{b+2}} + \cdots + 2^{m_\ell} + 2^{b+1} - \ell - 1 \geq (\ell - b + 2)2^b - \ell - 1 \geq 0 \; .$$

Here the last inequality follows because $(\ell - b + 2)2^b$ clearly grows with b for $2 \leq b \leq \ell$, and at $b = 2$ we have $\ell 2^2 \geq \ell + 1$, which holds for all $\ell > 1$. Combining the bounds (5.2) and (5.3) yields

$$\frac{\theta(a_1, a_2, \ldots, a_\ell)}{\theta(m_1, m_2, \ldots, m_\ell, b)} \leq \frac{2^2 b!}{(b+1)^b} \leq \frac{4b!}{(b+1)^b} < 1$$

for $b \geq 2$, since $(4\cdot 2!)/(2+1)^2 = 8/9$ and it is easy to verify that $4b!/(b+1)^b$ decreases when b grows.

It remains to consider the case $b > \ell$. We will first examine the cases $b = 3$ and $b = 4$, and then the remaining case $b \geq 5$.

Suppose $b = 3$; hence, $\ell = 2$. Thus $a_1 = m_1 + 2$ and $a_2 = m_2 + 1$, and so

$$\frac{\theta(a_1, a_2)}{\theta(m_1, m_2, b)} = \frac{3!}{(m_1+2)(m_1+1)(m_2+1)} \left(\frac{2^{m_1+2} + 2^{m_2+1} - 1}{2^{m_1} + 2^{m_2} + 2^3 - 2}\right)^2 .$$

Now, if $m_1 = 3$, then $m_2 = 3$, and the above ratio evaluates to $6/80(47/22)^2 < 1$. Otherwise $m_1 \geq 4$, and the ratio can be bounded from above by $6/(6 \cdot 5 \cdot 4)4^2 = 4/5 < 1$.

Next suppose $b = 4$. This means $a_i \leq m_i + 2$ for all $i = 1, 2, \ldots, \ell$. Using the bound (5.2) yields

$$\frac{\theta(a_1, a_2, \ldots, a_\ell)}{\theta(m_1, m_2, \ldots, m_\ell, b)} \leq \frac{4^2 \cdot 4!}{(4+1)^4} = \frac{384}{625} < 1 \; .$$

Finally, suppose $b \geq 5$. Observe $a_i \leq m_i + \lceil b/\ell \rceil \leq m_i + (b+\ell-1)/\ell \leq m_i + (b+1)/2$. Thus,

$$\begin{aligned} & 2^{a_1} + 2^{a_2} + \cdots + 2^{a_\ell} + 1 - \ell \\ & \leq 2^{(b+1)/2}\bigl(2^{m_1} + 2^{m_2} + \cdots + 2^{m_\ell} + 2^b - \ell\bigr) \ ; \end{aligned} \tag{5.4}$$

note that $2^b - \ell$ is positive, since $b > \ell$. Combining the bounds (5.2) and (5.4) yields

$$\frac{\theta(a_1, a_2, \ldots, a_\ell)}{\theta(m_1, m_2, \ldots, m_\ell, b)} \leq \frac{2^{b+1} b!}{(b+1)^b} < 1 \ .$$

Here the last inequality follows because at $b = 5$ we have $2^{5+1}5!/(5+1)^5 = 80/81 < 1$ and because this bound decreases when b grows. To verify the latter claim, observe that the bound at b divided by the bound at $b-1$ equals $2[b/(b+1)]^b \leq 2[(b+1)/b]\mathrm{e}^{-1} \leq 8/(3\mathrm{e}) < 1$ for any $b \geq 3$. □

Combining the shortening lemma (Lemma 5.13) with the balancing lemma (Lemma 5.12) immediately yields the following summary.

Lemma 5.14 *Let $m \geq 1$ be an integer. Then $\psi(m)$ equals the smaller of $\theta(m) = 4^m$ and $\theta(\lceil m/2 \rceil, \lfloor m/2 \rfloor)$.*

We are now ready to show that the time–space product of the $13 * 13$ scheme is the smallest one can achieve with parallel compositions of bucket orders.

Theorem 5.15 (Lower bound) *Let P be a parallel composition of bucket orders on $\{1, 2, \ldots, n\}$. Then the time–space product $\theta(\mathcal{P}(P))^{1/n}$ is at least $4(\kappa\lambda)^{1/26} \geq 3.9271$, where κ and λ are as defined in Theorem 5.9.*

Proof. By the bound (5.1), it suffices to show that $\psi(m)^{1/m}$ is minimized at $m = 26$. By calculation with a computer, using Lemma 5.14, we find that this is indeed the case when $1 \leq m \leq 149$ (results not shown).

It remains to show that $\psi(m)^{1/m} \geq \psi(26)^{1/26} = 3.9271\ldots$ for all $m \geq 150$. Because this clearly holds if $\psi(m) = \theta(m)$, we may, by Lemma 5.14, without any loss in generality assume $\psi(m) = \theta(\lceil m/2 \rceil, \lfloor m/2 \rfloor)$. To this end, define $v(m) := \binom{m}{\lfloor m/2 \rfloor}^{1/m}$ and observe

$$\begin{aligned} \psi(m)^{1/m} &= v(m)\bigl(2^{\lceil m/2 \rceil} + 2^{\lfloor m/2 \rfloor} - 1\bigr)^{2/m} \\ &\geq v(m)\bigl(2^{m/2+1} - 1\bigr)^{2/m} \\ &\geq 2v(m) \ . \end{aligned}$$

Next we show that $\upsilon(m)$ grows with m, by proving $\upsilon(2a-1)/\upsilon(2a) \leq 1$ and $\upsilon(2a)/\upsilon(2a+1) \leq 1$ for any $a = 1, 2, \ldots$ (actually, for any $m \geq 150$ would do). The former is shown by

$$\left(\frac{\upsilon(2a-1)}{\upsilon(2a)}\right)^{2a-1} = \binom{2a}{a}^{\frac{1}{2a}} \frac{a}{2a} \leq \left(2^{2a}\right)^{\frac{1}{2a}} \frac{1}{2} = 1 ;$$

the latter is shown by

$$\begin{aligned}\left(\frac{\upsilon(2a)}{\upsilon(2a+1)}\right)^{2a+1} &= \binom{2a}{a}^{\frac{1}{2a}} \frac{a+1}{2a+1} \\ &\leq \left(\frac{2^{2a}}{\mathrm{e}}\right)^{\frac{1}{2a}} \frac{1}{2}\left(1 + \frac{1}{2a+1}\right) \\ &\leq \mathrm{e}^{-\frac{1}{2a}} \mathrm{e}^{\frac{1}{2a+1}} \\ &< \mathrm{e}^0 = 1 ,\end{aligned}$$

where the first inequality holds for $a \geq 2$, whereas in the case $a = 1$ we replace e by 2 and obtain the bound $2\sqrt{2}/3 < 1$.

Now it suffices to verify that $2\upsilon(150) = 3.92778\ldots > 3.9271$. □

The previous results can be adapted to solve the structure discovery problems in Bayesian networks. Combining Theorems 4.14, 4.16 and 5.9 yields the following theorem.

Theorem 5.16 *Given explicit input, the* Osd *problem and the* Fp *problem can be solved in $O(2.71^n n^2 \tau(n))$ time and $O(1.46^n n)$ space provided that each node has at most $O(1.45^n)$ possible parent sets, which form a downward-closed set family, and the membership in the possible parent sets can be evaluated in $\tau(n)$ time.*

Finding an optimal partial order scheme in general is an interesting question. While both the number of ideals and the number of the reorderings of a parallel bucket order admit closed-form formulas, this is not the case in general. Both computing the number of ideals [26] and the number of linear extensions [12][2] are #P-complete problems. So far it remains an open problem whether the $13 * 13$ scheme is the optimal partial order scheme. We have implemented a script and computed the number of ideals and the number of linear extensions for all possible partial orders up to 8 elements. For all numbers of elements n from 2 to 8, the balanced scheme

[2]The $n!$ divided by the number of linear extensions sets a lower bound for the size of the cover.

described in Lemma 5.12 is optimal. Therefore, we present the following conjecture that states that the $13 * 13$ scheme is actually an optimal partial order scheme.

Conjecture 5.17 *The time–space product of a POF $\mathcal{P}$ on N is at least $4(\kappa\lambda)^{1/26}$.*

5.4 Parallel Bucket Orders in Practice

In the previous section we showed that out of all possible parallel bucket order schemes, the $(13 * 13)^{\lfloor n/26 \rfloor}$ scheme minimizes the time–space product. Unfortunately, the practical value of this result is rather limited. To see this, we recall Theorem 5.9 which shows that permutation problems can be solved using the $(13 * 13)^{\lfloor n/26 \rfloor}$ scheme in $O^*((2\kappa^{1/26})^n)$ time and $O^*((2\lambda^{1/26})^n)$ space, where $\kappa > 2.539054 \times 10^3$ and $\lambda < 2.441258 \times 10^{-4}$. In words, the $(13 * 13)^{\lfloor n/26 \rfloor}$ scheme reduces the space requirement at least by factor 4000 but the time requirement increases at least by factor 2500, which renders the time requirement the bottleneck in practice.

In practice, we usually have a fixed amount of memory available, which sets a strict limit to the space usage. Therefore, we want to solve the problem in a given space as fast as possible. Lemma 5.12 shows that for bucket orders on a base-set of size m, a balanced bucket order (buckets are about the same size) with two buckets gives the best tradeoff. Table 5.1 shows the time and space requirements of the POFs $(\lfloor m/2 \rfloor, \lceil m/2 \rceil)^{\lfloor n/m \rfloor}$ in the range $1 \leq m \leq 40$. Notice that when $m < 13$, both the space requirement and the time–space product decrease whenever m increases. Beyond that point both the space requirement and the time–space product decrease for even m. The results suggest that in practice one should choose the smallest m that gives a small enough space requirement. We also notice that the time–space product for $m \geq 10$ is always less than 4. This suggests that even these "suboptimal" schemes can yield good tradeoffs.

In what follows, we present time and space bounds for solving the Osd and Fp problems. Naturally, the same upper bounds hold also for all the polynomial local time permutation problems of degree 1. However, the structure discovery problems are more challenging as one needs to handle the possibly exponential-sized input. Next, we consider a scenario in which the number of parallel bucket orders, p, is between 0 and $\lfloor n/m \rfloor$. A POF $(\lfloor m/2 \rfloor, \lceil m/2 \rceil)^p$ consists of $\binom{m}{\lfloor m/2 \rfloor}^p$ partial orders, each of which has $2^{n-mp}(2^{\lceil m/2 \rceil} + 2^{\lfloor m/2 \rfloor} - 1)^p$ ideals. Now plugging these numbers into Theorems 4.14, 4.15 and 4.16, we get the following time and space bounds.

Table 5.1: Time and space requirement and time–space product of $(\lfloor m/2 \rfloor * \lceil m/2 \rceil)^{\lfloor n/m \rfloor}$ schemes for $1 \le m \le 40$. The optimal tradeoff is shown in bold font.

m	T	S	TS	m	T	S	TS
1	2.0000	2.0000	4.0000	21	2.6930	1.4658	3.9472
2	2.4495	1.7321	4.2427	22	2.6918	1.4595	3.9285
3	2.4663	1.7100	4.2172	23	2.6995	1.4613	3.9445
4	2.5458	1.6266	4.1409	24	2.6981	1.4557	3.9275
5	2.5603	1.6154	4.1358	25	2.7054	1.4574	3.9428
6	2.5874	1.5705	4.0633	**26**	2.7039	1.4525	**3.9271**
7	2.6009	1.5651	4.0705	27	2.7107	1.4542	3.9417
8	2.6126	1.5362	4.0131	28	2.7091	1.4497	3.9272
9	2.6253	1.5339	4.0268	29	2.7154	1.4514	3.9411
10	2.6308	1.5134	3.9812	30	2.7138	1.4473	3.9276
11	2.6427	1.5129	3.9979	31	2.7198	1.4490	3.9408
12	2.6452	1.4974	3.9608	32	2.7181	1.4452	3.9282
13	2.6563	1.4979	3.9787	33	2.7238	1.4469	3.9408
14	2.6573	1.4856	3.9476	34	2.7221	1.4434	3.9289
15	2.6676	1.4867	3.9658	35	2.7275	1.4450	3.9410
16	2.6677	1.4767	3.9392	36	2.7258	1.4418	3.9298
17	2.6773	1.4781	3.9571	37	2.7309	1.4433	3.9413
18	2.6767	1.4697	3.9338	38	2.7292	1.4403	3.9307
19	2.6856	1.4713	3.9512	39	2.7340	1.4418	3.9418
20	2.6847	1.4641	3.9305	40	2.7323	1.4390	3.9316

Theorem 5.18 *Given explicit input, the* Osd *and the* Fp *problem can be solved in* $O\Big(\big[\binom{m}{\lfloor m/2 \rfloor}(I + F\tau(n))\big]n^2\Big)$ *time and* $O(\big[I+F\big]n)$ *space, where* $I = 2^{n-mp}(2^{\lceil m/2 \rceil} + 2^{\lfloor m/2 \rfloor} - 1)^p$, *for any* $m = 2, \ldots, n$ *and* $p = 0, \ldots, \lfloor n/m \rfloor$, *provided that each node has at most* F *possible parent sets, which form a downward-closed family, and the membership in the possible parent sets can be evaluated in* $\tau(n)$ *time.*

Theorem 5.19 *Given implicit input, the* Osd *and the* Fp *problem can be solved in* $O\Big(\big[\binom{m}{\lfloor m/2 \rfloor}(I + F\tau(n)\delta(n))\big]n^2\Big)$ *time and* $O(In)$ *space, where* $I = 2^{n-mp}(2^{\lceil m/2 \rceil} + 2^{\lfloor m/2 \rfloor} - 1)^p$, *for any* $m = 2, \ldots, n$ *and* $p = 0, \ldots, \lfloor n/m \rfloor$, *provided that each node has at most* F *possible parent sets, which form a downward-closed family, the membership in the possible parent sets can be* *evaluated in $\tau(n)$ time and a local score can be computed from data in $\delta(n)$ time.*

We observe that the number of ideals, I, dominates both the time and the space requirements, as long as the size of input, F is not too large. Generally, the number of ideals grows exponentially with n and thus we can in practice handle an exponential number of possible parent sets with essentially no extra cost. Next, we take a closer look at this issue and allow the number of possible parents to grow linearly with n. Formally, let the maximum indegree be $k = \mu n$, with some *slope* $\mu \leq 1/2$. The interesting question is how large μ can be be still guaranteeing that the size of the input does not dominate the time requirement.

Next, we present a bound for μ. The argumentation follows the same pattern as in Section 5.1. We notice that the slope depends on the scheme that is used. For a moment, let us focus on the $(\lfloor m/2 \rfloor * \lceil m/2 \rceil)^{\lfloor n/m \rfloor}$ scheme, and for simplicity assume that m is even and n is divisible by m. For any fixed m, we bound the largest slope by μ_m in similar fashion as in Section 5.1 for the pairwise scheme. We observe that, by the inequality (A.3) in Appendix A, the number of possible parent sets $\sum_{i=0}^{\mu_m n} \binom{n-1}{i}$ is bounded from above by $2^{H(\mu_m)n}$, where H is the binary entropy function. On the other hand, every member of a POF $(m/2 * m/2)^{n/m}$ has $((2^{m/2+1} - 1)^{1/m})^n$ ideals. Thus, the number of ideals dominates the space and time requirements if $2^{H(\mu_m)n} \leq ((2^{m/2+1} - 1)^{1/m})^n$; or equivalently, $H(\mu_m) \leq (1/m)\log_2(2^{m/2+1} - 1)$. By solving this inequality numerically, we get a bound for μ_m. Table 5.2 shows the μ_m for each even $m \leq 26$.

Table 5.2: Bounds on the maximum indegree slopes for the $(m/2*m/2)^{n/m}$ schemes.

m	μ_m	m	μ_m	m	μ_m
2	0.238	12	0.139	20	0.127
4	0.190	14	0.135	22	0.125
6	0.167	16	0.131	24	0.124
8	0.153	18	0.129	26	0.123
10	0.145				

Since each member of the $(m/2*m/2)^{n/m}$ scheme has as many or fewer ideals than each member of the $(m/2 * m/2)^p$ scheme, for $p \leq n/m$, we have the following characterization.

Corollary 5.20 *Given explicit input, the* OSD *and the* FP *problem can be solved in $O(\binom{m}{m/2} In^2)$ time and $O(In)$ space, where $I = 2^{n-mp}(2^{m/2+1} - 1)^p$, for any $p = 0, 1, 2, \ldots, \lfloor n/m \rfloor$ and $m = 2, 4, 6, \ldots, 26$, provided that each node has at most $\mu_m n$ parents, with μ_m as given in Table 5.2.*

We note that given that each node has at most $\mu_m n$ parents, the number of ideals dominates the input size and both the OSD and the FP problem with both explicit and implicit input can be solved in the same space (within a constant factor). We further note that in this case explicit input seems superior as the time usage is, in practice, considerably lower as there is no need to recompute local scores.

To get a grasp of the bounds for the slope, consider, for example, a 30-node DAG. Now, the maximum indegree can be set to $\lfloor 0.238 \times 30 \rfloor = 7$ for the $(1 * 1)^p$ scheme (the pairwise scheme) and to $\lfloor 0.139 \times 30 \rfloor = 4$ for the $(6 * 6)^p$ scheme. A larger maximum indegree may render the size of the input to dominate the running time. In Section 5.4.2 we will show that these bounds are only slightly conservative.

5.4.1 Parallelization

All the schemes presented so far are easily parallelized onto several processors each with its own memory. This is due to the fact that computations for every $P \in \mathcal{P}$ are independent of each other and hence each partial order can be considered separately. For example, the two-bucket scheme (Section 3.1.1) can be parallelized onto $\binom{n}{s}$ processors. The divide-and-conquer scheme (Section 3.1.2) can be parallelized onto $\binom{n}{n/2}$ processors. If the recursion is applied several times the execution at each level can be further parallelized.

The $(\lfloor m/2 \rfloor, \lceil m/2 \rceil)^p$ schemes can obviously be parallelized onto $\binom{m}{\lfloor m/2 \rfloor}^p$ processors. Furthermore, considering the OSD problem, the optimal local scores in lines 7–9 in Algorithm 5 can be precomputed, that is, not merging with the computation of the score of an optimal network in line 8– in parallel for each n nodes as in the Silander–Myllymäki implementation [96]. Thus, in total this amounts to parallelization onto $\binom{m}{\lfloor m/2 \rfloor}^p n$ processors. This parallelization is efficient in the sense that the running time decreases by the same factor that the number of parallel processors increases. By Corollary 5.20, ignoring factors polynomial in n, the running time per processors becomes $O((2^{m/2+1} - 1)^{p/m} 2^{n-mp})$ (under the conditions of Corollary 5.20) which is exponentially less than 2^n when p grows. This can be verified by noticing that $(2^{m/2+1} - 1)^{p/m} 2^{n-mp} \leq (2^{m/2+1})^{p/m} 2^{n-mp} = 2^{p/2+p/m} 2^{n-mp} = 2^{n-(m-(m+2)/(2m))p}$. When $m \geq 2$, the expression decreases whenever p grows.

5.4.2 Empirical Results

So far we have analyzed the asymptotic time and space requirements of parallel bucket orders. Although the asymptotic bounds are interesting, they do not necessarily match the bounds in practice with small input. Therefore, an empirical study on the running times and space usage of parallel bucket orders was conducted.

The dynamic programming algorithm over parallel bucket orders for the OSD problem was implemented in the C++ language. The OSD problem was chosen as it allows the investigations related to the number of possible parent sets. The FP problem was not considered as its behavior is similar to the OSD problem, and thus the second algorithm would not have added value to the current investigation. The implementation is not optimized, and so the empirical results should be viewed as a mere proof of concept[3]. The implementation was tested using the $(\lfloor m/2 \rfloor, \lceil m/2 \rceil)^p$ scheme varying the number of nodes n, bucket order sizes m and the number of parallel bucket orders p. The experiments were run on Intel Xeon R5540 processors, each with 32 GB of RAM. In all experiments explicit input was used, that is, the local scores were taken as given, so computing them is not included in the running time estimates.

First, the running time for the limit of 16 GB of memory was examined, letting the number of nodes n vary from 25 to 34, with maximum indegree set to 3. First, the smallest bucket order size m that yields a memory requirement of 16 GB or less was estimated. Then Algorithm 5 was run for ten partial orders in a POF $(\lceil m/2 \rceil, \lfloor m/2 \rfloor)^1$ and the average of the running times was computed; finally, the average was multiplied by the size of the POF to get an estimate of the total running time; see Table 5.3. We observe that, as expected, the time requirement grows rapidly with n: An optimal 25-node DAG can be found in about 25 minutes. Already 30-node DAGs require over 16 days of CPU time. And 34-node DAGs become feasible only with large-scale parallelization: with 1000 processors an optimal network can be found in about 6 days.

Next, consider how different schemes affect the running times and the space usage in practice. Two specializations of the generalized bucket order scheme $(\lceil m/2 \rceil * \lfloor m/2 \rfloor)^p$ were chosen for the analysis: the *practical scheme*, where $p = 1$, and the pairwise scheme, where $m = 2$. The re-

[3]The Silander–Myllymäki implementation [96] is about five times faster when the algorithms were run on the same setting, that is, the algorithm for dynamic programming over partial orders is given just the trivial partial order on the nodes.

Table 5.3: Running times for a varying number of nodes when space is limited to 16 GB. Columns: n = the number of nodes, p = the number of parallel bucket orders, m = the size of the balanced bucket order, Time = running time (in CPU hours) per one partial order, PO = the number of partial orders to cover all linear orders, TT = total running time (in CPU hours).

n	p	m	Time	PO	TT
25	0	0	0.42	1	0.42
26	1	3	0.44	3	1.34
27	1	5	0.49	10	4.9
28	1	8	0.35	70	24.8
29	1	10	0.39	252	97.7
30	1	12	0.43	924	394
31	1	14	0.49	3432	1671
32	1	16	0.53	12 870	6784
33	1	18	0.83	48 620	40 332
34	1	20	0.78	184 756	144 930

sults are shown in Tables 5.4 and 5.5. As expected, the practical scheme yields a clearly better space–time tradeoff than the pairwise scheme. This is perhaps even more clearly pronounced in Figure 5.4, which shows the empirical tradeoffs in the space–time plane along with the analytical bounds (Theorem 5.9, Proposition 5.3); here the empirical results were normalized as follows. Each empirical time and space requirement was divided, respectively, by the empirical time and space requirement of the basic dynamic programming algorithm (the case of the trivial order) and the 26th root of the ratio, multiplied by 2, was taken as the normalized value. The analytical bounds were normalized analogously. We see that, in general, the empirical and analytical bounds seem to be in a good agreement, the analytical bounds being slightly conservative for medium m (the practical scheme) and medium p (the pairwise scheme).

Also the influence of the maximum indegree k on various characteristics of the implementation was investigated. To this end, the (degenerate) scheme $(10 * 10)^1$ with $n = 20$ nodes was analyzed, varying k from 1 to 8. For interpretation of the results, shown in Table 5.6, it is useful to note that the partial orders in question have 2047 ideals. For comparison, the (worst-case) input size is 1160 for $k = 3$ and 5036 for $k = 4$. So we conclude that the number of ideals dominates the input size precisely when

Table 5.4: Running times (in CPU hours) and memory usage (in MB) of the practical scheme $(\lceil m/2 \rceil * \lfloor m/2 \rfloor)^1$ with $2 \leq m \leq 26$ and $n = 26$.

m	p	Space	Time	m	p	Space	Time
0	0	21248	1.01	14	1	331	32.51
2	1	15936	1.13	15	1	248	43.97
3	1	13280	1.41	16	1	166	64.35
4	1	9296	1.75	17	1	124	87.79
5	1	7304	2.13	18	1	83	129.65
6	1	4980	2.80	19	1	62	171.93
7	1	3818	3.57	20	1	41	256.61
8	1	2573	4.96	21	1	31	342.92
9	1	1950	6.88	22	1	21	489.88
10	1	1307	9.06	23	1	16	638.48
11	1	986	12.15	24	1	10	976.50
12	1	659	17.43	25	1	8	1444.53
13	1	495	24.31	26	1	5	2600.15

Table 5.5: Running times (in CPU hours) and memory usage (in MB) of the pairwise scheme $(1 * 1)^p$ with $1 \leq p \leq 13$ and $n = 26$.

m	p	Space	Time	m	p	Space	Time
0	0	21248	1.01	2	7	2836	12.79
2	1	15936	1.13	2	8	2127	20.25
2	2	11952	1.48	2	9	1595	32.36
2	3	8964	2.09	2	10	1197	51.66
2	4	6723	3.19	2	11	897	78.96
2	5	5042	4.97	2	12	673	126.52
2	6	3782	7.90	2	13	505	198.88

$k \leq 3$. Now, recall that the bounds in Corollary 5.20 guarantee this only for $k \leq 2$, indicating that the analytical bounds are not tight but slightly too pessimistic. Table 5.6 also shows, perhaps somewhat surprisingly, that even if the input size for $k = 5$ is more than 3 times the input size for $k = 4$, the total running time less than doubles. This can be explained by the fact that the respective increase in the tail accesses, from 46,520 to 160,260, does not yet pay off, since the number of computation steps that

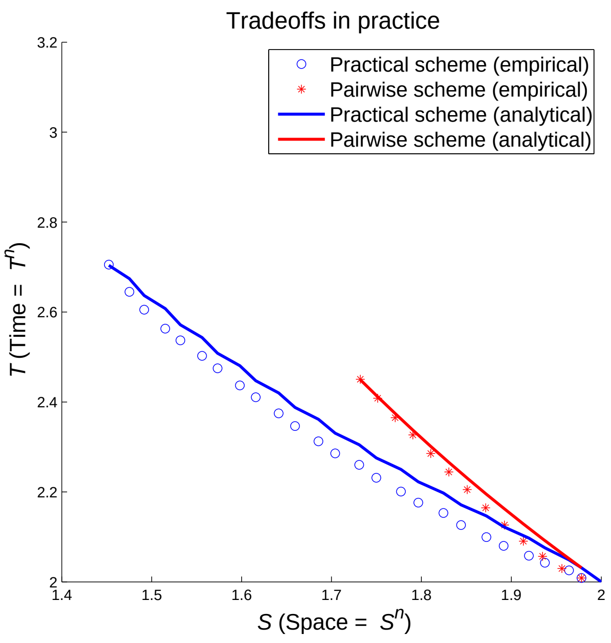


Figure 5.4: Empirical and analytical time and space requirements of the pairwise scheme and the practical scheme, for $n = 26$ nodes; see text for a more detailed description.

are not related to the input (nor the maximum indegree) appears to be as large as $358,300$. For larger k, the running time will grow about linearly with the number of tail accesses.

5.4.3 Scalability beyond Partial Orders

The results presented in the previous section show that the partial order approach allows one the learn networks of little more than 30 nodes. Although this is an improvement compared to the standard dynamic programming, the scalability of the partial order approach is rather limited. Next, we discuss the ways to solve the structure discovery problems when the number of nodes is large.

The weakness of the presented dynamic programming algorithms is that the time requirement in the best-case equals to the time requirement in the worst-case. This is due to the fact that they do not take advantage of the structure of the data. However, it is often unnecessary to go explicitly

Table 5.6: Characteristics of the practical scheme $(10 * 10)^1$ with $n = 20$ nodes, for varying maximum indegree k. Columns: Input size = the number of parent sets per node, Regular accesses = the number of accesses to input when the parent set is an ideal, Tail accesses = the number of accesses to input when the parent set is in the tail of an ideal, Total time = the running time in hours. Regular and tail accesses are presented per partial order.

k	Input size	Regular accesses	Tail accesses	Total time
1	20	210	90	2.4
2	191	1020	1350	2.5
3	1160	3060	9840	2.8
4	5036	6420	46500	4.3
5	16664	10200	160260	8.5
6	43796	13140	429480	18.3
7	94184	14700	932160	35.5
8	169766	15240	1687530	60.4

through the whole search space. Therefore, one way to improve the scalability of the structure discovery is to make use of the inner structure of the data whenever possible.

Recently, there have been attempts to improve the scalability of the dynamic programming approach using the A* search in the subset lattice [73, 112]. Although the A* search procedure often speeds up the computation, this approach does not seem to yield substantial improvements in scalability. Other approaches like linear programming [21] and branch-and-bound [24] have been used to find optimal networks on about 60 nodes; thorough evaluations on the capabilities of these approaches are so far missing. Here the running time can be greatly influenced by the data. For example, the running time of the branch-and-bound method skyrockets if one uses BDeu score with a large equivalent sample size.

In this thesis we have considered exact algorithms only. Nevertheless, one can improve scalability of the structure discovery in Bayesian networks by employing approximation algorithms and heuristics. Here, exact algorithms can be used as components of approximation algorithms and heuristics. For example, one could apply the partial order approach to develop more efficient MCMC for approximating the posterior probabilities of structural features. Many modern MCMC algorithms [32, 39] sample linear orders and compute the posterior probability conditional to the linear order. If one samples partial orders instead of linear orders, the size of the sample space can be decreased, since we need to consider only the partial orders in an exact cover. Simultaneously, one could compute the posterior conditional to a partial order in essentially the same time as for a linear order given that the partial order in question has a sufficiently small number of ideals. This is due to the fact that the computation of the posterior conditional to the linear order still requires evaluating the zeta transform at some points. It is worth pointing out that, in fact, this kind of heuristic algorithm has been implemented in a recent paper; see Niinimäki et al. [77].

One factor that limits scalability of all score-based algorithms is the need to compute (or store) local scores. If one has n nodes and a node can have at most k parents, the total number of local scores is $n\sum_{i=0}^{k}\binom{n-1}{i}$. When n grows, this number becomes a limiting factor. For example, setting $n = 200$ and $k = 5$ yields almost 500 billion local scores. Thus, for data sets consisting of hundreds of nodes, one has to resort to local search methods.

Chapter 6

Learning Ancestor Relations

So far we have implicitly assumed complete data, or in other words, that all relevant nodes are observed. In practice, however, this is often not the case. Thus, we investigate how the unobserved nodes affect structure discovery.

As mentioned earlier, there are two approaches to structure discovery in Bayesian networks: constraint-based and score-based. While the constraint-based methods are not particularly suitable for importing prior knowledge, using the data efficiently or handling nonidentifiability, they have, however, given rise to a profound theory for dealing with unobserved variables. The best known algorithms that identify unobserved nodes are IC* [84, 85] and FCI [100, 101]. The score-based methods [19, 47], particularly Bayesian ones [39, 65], on the other hand, excel in flexibility and efficiency. However, a principled treatment of unobserved nodes is computationally infeasible and the handling of unobserved nodes in practice is limited to some score-based heuristics for finding unobserved nodes [28, 29, 31, 38]. Therefore, the unobserved nodes are often ignored altogether: one either refuses to make any conclusions, especially causal ones, about the DAG, or one makes conclusions with unquantified risk of erroneous claims.

Motivated by these issues, we study in this chapter the potential of Bayesian averaging in learning of structural features on *observed nodes only.* As the score-based methods are supposed to model the data without unobserved variables, it is crucial to know what kind of features can be reliably learned from observational data, even when there are unobserved nodes at work. To this end, we investigate ancestor relations, that is, the existence of a directed path between two nodes (see Definition 2.5).

The idea of studying ancestor relations is not new. Spirtes et al. [101] investigated the prospects of learning ancestor relations using FCI algorithm in a small case study. Their results suggest that reliable learning of ancestor relations is possible in the presence of unobserved nodes; however, direct comparison to Bayesian averaging is not reasonable, as the predictions by FCI are unquantified and predictions are not necessarily made for all pairs of nodes. Later, Friedman and Koller [39] studied ancestor relations using the Bayesian approach under the assumption of no unobserved variables. They sampled DAGs (via node orderings) from their posterior distribution using a well-known Markov chain Monte Carlo simulation[1] and the posterior probabilities of ancestor relations, also called path features, are estimated based on the sampled DAGs; based on the posterior probabilities, the ancestor relation is either claimed to hold or not to hold, potentially depending on the relative costs of making incorrect positive or negative claims.

Next, in Section 6.1 we continue the development of Bayesian algorithms and present an exact Bayesian averaging algorithm for computing posterior probabilities of ancestor relations. Then in Section 6.2 we report results from empirical tests.

6.1 Computation

Recall the ancestor relation problem (Definition 2.5). To present the ancestor relation problem formally, let $f(A)$ be an indicator for the ancestor relation that returns 1 if there is a directed path from the source node s to the target node t in A and 0 otherwise. We use the shorthand notation $s \rightsquigarrow t$ to denote the event $f(A) = 1$. The goal is to compute the posterior probability of the ancestor relation, that is, $Pr(s \rightsquigarrow t|D)$. Now assuming an order-modular prior on A and a decomposable likelihood score, we have

$$\begin{aligned}
Pr(s \rightsquigarrow t, D) &= \sum_{A} f(A) Pr(D|A) Pr(A) \\
&= \sum_{A} f(A) \prod_{v \in N} Pr(D_v|D_{A_v}, A_v) \sum_{L \supseteq A} Pr(A, L) \\
&= \sum_{A} f(A) \prod_{v \in N} Pr(D_v|D_{A_v}, A_v) \sum_{L \supseteq A} \prod_{v \in N} \rho_v(L_v) q_v(A_v) \\
&= \sum_{A} \sum_{L \supseteq A} f(A) \prod_{v \in N} \big[Pr(D_v|D_{A_v}, A_v) q_v(A_v) \rho_v(L_v)\big] \\
&= \sum_{L} \sum_{A \subseteq L} f(A) \prod_{v \in N} \big[Pr(D_v|D_{A_v}, A_v) q_v(A_v) \rho_v(L_v)\big].
\end{aligned}$$

[1]For more about Markov chain Monte Carlo (MCMC) methods, see, for example, Gilks et al. [44].

The difference compared to the equation (2.3) is that the indicator $f(A)$ does not factorize and therefore the straightforward dynamic programming algorithm from Section 2.3.2 does not apply here. Next, we modify the dynamic programming algorithm to solve the ancestor relations problem.

The idea of the algorithm is to compute for every node subset S its contribution to the target probability, $Pr(s \rightsquigarrow t, D)$, assuming the nodes in S are the first $|S|$ nodes in the linear order L; the contribution is over all DAGs on the node set S. The key difference to the dynamic programming algorithms in Sections 2.3.1 and 2.3.2 is that, aside from the node set S, we need to keep a handle on the nodes in S that are descendants of the source node s. To this end, define a set $T \subseteq S$ such that $t \in T$ if and only if s is an ancestor of t or $t = s$. Thus, every DAG on S determines exactly one such set $T \subseteq S$.

Furthermore, for sets $S \subseteq N$ and $T \subseteq S$ and a linear order $L_S \subseteq S \times S$ on S, we use the shorthand

$$\mathcal{A}(L_S, S, T) = \{A_S \subseteq L_S : \forall v \in S \ (s \rightsquigarrow v \text{ in } A_S \text{ iff } v \in T)\ \} .$$

In words, $\mathcal{A}(L_S, S, T)$ contains a particular DAG A_S on S if and only if A_S is compatible with L_S and A_S contains a path from s to every node $v \in T$, and not to any other node in S. For fixed S and L_S, the sets $\mathcal{A}(L_S, S, T)$ parametrized by T partition the space of DAGs on N compatible with L_S. Figure 6.1 shows an example that illustrates the partitioning of the sets of DAGs by $\mathcal{A}(L_S, S, T)$.

The dynamic programming algorithm will compute a function $g_s(S, T)$, defined for all $S \subseteq N$ and $T \subseteq S$ by

$$g_s(S,T) \;\; = \;\; \sum_{L_S} \sum_{A_S \in \mathcal{A}(L_S,S,T)} \prod_{v \in S} \rho_v(L_v) \beta_v(A_v) \ ,$$

where the outer summation is over all linear orders L_S on S. Intuitively, $g_s(S, T)$ is the sum of $p(A, D, L)$ over all DAGs A_S and linear orders L, with $A_S \subseteq L$, such that S are the first nodes in the order L and there is a path from s to $v \in S$ in A_S if and only if $v \in T$. That the values $g_s(S, T)$ are sufficient for computing the target quantity $p(s \rightsquigarrow t, D)$ is shown by the following result.

Lemma 6.1

$$p(s \rightsquigarrow t, D) = \sum_{T : s,t \in T} g_s(N, T) \ .$$

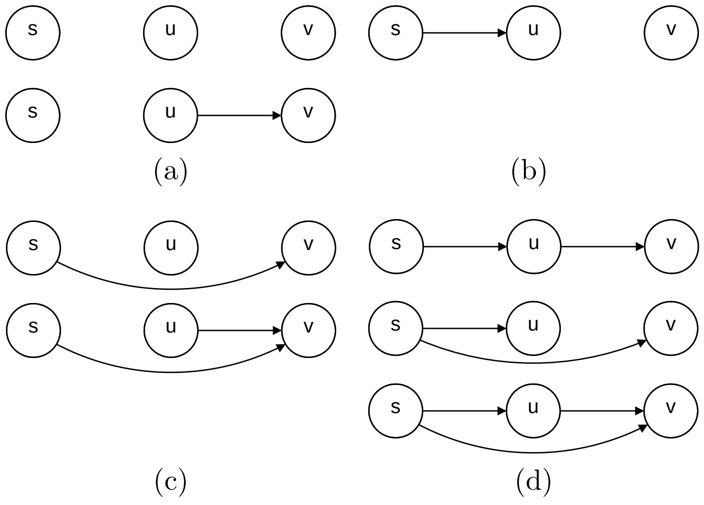


Figure 6.1: DAGs on node set $S = \{s, u, v\}$ compatible with linear order $L_S = \{ss, su, sv, uu, uv, vv\}$ partitioned according to $\mathcal{A}(L_S, S, T)$ given source node s. (a) $T = \{s\}$, (b) $T = \{s, u\}$, (c) $T = \{s, v\}$, and (d) $T = \{s, u, v\}$.

Proof. The definitions directly yield

$$p(s \rightsquigarrow t, D) = \sum_{L} \sum_{\substack{A \subseteq L \\ s \rightsquigarrow t \text{ in } A}} \prod_{v \in N} \rho_v(L_v)\beta_v(A_v)\,,$$

the outer summation being over all linear orders L on N.

We next break the inner summation into two nested summations by observing that the sets $\mathcal{A}(L, N, T)$, for $s, t \in T$, form a partition of the set $\mathcal{A}(L) = \{A \subseteq L : s \rightsquigarrow t \text{ in } A\}$: indeed, each DAG $A \in \mathcal{A}(L)$ determines precisely one node set T such that A contains a path from s to v for the nodes in $v \in T$ and no paths from s to any other nodes. Thus we have

$$\begin{aligned} p(s \rightsquigarrow t, D) &= \sum_{L} \sum_{T: s,t \in T} \sum_{A \in \mathcal{A}(L,S,T)} \prod_{v \in N} \rho_v(L_v)\beta_v(A_v) \\ &= \sum_{T: s,t \in T} \sum_{L} \sum_{A \in \mathcal{A}(L,S,T)} \prod_{v \in N} \rho_v(L_v)\beta_v(A_v) \\ &= \sum_{T: s,t \in T} g_s(N, T)\,. \end{aligned}$$

This completes the proof. □

From the algorithmic point of view, the pair (S, T) is sufficient for enabling a factorization of the sum over the A_S into independent sums over the parent sets A_v, for $v \in S$. Indeed, we have the following recurrence.

Lemma 6.2

$$\begin{aligned} g_s(S,T) &= 1 \quad \textit{for } S = \emptyset \textit{ and } T = \emptyset\,, \\ g_s(S,T) &= 0 \quad \textit{for } s \in T \textit{ and } (s \in S \textit{ or } T \neq \emptyset)\,, \\ g_s(S,T) &= \sum_{v \in S} g_s(S \setminus \{v\}, T \setminus \{v\}) \rho_v(S \setminus \{v\}) \bar{\alpha}_v(S,T) \qquad \textit{otherwise,} \end{aligned}$$

where

$$\bar{\alpha}_v(S,T) = \begin{cases} \displaystyle\sum_{\substack{A_v \subseteq S \setminus \{v\} \\ A_v \cap T \neq \emptyset}} \beta_v(A_v) & \textit{if } v \in T,\ v \neq s, \\ \displaystyle\sum_{A_v \subseteq (S \setminus \{v\}) \setminus T} \beta_v(A_v) & \textit{if } v \in S \setminus T \textit{ or } v = s. \end{cases}$$

Proof. Proof is by straightforward induction on the size of S. First, observe that the sum over L_S in the definition of $g_s(S,T)$ breaks into a double-summation, in which the outer summation is over the last node $v \in S$ in the order L_S and the inner summation is over all linear orders, $L_{S \setminus \{v\}}$, on the remaining nodes $S \setminus \{v\}$. Second, observe that the summation over $A_S \in \mathcal{A}(L_S, S, T)$ breaks into a double-summation, in which the outer summation is over the DAGs $A_{S \setminus \{v\}} \in \mathcal{A}\big(L_{S \setminus \{v\}}, S \setminus \{v\}, T \setminus \{v\}\big)$ and the inner summation is over the parent sets $A_v \subseteq S \setminus \{v\}$ satisfying the requirement that (a) if there is no path from s to v, that is, $v \notin T$, then there must be no path from s to u for any parent $u \in A_v$ of v, and (b) if there exists a path from s to v, that is, $v \in T$, then there must exist a path from s to u for at least one parent u of v. □

Figure 6.2 illustrates the requirements on choosing parent sets for the node v in the last equation in Lemma 6.2. In Figure 6.2(a) $v \in T$ meaning that it is required that there is a path from s to v. Now, we can choose any parent set for v as long as at least one of the parents is in T. On the other hand, in Figure 6.2(b) $v \notin T$, and thus it is required that there is no path from s to v. Now, we have to choose the parents of v from $S \setminus T$.

The evaluation of the values $g_s(S,T)$ using the recurrence is complicated by the fact that the inner summation, $\bar{\alpha}_v(S,T)$, is over exponentially many sets A_v and, furthermore, there is a condition that depends not only on the set S but the set T. Fortunately, the inner summation can be precomputed for each $v \in N$ and $S \in N \setminus \{v\}$. Indeed, if $v \notin T$, then the sum is over all

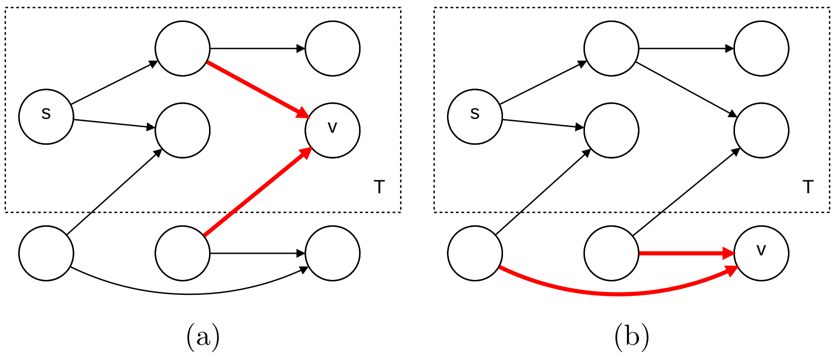


Figure 6.2: Choosing parent sets for a node $v \in S$ when (a) $v \in T$ and (b) $v \notin T$.

subsets A_v of $(S \setminus \{v\}) \setminus T$; if $v \in T$, then the sum is over all the remaining subsets of $S \setminus \{v\}$. Thus, it suffices to precompute the zeta transform

$$\alpha_v(U) = \sum_{A_v \subseteq U} \beta_v(A_v)$$

for all $U \subseteq N \setminus \{v\}$. The sums $\bar{\alpha}_v(S, T)$ for the cases $v \notin T$ and $v \in T$ are then obtained as $\alpha_v((S \setminus \{v\}) \setminus T)$ and $\alpha_v(S \setminus \{v\}) - \alpha_v((S \setminus \{v\}) \setminus T)$, respectively. Recall from Section 2.3.2 that the zeta transform of β_v can be computed, given β_v, by the fast zeta transform algorithm in $O(n2^n)$ time and $O(2^n)$ space.

After computing the probability $Pr(s \rightsquigarrow t, D)$ we get the posterior probability $Pr(s \rightsquigarrow t | D)$ by normalizing, that is,

$$Pr(s \rightsquigarrow t | D) = Pr(s \rightsquigarrow t, D) / Pr(D).$$

The probability $Pr(D)$ can be computed using the recurrence (2.7) in Section 2.3.2 with a trivial indicator function.

The following theorem summarizes the time and space bounds of the algorithm.

Theorem 6.3 *Given the local scores $Pr(D_v | D_{A_v}, A_v)$ for all v and $A_v \subseteq N \setminus \{v\}$, the ancestor relation problem for any pair $s \rightsquigarrow t$, $s, t \in N$ can be solved in $O(n3^n)$ time and $O(3^n)$ space. Further, the ancestor relation problem for all $n(n-1)$ pairs $s \rightsquigarrow t$, $s, t \in N$ can be solved in $O(n^2 3^n)$ time and $O(3^n)$ space.*

Proof. Let us consider the computation of the posterior probability of one ancestor relation. The precomputation of the inner sum requires n zeta transforms and thus takes $O(n^2 2^n)$ time and $O(n2^n)$ space. The value of $g_s(S,T)$ is computed and stored for $\sum_{i=0}^{n} \binom{n}{i} 2^i = 3^n$ set pairs (S,T); the identity holds by the binomial theorem, see the equation (A.1) in Appendix A. Given the precomputations, each (S,T) requires $O(n)$ time. Finally, the posterior probability for $s \rightsquigarrow t$ is obtained by summing the values $g_s(N,T)$ over $O(2^n)$ sets $\{t\} \subseteq T \subseteq N$. Thus the total time requirement is $O(n3^n)$ and the space requirement $O(3^n)$.

The posterior probabilities for a fixed source node s and all different target nodes t can be computed as above with the exception that the last step is repeated $n-1$ times. Thus the time and space requirement stay at $O(n3^n)$ and $O(3^n)$, respectively. For computing the posterior probabilities for all pairs, it suffices to repeat the previous procedure for each possible source node s, that is, n times. The time requirement is, thus, $O(n^2 3^n)$ and the space requirement $O(3^n)$. $\square$

6.2 Empirical Results

Now we study how learning ancestor relations performs in practice. Our approach is to generate data from a Bayesian network, called the *ground truth*, and compare the learned arcs and ancestor relations to the ground truth. Obviously, the learning performance is not expected to be perfect: when there are unobserved nodes at work, we easily learn arcs that are not present in the ground truth; this happens especially when an unobserved node is a common parent of two nodes that are not connected by an arc; namely, the two nodes are marginally dependent, and thus, in absence of the common parent, it is likely that we learn an arc between them, a false positive.

On the other hand, we may expect that much of the structure can be learned even in the presence of unobserved nodes. For example, if an unobserved node has exactly one child and one parent in the ground truth, both observed, then it is likely that the two arcs through the unobserved node in the middle will be just contracted to a single arc, which encodes a correct ancestor relation. We call the graph obtained from the ground truth by such contractions—that is, by connecting each parent of an unobserved node to every child of the node—the *shrunken ground truth.* The shrunken ground truth captures the ancestor relations of the ground truth: if a node s is an ancestor of a node t in the ground truth it is an ancestor of t in the shrunken ground truth as well. Likewise, if s is not an ancestor of t in the ground truth, it will not be an ancestor of t in the shrunken ground truth, either. Thus, the shrunken ground truth will serve as the representation of the true ancestor relations of the underlying Bayesian network.

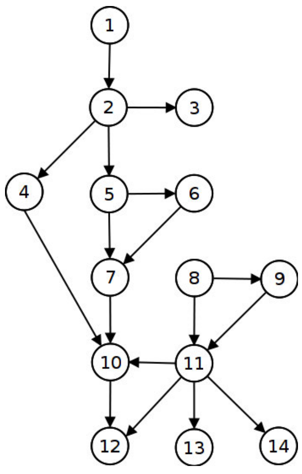
The algorithm of Section 6.1 for Bayesian learning of ancestor relations was implemented in Matlab. The experiments discussed next were conducted using the well-known BDeu (Bayesian Dirichlet equivalence uniform) score [13, 47] with the equivalent sample size of 1, a uniform prior over linear orders on the nodes, and a uniform prior over parent sets of size at most a user-defined bound, which was set to 6.

To illustrate the challenges that are faced while learning ancestor relations, a case-study is presented in the following section. Then, the statistical efficiency of learning ancestor relations is investigated by conducting a simulation study in Section 6.2.2. Finally, results on real life data are presented in Section 6.2.3.

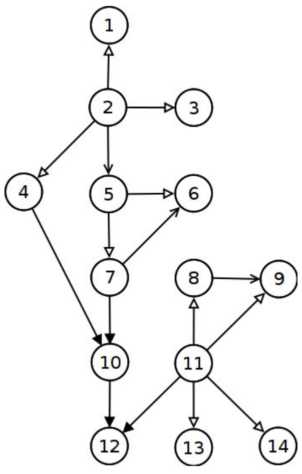
6.2.1 Challenges of Learning Ancestor Relations

It is instructive to examine some representative challenges that we face when learning ancestor relations and arcs. We consider a Bayesian network whose DAG is shown in Figure 6.3(a). All 14 variables are binary. The parameters of the network, that is, the probability of a node taking the value 1 given a particular value combination of its parents was drawn uniformly at random from the range $[0, 1]$ for each node and value combination of its parents. We generated $10,000$ samples from the Bayesian network and learned ancestor relations from the data. Note that there are 16 arcs and 39 ancestor–descendant pairs in the ground truth. The DAG has quite a large Markov equivalence class, 140 graphs in total, and so one cannot expect to deduce ancestor relations from a single MAP DAG reliably.

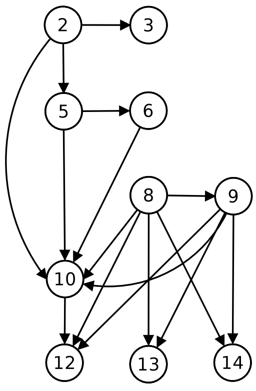
For clarity of presentation, the findings are discussed mainly in terms of arcs instead of ancestor relations. Figure 6.3(b) shows arcs that are assigned a posterior probability of 0.5 or larger. Suppose we claim every arc or ancestor relation with probability 0.5 or larger to be present. Then, in total there are 12 true positive arcs, 4 false positive arcs, 20 true positive ancestor relations, and 4 false positive ancestor relations. Inspection reveals that the ancestor relation errors are due to a few flipped arcs. For example, in the ground truth there is a path from node 1 to eight different nodes. Thus, flipping the arc from 1 to 2 causes one false positive and eight false negative ancestor relations. While arc errors are rather independent, one flipped arc can lead to numerous ancestor relation errors, as seen earlier. It should also be noted that arc flips that are prone to cause a larger number of ancestor relation errors are also more probable, namely, an arc is easily flipped when it does not break or create any v-structure, which is typically the case when one of the nodes is a source node in the ground truth.



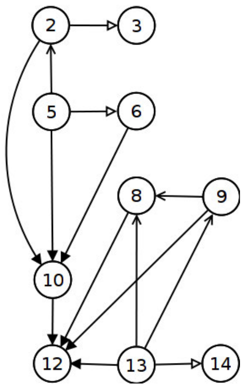
(a)



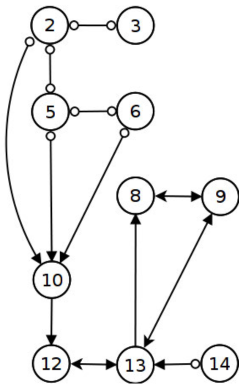
(b)



(c)



(d)



(e)

Figure 6.3: Graphs. (a) The ground truth, from which 10, 000 samples were generated. (b) Arcs with posterior probability at least 0.5. The arrowheads $>$, $\triangleright$, and $\blacktriangleright$ indicate that the probability is in the interval $(0.5, 0.8]$, $(0.8, 0.99]$, or $(0.99, 1]$, respectively. (c) The shrunken ground truth when nodes 1, 4, 7, and 11 are not observed. (d) Arcs with posterior probability at least 0.5 when nodes 1, 4, 7, and 11 are not observed. (e) A partially directed graph learned using the FCI algorithm when nodes 1, 4, 7, and 11 are not observed.

The presence of unobserved nodes leads to claiming arcs between nodes that are only marginally dependent. Discarding nodes 1, 4, 7, and 11 leads to the shrunken ground truth shown in Figure 6.3(c). In Figure 6.3(d) we see a DAG constructed from the arcs with probability 0.5 or larger. Node 1 does not have children, so its disappearance should not affect the structure among the rest of the nodes. However, the removal of nodes 4, 7, and 11 affects the rest of the nodes: For instance, node 11 is a common cause of nodes 13 and 14, and so an arc appears between nodes 13 and 14. Also, nodes 5 and 6, which are parents of node 7 in the ground truth, have become parents of node 10, a child of node 7 in the ground truth. Similarly, the removal of node 4 also leads to appearance of some direct arcs from its parents to its children. After discarding the unobserved nodes, the shrunken ground truth contains 14 arcs and 18 ancestor relations. The algorithm finds 8 true positive arcs, 6 false positive arcs, 11 true positive ancestor relations, and 7 false positive ancestor relations. This suggests that ancestor relations can sometimes be learned as well as individual arcs.

For comparison, also a *partial ancestral graph* (PAG) [87] was learned from the data with unobserved nodes using the fast causal inference (FCI) algorithm [100], which is designed for causal discovery with unobserved variables. The output PAG is shown in Figure 6.3(e). An arc marked with two arrowheads indicates that the algorithm claims the two nodes have a common (unobserved) cause; the symbol $\circ$ is a wildcard, indicating that there can be an arrowhead or there is no arrowhead. The results are generally in good agreement with the ground truth. The FCI algorithm is able to detect the unobserved parent of nodes 12 and 13. However, it is not sure whether there is an unobserved parent between nodes 13 and 14, and it is unable to detect the unobserved parent between nodes 12 and 14. It also finds an unobserved parent between nodes 8 and 9, which is not in agreement with the ground truth. As the wildcards assigned by the FCI algorithm do not quantify the uncertainty about the associated arcs, but the algorithm is ignorant regarding some ancestor relations, the algorithm may loose statistical power in detecting such relations; this issue will be further examined and discussed in the next section.

6.2.2 A Simulation Study

Synthetic data consisting of one hundred Bayesian networks on 14 binary nodes and maximum indegree 4, each with $10,000$ data points were obtained as follows.

1. Draw a linear order L on the node set $\{1, 2, \ldots, 14\}$ uniformly at

random (u.a.r.).

2. For each node v independently:

   (a) let d_v be the number of predecessors of v in L;

   (b) draw the number of parents of v, denoted as n_v, from $\{0, 1, \ldots, \min\{4, d_v\}\}$ u.a.r.;

   (c) draw the n_v parents of v from the predecessors of v in L u.a.r.;

   (d) for each value configuration of the parents: draw the probability of a sample getting the value 1 from the uniform distribution on range $[0, 1]$.

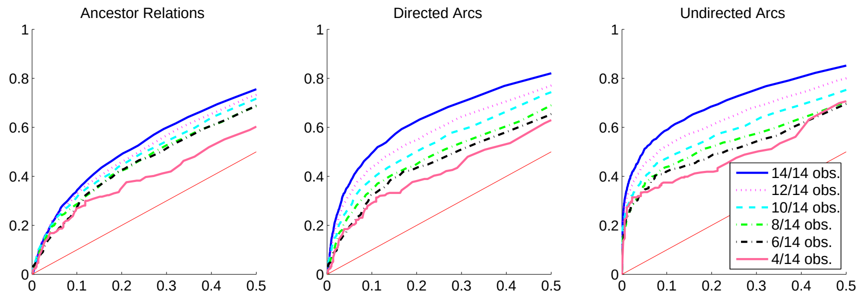
3. Draw $10,000$ samples independently from the Bayesian network.

From each data set 24 subsets were generated by discarding $\ell = 0, 2, 4, 6, 8, 10$ randomly picked nodes and the associated data, and by including the first $m = 100, 500, 2000, 10000$ data points.

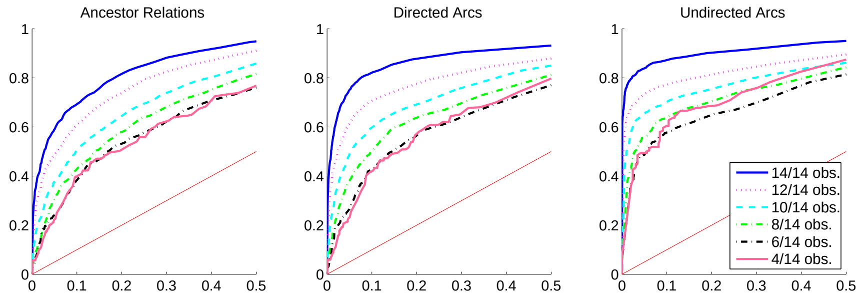
Bayesian averaging was applied to each data set and the performance of learning arcs and ancestor relations is summarized by ROC curves in Figure 6.4. The ROC curve is obtained by setting a threshold for the posterior probability (of arcs or ancestor relations), and every time the posterior probability exceeds the threshold, we claim the respective arc or ancestor relation is present. Comparing these claims to the arc and ancestor relations that actually hold in the (shrunken) ground truth, we obtain true positives (TP) and false positives (FP) rates. By varying the threshold the pairs of these rates form a ROC curve, which shows the learning power (TP rate) as a function of the FP rate. The running times of the algorithm for 10, 12, and 14 observed nodes were roughly 3 minutes, 40 minutes, and 8 hours, respectively.

As expected, the more data one has, the easier it is to learn both ancestor relations and arcs. Likewise, the task becomes harder as the number of unobserved nodes grows. Note that Koivisto [62] has studied learning undirected edges with no unobserved nodes with data generated by a procedure that is similar to the one presented here. The results presented in Figure 6.4 for undirected graphs with no unobserved nodes are in good agreement with Koivisto's results. The results (Figure 6.4) also suggest that the power of learning directed and undirected arcs is about the same, however, the power of learning ancestor relations being slightly smaller.

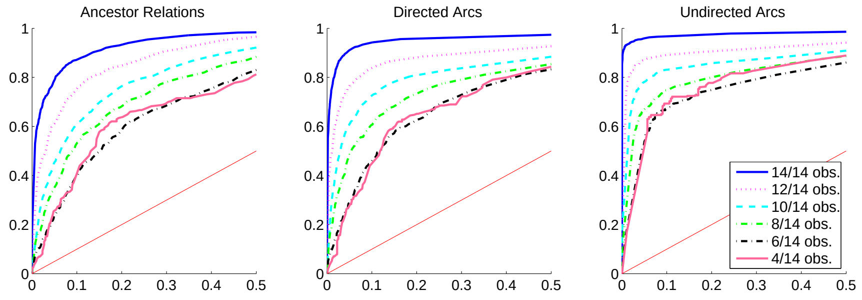
The Bayesian averaging approach was then compared to deducing structural features from a single MAP DAG. Two ways to pick a MAP DAG were considered: an optimistic and a random approach. In the optimistic



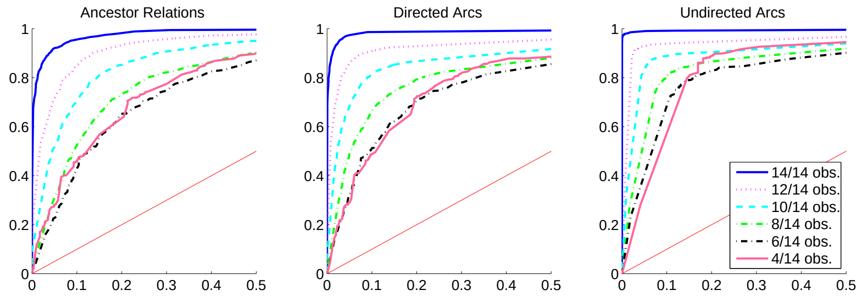
(a)



(b)



(c)



(d)

Figure 6.4: ROC curves. The data contain (a) 100, (b) 500, (c) 2000 or (d) 10,000 samples over 14 nodes. The straight red line is the curve obtained by random guessing. The data-generating graphs contained on average 23.7 arcs and the shrunken ground truths on average 19.7, 15.9, 11.1, 7.0, and 3.3 arcs for 12, 10, 8, 6, and 4 observed nodes, respectively.

approach, one chooses a member of the Markov equivalence class of a MAP DAG that yields the largest true positives rate, and uses its true and false positives rates. This approach is arguably unrealistic in practice but serves as an upper bound for any approach based on a single MAP DAG. In the random approach, the true and false positives rates over all DAGs in the Markov equivalence class of a MAP DAG are averaged; the averaged rates correspond to the respective expectations if one picks such a DAG at random. The true and false positives rates for these two approached are shown in Tables 6.1 and 6.2; column "diff." shows the difference between the true positives rates of the MAP DAG approach and the Bayesian averaging approach (the false positives rate being matched, of course); a negative value indicates that the Bayesian averaging approach is more powerful. Figures 6.5 and 6.6 illustrate the differences between the true positive rates.

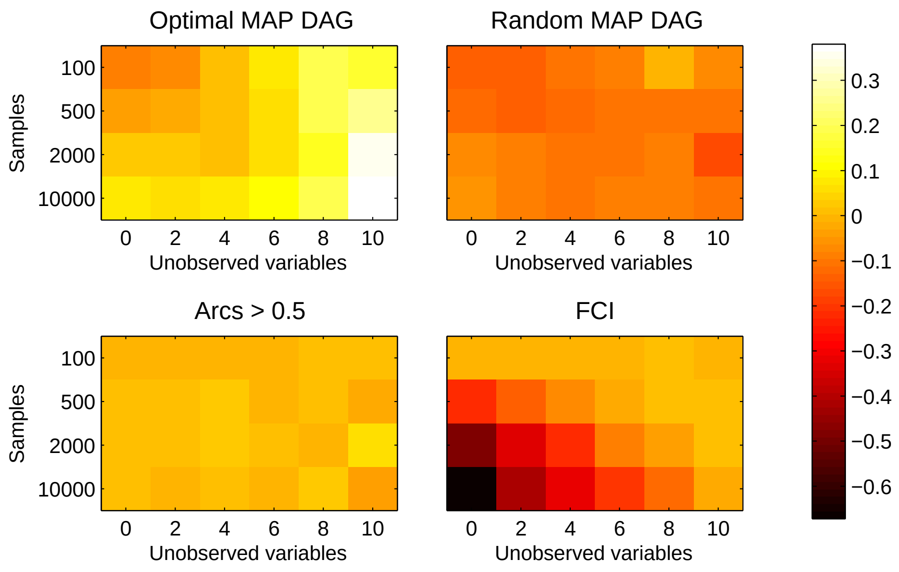


Figure 6.5: The differences in true positive rates between various methods and full Bayesian averaging for ancestor relations. The negative values imply that full Bayesian averaging is more powerful.

The results suggest that the random MAP DAG approach performs significantly worse than Bayesian averaging. On the other hand, the optimistic MAP DAG approach sometimes performs better than Bayesian averaging, especially when the data are abundant and there are many unobserved

Table 6.1: Comparison of TP and FP rates for ancestor relations

		Opt. MAP DAG			Rand. MAP DAG			Arcs > 0.5			FCI		
m	ℓ	TP	FP	diff.	TP	FP	diff.	TP	FP	diff.	TP	FP	diff.
100	0	0.41	0.19	-0.09	0.37	0.21	-0.14	0.20	0.04	-0.01	0.002	0.000	0.002
100	2	0.35	0.16	-0.07	0.30	0.18	-0.14	0.17	0.03	-0.01	0.001	0.000	-0.001
100	4	0.34	0.11	0.01	0.27	0.14	-0.11	0.16	0.03	-0.01	0.002	0.000	0.002
100	6	0.34	0.08	0.07	0.24	0.12	-0.08	0.15	0.03	-0.00	0.002	0.000	0.002
100	8	0.36	0.05	0.19	0.23	0.09	-0.01	0.12	0.03	0.00	0.003	0.000	0.003
100	10	0.31	0.04	0.16	0.17	0.09	-0.06	0.12	0.02	0.01	0.000	0.000	0.000
500	0	0.65	0.10	-0.04	0.61	0.13	-0.12	0.58	0.04	0.01	0.014	0.002	-0.218
500	2	0.61	0.11	-0.02	0.54	0.14	-0.13	0.50	0.05	0.01	0.014	0.002	-0.143
500	4	0.53	0.11	0.01	0.45	0.14	-0.12	0.42	0.06	0.02	0.010	0.001	-0.073
500	6	0.50	0.10	0.06	0.40	0.14	-0.11	0.35	0.06	-0.01	0.011	0.000	-0.028
500	8	0.48	0.07	0.19	0.33	0.12	-0.10	0.27	0.06	0.00	0.014	0.000	0.014
500	10	0.51	0.05	0.26	0.32	0.12	-0.10	0.27	0.06	-0.02	0.005	0.000	0.005
2000	0	0.84	0.06	0.02	0.78	0.08	-0.08	0.78	0.05	0.01	0.048	0.004	-0.482
2000	2	0.76	0.10	0.02	0.70	0.12	-0.09	0.69	0.07	0.02	0.047	0.005	-0.329
2000	4	0.67	0.12	0.01	0.60	0.15	-0.11	0.60	0.09	0.02	0.041	0.007	-0.217
2000	6	0.64	0.12	0.06	0.54	0.16	-0.10	0.51	0.09	0.01	0.037	0.004	-0.090
2000	8	0.59	0.12	0.14	0.45	0.17	-0.09	0.40	0.10	-0.00	0.020	0.002	-0.033
2000	10	0.69	0.07	0.36	0.44	0.18	-0.18	0.41	0.09	0.07	0.005	0.000	0.005
10 000	0	0.93	0.02	0.07	0.86	0.06	-0.06	0.87	0.02	0.00	0.129	0.011	-0.660
10 000	2	0.86	0.08	0.06	0.79	0.11	-0.08	0.79	0.07	-0.00	0.121	0.010	-0.410
10 000	4	0.80	0.11	0.07	0.70	0.15	-0.11	0.70	0.09	0.01	0.100	0.015	-0.326
10 000	6	0.73	0.13	0.11	0.62	0.18	-0.10	0.60	0.13	0.00	0.086	0.010	-0.199
10 000	8	0.72	0.14	0.19	0.57	0.20	-0.09	0.54	0.14	0.02	0.037	0.006	-0.125
10 000	10	0.84	0.09	0.38	0.57	0.21	-0.11	0.54	0.14	-0.03	0.014	0.002	-0.025

Table 6.2: Comparison of TP and FP rates for arcs

		Opt. MAP DAG			Rand. MAP DAG			Arcs > 0.5			FCI		
m	ℓ	TP	FP	diff.	TP	FP	diff.	TP	FP	diff.	TP	FP	diff.
100	0	0.34	0.05	-0.05	0.31	0.06	-0.10	0.26	0.03	0.00	0.003	0.000	0.003
100	2	0.30	0.06	-0.04	0.26	0.06	-0.11	0.22	0.02	0.00	0.002	0.000	-0.001
100	4	0.28	0.05	0.01	0.23	0.06	-0.08	0.18	0.02	0.00	0.003	0.000	0.003
100	6	0.28	0.04	0.05	0.21	0.06	-0.06	0.16	0.03	0.00	0.002	0.000	0.002
100	8	0.30	0.03	0.15	0.20	0.06	-0.02	0.13	0.03	0.00	0.003	0.000	0.003
100	10	0.31	0.03	0.15	0.18	0.07	-0.05	0.14	0.03	0.00	0.000	0.000	0.000
500	0	0.64	0.03	-0.03	0.60	0.03	-0.09	0.62	0.02	0.00	0.023	0.001	-0.273
500	2	0.55	0.03	0.00	0.50	0.04	-0.09	0.52	0.03	0.00	0.020	0.002	-0.153
500	4	0.46	0.04	0.02	0.41	0.05	-0.09	0.42	0.03	0.00	0.014	0.001	-0.076
500	6	0.42	0.04	0.06	0.34	0.06	-0.08	0.35	0.04	0.00	0.012	0.000	-0.026
500	8	0.42	0.04	0.18	0.30	0.07	-0.04	0.28	0.05	0.00	0.016	0.000	0.016
500	10	0.49	0.04	0.28	0.32	0.08	-0.08	0.30	0.07	0.00	0.005	0.000	0.005
2000	0	0.83	0.02	0.03	0.78	0.02	-0.06	0.81	0.02	0.00	0.075	0.003	-0.511
2000	2	0.71	0.03	0.02	0.66	0.04	-0.06	0.69	0.03	0.00	0.067	0.003	-0.319
2000	4	0.60	0.05	0.00	0.55	0.06	-0.08	0.59	0.04	0.00	0.054	0.005	-0.206
2000	6	0.55	0.05	0.05	0.47	0.07	-0.07	0.50	0.06	0.00	0.042	0.004	-0.088
2000	8	0.51	0.07	0.16	0.40	0.10	-0.05	0.41	0.08	0.00	0.023	0.002	-0.029
2000	10	0.65	0.06	0.33	0.43	0.12	-0.09	0.44	0.10	0.00	0.005	0.000	0.005
10 000	0	0.93	0.01	0.08	0.87	0.02	-0.04	0.89	0.01	0.00	0.173	0.006	-0.672
10 000	2	0.83	0.03	0.07	0.77	0.04	-0.05	0.80	0.03	0.00	0.146	0.006	-0.395
10 000	4	0.75	0.05	0.08	0.67	0.06	-0.06	0.70	0.05	0.00	0.111	0.011	-0.304
10 000	6	0.67	0.07	0.09	0.58	0.09	-0.06	0.61	0.08	0.00	0.090	0.009	-0.190
10 000	8	0.64	0.09	0.15	0.52	0.12	-0.04	0.54	0.11	0.00	0.040	0.006	-0.118
10 000	10	0.79	0.08	0.34	0.56	0.15	-0.09	0.58	0.14	0.00	0.015	0.002	-0.027

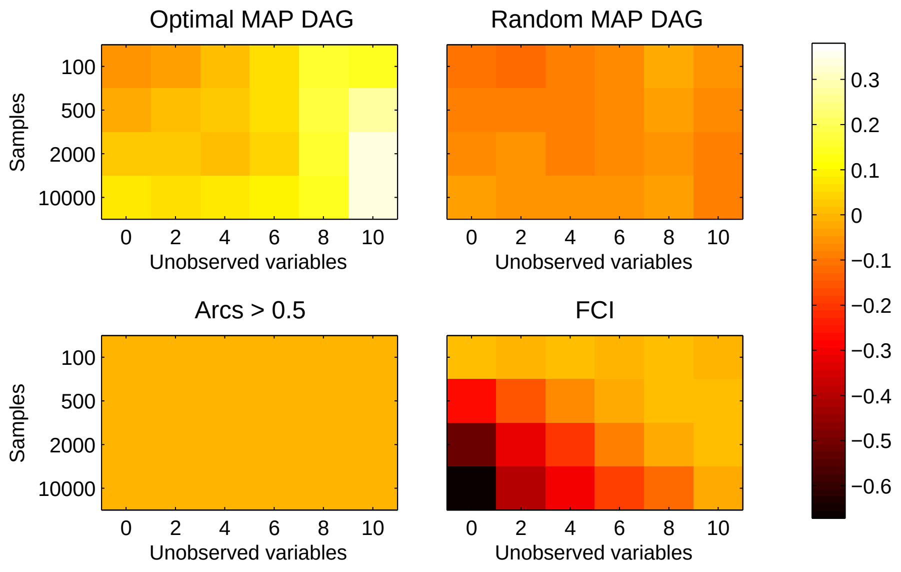


Figure 6.6: The differences in true positive rates between various methods and full Bayesian averaging for arcs. The negative values imply that full Bayesian averaging is more powerful.

nodes.

Furthermore, Bayesian averaging was compared to deducing ancestor relations from the arc probabilities. For this purpose, a graph was constructed to consist of the arcs whose posterior probability was larger than 0.5, that is, the arc is more likely to be present than absent, and the ancestor relations were deduced from this graph. The results (Tables 6.1 and 6.2, Figures 6.5 and 6.6) show that the performance of deducing ancestor relations from arcs does not differ significantly from learning ancestor relations directly.

The aforementioned approach was further compared to direct learning of ancestor relations. To this end, it was assumed that only the ancestor relations whose probability is more than 0.5 exist. The ancestor relation predictions for deducing the ancestor relations from arcs and the direct computation of ancestor relations were cross-tabulated; see Table 6.3. Table 6.3 shows the average number of the node pairs for which either both methods, only deducing from arc probabilities, only the direct computation of ancestor relation probabilities or neither method claims an ancestor relation to be present. Table 6.3 also shows the probability that the claim made by the direct computation is correct; "." denotes that no claims falling into the particular category were made. Most of the time, both methods make the same predictions. Whenever the predictions differ, the prediction by direct computation is usually slightly more probable to be correct. Also notice that the two methods follow each other closely with larger datasets.

Table 6.3: Ancestor Relations predicted by arcs and direct computation

		Pred. Ancestor Relations				Corr. Pred. by Dir. Comp.			
m	ℓ	both	arcs	direct	none	both	arcs	direct	none
100	0	13.6	1.1	1.8	165.5	0.61	0.64	0.50	0.79
100	2	8.3	0.4	0.9	122.4	0.63	0.59	0.61	0.79
100	4	5.3	0.3	0.5	84.0	0.63	0.52	0.59	0.78
100	6	3.1	0.2	0.2	52.5	0.62	0.56	0.57	0.78
100	8	1.4	0.1	0.0	28.4	0.59	0.25	1.00	0.77
100	10	0.6	0.0	0.0	11.4	0.68	.	1.00	0.77
500	0	30.5	0.5	1.3	149.7	0.82	0.48	0.53	0.88
500	2	20.7	0.4	0.6	110.2	0.76	0.77	0.48	0.86
500	4	12.7	0.5	0.6	76.3	0.70	0.65	0.35	0.84
500	6	7.0	0.2	0.3	48.5	0.66	0.81	0.63	0.82
500	8	3.3	0.1	0.1	26.5	0.61	0.56	0.45	0.79
500	10	1.3	0.0	0.0	10.6	0.60	0.33	.	0.79
2000	0	39.7	0.2	0.4	141.8	0.85	0.82	0.39	0.93
2000	2	28.4	0.2	0.4	103.0	0.76	0.55	0.61	0.91
2000	4	18.6	0.2	0.4	70.8	0.69	0.47	0.30	0.88
2000	6	10.6	0.2	0.3	44.9	0.64	0.70	0.47	0.86
2000	8	5.2	0.1	0.1	24.6	0.58	0.89	0.54	0.82
2000	10	2.0	0.1	0.1	9.9	0.61	0.25	0.60	0.82
10000	0	40.9	0.1	0.4	140.7	0.92	0.60	0.61	0.96
10000	2	31.2	0.2	0.3	100.3	0.79	0.78	0.32	0.94
10000	4	21.6	0.1	0.2	68.0	0.71	0.60	0.36	0.91
10000	6	13.3	0.2	0.2	42.3	0.60	0.68	0.53	0.88
10000	8	7.2	0.0	0.1	22.7	0.57	0.75	0.44	0.85
10000	10	2.8	0.0	0.0	9.1	0.58	0.67	0.00	0.84

Bayesian averaging was also compared to the fast causal inference (FCI) method [100]; see Tables 6.1 and 6.2. It is quite challenging to make a fair comparison because FCI outputs a partial ancestral graph (PAG) that cannot be directly compared to a DAG. It was decided to ignore the wildcard arcs and claim only arcs and ancestor relations that FCI is sure about; this follows the approach of Spirtes et al. [101]. The results (Tables 6.1 and 6.2, Figures 6.5 and 6.6) show that FCI is very conservative: it does not make many mistakes but it often answers “don’t know”. This results in a relatively low statistical power of discovering arcs and ancestor relations, sometimes significantly lower than that of the Bayesian averaging approach (at matched FP rates).

One should notice, though, that FCI can discover unobserved nodes with some success. However, usually the unobserved nodes that FCI “finds,” do not seem to match the ones in the ground truth. For example, when the sample size is 2000 and there are no unobserved nodes, FCI finds on average 6.0 unobserved nodes. And when there are two unobserved nodes, only 11% of the “found” 4.8 unobserved nodes match the ground truth. In general, as the number of unobserved nodes increases, the number of found unobserved nodes decreases, but the percentage of correctly detected unobserved nodes increases; for example, when there are 8 unobserved nodes 54% of the claimed 0.7 unobserved nodes are correct.

6.2.3 Real Life Data

The Bayesian averaging algorithm was tested on two real-life datasets found from the UCI machine learning repository [37]: Adult (15 variables, 32,561 samples) and Housing (14 variables, 506 samples). All continuous variables were discretized to binary variables using the median as the cutpoint. Furthermore, in the Adult dataset the variable “native-country”, which had 40 distinct values, was transformed to a binary variable where 0 corresponded to value “USA” and 1 to all other values. For both datasets, the maximum indegree was set to be 4. Figures 6.7 and 6.8 show the graphs deduced from the arc probabilities.

Here, one does not know the ground truth and thus one has to resort to other comparisons. The goal was to investigate whether learning ancestor relations uncovers some information that cannot be obtained simply by analyzing the arc probabilities. To this end, ancestor relations were deduced both from the ancestor relations probabilities and the arc probabilities. For Adult, there are 210 potential ancestor relations. Both methods imply the presence of the same 79 ancestor relations. For Housing the methods are in almost as good agreement as for the Adult. For 71 ordered pairs, both methods claim that an ancestor relation is present and for 110 pairs that an ancestor relation is not present. There is, however, one node pair, ZN and RAD, for which deducing from arcs suggests that there is no ancestor relation while deducing from ancestor probabilities claims the opposite. This discrepancy is, however, due to the arbitrariness of the threshold:

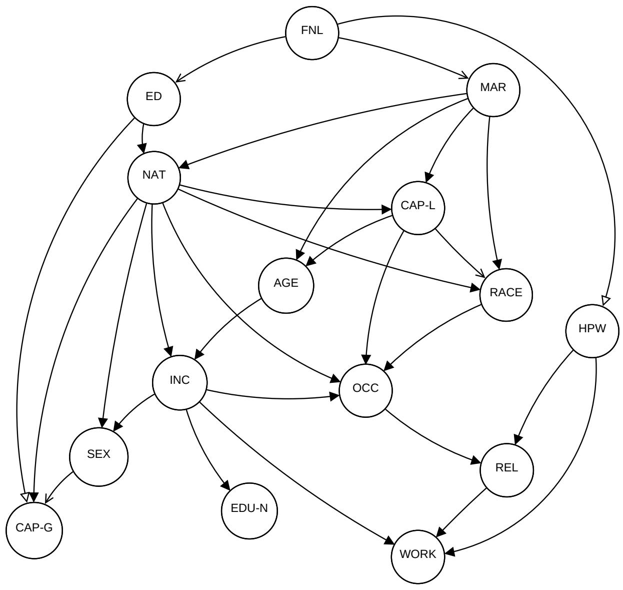


Figure 6.7: The ADULT data. Arcs with posterior probability at least 0.5. The arrowheads >, ▷, and ▶ indicate that the probability is in the interval $(0.5, 0.8]$, $(0.8, 0.99]$, or $(0.99, 1]$, respectively. INC = income, AGE = age, WORK = work class, FNL = fnlwgt, ED = education, ED-N = education-num, MAR = marital status, OCC = occupation, REL = relationship, RACE = race, SEX = sex, CAP-G = capital gain, CAP-L = capital loss, HPW = hours per week, NAT = native country.

the posterior probability of an arc from ZN to RAD was 0.49 while the probability of an ancestor relation ZN$\rightsquigarrow$RAD was 0.53. The experiments on real data suggest that the exact learning of the ancestor relations does not provide significant value added compared to deducing ancestor relations from arc probabilities.

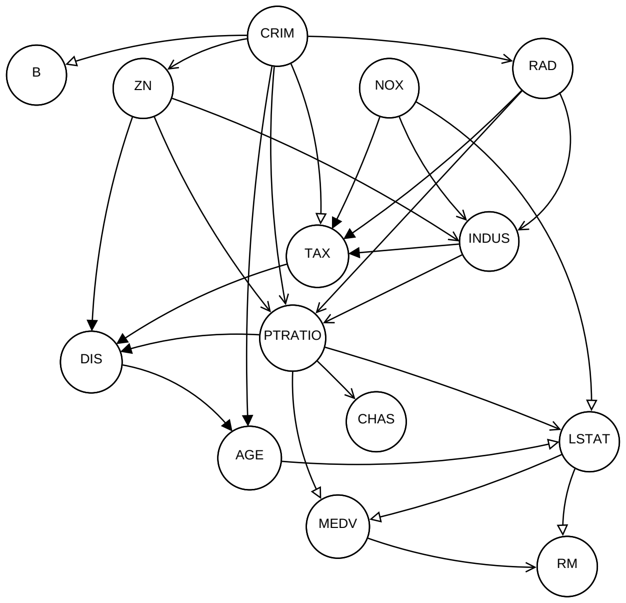


Figure 6.8: The Housing data. Arcs with posterior probability at least 0.5. The arrowheads >, ▷, and ▶ indicate that the probability is in the interval $(0.5, 0.8]$, $(0.8, 0.99]$, or $(0.99, 1]$, respectively. CRIM = per capita crime rate by town, ZN = proportion of residential land zoned for lots over 25,000 sq.ft., INDUS = proportion of non-retail business acres per town, CHAS = Charles River dummy variable (= 1 if tract bounds river; 0 otherwise), NOX = nitric oxides concentration (parts per 10 million), RM = average number of rooms per dwelling, AGE = proportion of owner-occupied units built prior to 1940, DIS = weighted distances to five Boston employment centres, RAD = index of accessibility to radial highways, TAX = full-value property-tax rate per \$10,000, PTRATIO = pupil-teacher ratio by town, B = $1000(\text{Bk} - 0.63)^2$ where Bk is the proportion of blacks by town, LSTAT = lower status of the population, MEDV = Median value of owner-occupied homes in \$1000's.

Chapter 7

Discussion

This thesis concerned two areas of structure discovery in Bayesian networks: space–time tradeoffs and learning ancestor relations. For space–time tradeoffs we introduced a variety of algorithmic schemes which yield varying time and space requirements for solving permutation problems in general and the OSD and FP problems in particular. A summary of the time and space bounds of the schemes presented in this thesis is shown in Figure 7.1. The main contribution, the partial order approach, applies to the space range between polynomial space[1] and $O^*(2^n)$. Especially, it applies to the space range that is most interesting in practical implementations, namely the space requirement between $O^*(2^{n/2})$ and $O^*(2^n)$. To analyze the efficiency of the presented schemes, we introduced the time–space product. Time–space products less than 4 are considered good as they imply that whenever the space requirement is halved the time requirement less than doubles. We have shown that when the available space is at least $O^*(2^{n/2})$, one can always (in theory) use schemes with time–space product less than 4. Although we did not present any scheme that works with space less than $O^*(2^{n/2})$ and has a time–space product less than 4, it seems that this can be achieved straightforwardly by combining the divide-and-conquer scheme with partial order schemes. This is, however, not particularly interesting because such schemes are infeasible in practice due to their large time requirements.

The motivation of the space–time tradeoffs comes from practical needs for solving large problem instances while guaranteeing that an optimum will be found. The presented algorithmic schemes address these issues in two ways. First, they reduce space, which is often the bottleneck, and

[1]Polynomial space is achieved with a POF that consists of the reorderings of a linear order.

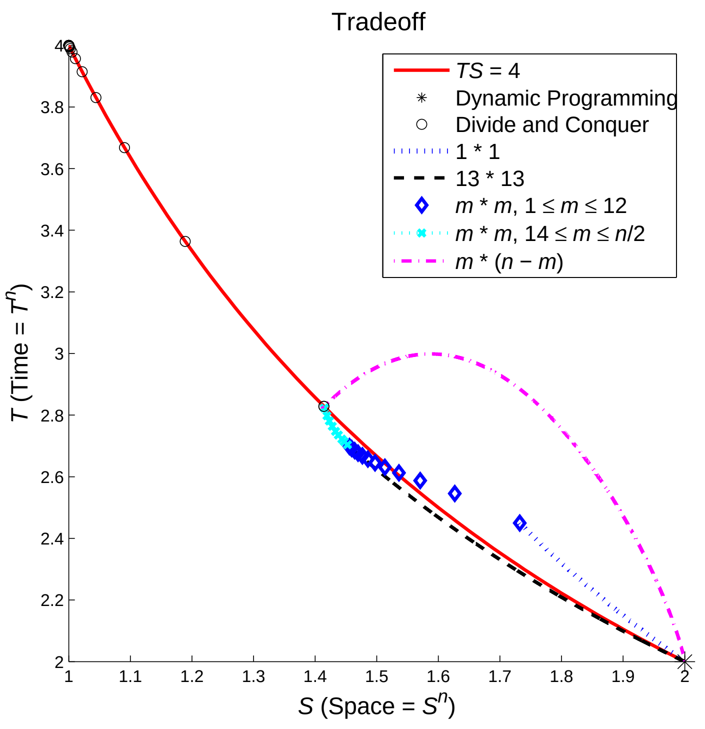


Figure 7.1: Comparison of time and space requirements of algorithmic schemes.

thus enable solving larger problem instances. Second, they can be easily parallelized which can be a serious advantage as large computer clusters with thousands of cores are becoming more common.

The results on space–time tradeoffs seem promising. However, this is not the end of the story as there are interesting open questions. For example, the partial order family which is optimal with respect to time–space product is not known. In Conjecture 5.17 we proposed that the $13*13$ scheme is optimal, however, proving or disproving this conjecture seems to be a cumbersome task.

It is worth noting that sometimes the phenomenon to be modeled has an inherent partial order and the modeler knows this partial order (or a part of it). This kind of precedence order is often known when dealing with, for example, causal discovery or time series. In such a case one needs to run the dynamic programming algorithm only for that particular partial order, thus saving both space and time, possibly significantly. In other words, the partial order approach generalizes the basic dynamic programming algorithm to fully exploit given precedence constraints, if any.

The previous observation concerning the partial order approach motivates the following research question about structure discovery in Bayesian networks. Sometimes it may be desired that the DAG obeys some structural constraint. For example, one may want the DAG to be compatible with some partial order as mentioned above. On the other hand, one may want to learn a sparse graph and thus limit the number of arcs. Further, one may want to guarantee that one is able to perform inference in the Bayesian network fast in the worst case, and thus bound the treewidth of the DAG by some constant[2]. The question is how the addition of constraints affects the running time.

As mentioned above, any constraint on the order of the nodes makes the problem easier to solve. However, in general adding more constraints does not necessarily make a problem easier to solve. For example, if one wants to learn a sparse graph, that is, a graph with at most s arcs, for some fixed s, one can modify the dynamic programming algorithm and keep track of the best network for each possible arc count. This increases both the time and the space requirement by a factor polynomial in n. On the other hand, in the case where one restricts the search space to consist of the DAGs that have treewidth[3] at most t for some constant t the effect on the running time remains unclear; currently algorithms for this problem are heuristics [30]. As both Treewidth and Osd are permutation problems, one might hope to be able to bundle these two problems together and solve them by dynamic programming. However, straightforward combinations seem to fail and it is an open question whether this variant can be solved in, say, $O^*(2^n)$ time.

Recall that solving the Osd, Fp, and ancestor relation problem requires considering of a superexponential number of DAGs. However, modularity assumptions allow us to solve these problems significantly faster. Thus, this thesis serves as an example, how hard problems can be made easier (but not necessarily easy) by exploiting the modularity of the problem. In the Osd and Fp problems, we made several modularity and independence

[2]The exact inference in Bayesian networks with unbounded treewidth is not possible in polynomial time in the worst case, unless the exponential time hypothesis (ETH) fails [67]. Also the time requirement of some common inference algorithms like variable elimination is lower bounded by a factor exponential with respect to treewidth.

[3]The treewidth of a DAG equals to the treewidth of its moralized graph. The moralized graph of a DAG A is obtained by taking the skeleton of A and adding an edge (if such an edge does not already exist) between nodes that have a common child in A.

assumptions, which enabled us to formulate the problems as permutation problems and thus allowed us to use dynamic programming across only the 2^n subsets of the elements. However, here just assuming that something is modular is not enough: one needs to assume the right kind of modularity. To see this, consider the structure prior in the Osd and Fp problems. In Osd we assumed a modular structure prior. By contrast, in Fp we assumed an order-modular structure prior. However, if the modularity assumptions are reversed, that is, one assumes an order-modular prior for Osd and a modular prior for Fp, both problems seem to become more difficult as the fastest known algorithm for Fp with a modular prior runs in $O(3^n)$ time [105] and solving Osd with an order-modular prior seems to be a complicated task. In ancestor relations problem we made the same assumptions as in Fp. However, ancestor relations are not modular features and thus the ancestor relation problem seems inherently harder than Fp; the Bayesian averaging algorithm for ancestor relation problem runs in $O(3^n)$ time. This further exemplifies how modularity enables faster computation.

The study of learning ancestor relations was motivated by the need for structural features that are preserved when there are unobserved nodes in play. Here, we assumed that the arcs of a DAG represent the direct causal relations between nodes. We developed an algorithm for computing posterior probabilities of ancestor relations. As far as we know, this is the first non-trivial exact algorithm to compute posterior probabilities of nonmodular features. Our experiments show that the Bayesian model averaging approach outperforms the obvious rivals: deducing the ancestor relations from a MAP DAG and a constraint-based FCI algorithm [101]. The Bayesian averaging algorithm has, however, a big disadvantage compared to the competitors: time consumption. The $O(3^n)$ time requirement makes the Bayesian averaging algorithm infeasible for data sets with more than 20 variables and therefore limits the practical usefulness of the algorithm. Luckily, partial Bayesian averaging, that is, first inferring arcs based on their marginal posterior probabilities and then deducing ancestor relations from the so-constructed graph, seems to perform almost as good as full Bayesian averaging. Therefore, one could resort to partial Bayesian averaging whenever full Bayesian averaging takes too much time. This finding suggests that the arc probabilities convey almost all the information relevant for learning ancestor relations. Although some may say that the competitiveness of partial Bayesian averaging undermines the usefulness of the comparatively slow full Bayesian averaging algorithm, it should be noted that the insight about partial Bayesian averaging was achieved because we were able to compute the exact ancestor relation probabilities.

An obvious target for an improvement in learning ancestor relations is to decrease the time requirement. Besides that, accommodating our algorithm to handle priors that are not order-modular would be desirable. For the empirical point of view it would be interesting to see how well some existing score-based heuristics [29] for discovering unobserved nodes perform in terms of learning ancestor relations.

References

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] Steven C. Amstrup, Eric T. DeWeaver, David C. Douglas, Bruce G. Marcot, George M. Durner, Cecilia M. Bitz, and David A. Bailey. Greenhouse gas mitigation can reduce sea–ice loss and increase polar bear persistence. *Nature*, 468(7326):955–960, 2010.

[3] Nicos Angelopoulos and James Cussens. Bayesian learning of Bayesian networks with informative priors. *Annals of Mathematics and Artificial Intelligence*, 54:53–98, 2008.

[4] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

[5] Kenneth R. Baker and Linus E. Schrage. Finding an optimal sequence by dynamic programming: An extension to precedence-related tasks. *Operations Research*, 26(1):111–120, 1978.

[6] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of ACM*, 9(1):61–63, 1962.

[7] Jose M. Bernardo. The concept of exchangeability and its applications. *Far East Journal of Mathematical Sciences*, 4:111–121, 1996.

[8] Andreas Björklund and Thore Husfeldt. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica*, 52:226–249, 2008.

[9] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, pages 67–74. ACM, 2007.

[10] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed Moebius inversion and graphs of bounded degree. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 85–96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2008.

[11] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 672–683. Springer, 2006.

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

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

[14] David Maxwell Chickering. Learning Bayesian networks is NP-Complete. In Doug Fisher and Hans-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics*, pages 121–130. Springer-Verlag, 1996.

[15] David Maxwell Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

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

[17] David Maxwell Chickering, David Heckerman, and Chris Meek. Large-sample learning of Bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[18] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

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

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, second edition, 2001.

[21] 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.

[22] Sanjoy Dasgupta. Learning polytrees. In *Proceedings of the Fifteenth Conference Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 134–141. Morgan Kaufmann, 1999.

[23] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 2003.

[24] Cassio P. de Campos and Qiang Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

[25] Cassio P. de Campos, Zhi Zeng, and Qiang Ji. Structure learning of Bayesian networks using constraints. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 113–120. Omnipress, 2009.

[26] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

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

[28] Gal Elidan and Nir Friedman. Learning the dimensionality of hidden variables. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 144–151. Morgan Kaufmann, 2001.

[29] Gal Elidan and Nir Friedman. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, 6:81–127, 2005.

[30] Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

[31] Gal Elidan, Noam Lotner, Nir Friedman, and Daphne Koller. Discovering hidden variables: A structure-based approach. In *Advances in Neural Information Processing Systems 13 (NIPS) 2000*, pages 479–485. MIT Press, 2001.

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

[33] Kobra Etminani, Mahmoud Naghibzadeh, and Amir Reza Razavi. Globally optimal structure learning of Bayesian networks from data. In *Artificial Neural Networks - ICANN*, number 6352 in LNCS, pages 101–106. Springer, 2010.

[34] Jörg Flum and Martin Grohe. *Parametrized Complexity Theory.* Springer, 2006.

[35] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer, 2010.

[36] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, number 5125 in LNCS, pages 210–221. Springer, 2008.

[37] Andrew Frank and Arthur Asuncion. UCI machine learning repository, 2010.

[38] Nir Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, pages 125–133. Morgan Kaufmann, 1997.

[39] 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.

[40] Nir Friedman, Michael Linial, Iftach Nachman, and Dana Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3-4):601–620, 2000.

[41] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, 1979.

[42] Fanica Gavril. Some NP-Complete problems on graphs. In *Proceedings of the 11th Conference on Information Sciences and Systems (CISS)*, pages 91–95, 1977.

[43] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis.* Chapman & Hall/CRC, 2004.

[44] Walter R. Gilks, Sylvia Richardson, and David J. Spiegelhalter, editors. *Markov chain Monte Carlo in Practice.* Chapman & Hall/CRC, 1996.

[45] 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.

[46] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM Journal of Computation*, 16(3):486–502, 1987.

[47] 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.

[48] David Heckerman, Chris Meek, and Gregory F. Cooper. A Bayesian approach to causal discovery. In Clark Glymour and Gregory F. Cooper, editors, *Computation, Causation and Discovery*, pages 141–166. AAAI Press, 1999.

[49] David E. Heckerman, Eric J. Horvitz, and Bharat N. Nathwani. Toward normative expert systems: Part I the Pathfinder project. *Methods of Information in Medicine*, 31:90–105, 1992.

[50] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[51] Inari Helle, Tiina Lecklin, Ari Jolma, and Sakari Kuikka. Modeling the effectiveness of oil combating from an ecological perspective. *Journal of Hazardous Materials*, 185(1):182–192, 2011.

[52] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–417, 1999.

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

[54] 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)*, JMLR: W&CP, pages 358–365, 2010.

[55] Eun Yong Kang, Ilya Shpitser, and Eleazar Eskin. Respecting Markov equivalence in computing posterior probabilities of causal graphical features. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, pages 1175–1180. AAAI Press, 2010.

[56] Richard W. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[57] Robert Kennes. Computational aspects of the Möbius transformation of graphs. *IEEE Transaction on Systems, Man, and Cybernetics*, 22(2):201–223, 1992.

[58] Robert Kennes and Philippe Smets. Computational aspects of the Möbius transformation. In *Uncertainty in Artificial Intelligence 6 (UAI)*, pages 401–416. Elsevier, 1991.

[59] Donald E. Knuth. *The Art of Computer Programming*, volume 2 : Seminumerical algorithms. Addison-Wesley, third edition, 1997.

[60] Donald E. Knuth. *The Art of Computer Programming*, volume 1 : Fundamental algorithms. Addison-Wesley, third edition, 1997.

[61] Mikko Koivisto. *Sum-Product Algorithms for the Analysis of Genetic Risks*. PhD thesis, University of Helsinki, 2004.

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

[63] Mikko Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 583–590. IEEE Computer Society, 2006.

[64] Mikko Koivisto and Pekka Parviainen. A space–time tradeoff for permutation problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 484–492. SIAM, 2010.

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

[66] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models.* MIT Press, 2009.

[67] Johan H. P. Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pages 237–242. IOS Press, 2010.

[68] Pedro Larrañaga, Mikel Poza, Yosu Yurramendi, Roberto H. Murga, and Cindy M. H. Kuijpers. Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):912–926, 1996.

[69] Eugene L. Lawler. A comment on minimum feedback arc sets. *IEEE Transactions on Circuit Theory*, pages 296–297, 1964.

[70] Eugene L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[71] Eugene L. Lawler. Efficient implementation of dynamic programming algorithms for sequencing problems. Technical Report BW 106/79, Stiching Matematisch Centrum, Amsterdam, 1979.

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

[73] Brandon Malone, Changhe Yuan, Eric A. Hansen, and Susan Bridges. Improving the scalability of optimal Bayesian network learning with external-memory frontier breadth-first branch and bound search. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 479–488. AUAI Press, 2011.

[74] Chris Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15:383–389, 2001.

[75] Richard E. Neapolitan. *Learning Bayesian Networks.* Prentice Hall, 2004.

[76] Jesper Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 713–725. Springer, 2009.

[77] 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 557–564. AUAI Press, 2011.

[78] Sascha Ott, Seiya Imoto, and Saturo Miyano. Finding optimal models for small gene networks. In *Pacific Symposium on Biocomputing*, pages 557–567. World Scientific, 2004.

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

[80] 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.

[81] 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 Statistics (AISTATS)*, JMLR: W&CP, pages 589–596, 2010.

[82] Pekka Parviainen and Mikko Koivisto. Ancestor relations in the presence of unobserved variables. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 581–596. Springer, 2011.

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

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

[85] Judea Pearl and Thomas Verma. A theory of inferred causation. In *Proceedings of the 2nd International Conference on Principles of*

Knowledge Representation and Reasoning (KR), pages 441–452. Morgan Kauffman, 1991.

[86] Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal Bayesian network given a super-structure. *Journal of Machine Learning Research*, 9:2251–2286, 2008.

[87] Thomas Richardson. A discovery algorithm for directed cyclic graphs. In *Proceedings of the 12th Conference of Uncertainty in Artificial Intelligence (UAI)*, pages 454–461. Morgan Kaufmann, 1996.

[88] Jorma Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[89] Robert W. Robinson. Counting labeled acyclic digraphs. In *New Directions in the Theory of Graphs: Proceedings of the Third Ann Arbor Conference on Graph Theory, 1971*, pages 239–273. Academic Press, 1973.

[90] Gian-Carlo Rota. On the foundations of combinatorial theory. I. Theory of Möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 2:340–368, 1964.

[91] Walter J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[92] Linus Schrage and Kenneth R. Baker. Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26:444–449, 1978.

[93] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[94] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, and Antti Kerminen. A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7:2003–2030, 2006.

[95] Tomi Silander. *The Most Probable Bayesian Network and Beyond.* PhD thesis, University of Helsinki, 2009.

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

[97] Klaus Simon. Finding a minimal transitive reduction in a strongly connected digraph within linear time. In *Proceedings of the 15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'89)*, pages 245–259. Springer, 1990.

[98] Ajit P. Singh and Andrew W. Moore. Finding optimal Bayesian networks by dynamic programming. Technical Report CMU-CALD-05-106, Carnegie Mellon University, June 2005.

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

[100] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search.* Springer Verlag, 2000.

[101] Peter Spirtes, Chris Meek, and Thomas Richardson. Causal inference in the presence of latent variables and selection bias. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 499–506. Morgan Kaufmann, 1995.

[102] George Steiner. On the complexity of dynamic programming with precedence constraints. *Annals of Operations Research*, 26:103–123, 1990.

[103] Yoshinori Tamada, Seiya Imoto, and Satoru Miyano. Parallel algorithm for learning optimal Bayesian network structure. *Journal of Machine Learning Research*, 12:2437–2459, 2011.

[104] Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the Twenty-First Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 584–590. AUAI Press, 2005.

[105] 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.

[106] Jin Tian, Ru He, and Lavanya Ram. Bayesian model averaging using the k-best Bayesian network structures. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 589–597. AUAI Press, 2010.

[107] 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.

[108] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 455–464. ACM Press, 2009.

[109] Thomas S. Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 255–270. Elsevier, 1990.

[110] Sewall Wright. Correlation and causation. *Journal of Agricultural Research*, 20(7):557–585, 1921.

[111] Frank Yates. *The Design and Analysis of Factorial Experiments.* Number 35 in Technical Communication. Harpenden: Imperial Bureau of Soil Science, 1937.

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

Appendix A

Elementary Mathematical Facts

In this appendix, we present well-known mathematical facts that are used throughout the thesis.

Binomial Theorem

The binomial theorem whose discovery is attributed to Isaac Newton [60, p. 57–58] states that

$$\sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} = (x+y)^n$$

for $n \geq 0$.

Especially, setting $x = 2$ and $y = 1$, yields

$$\sum_{i=0}^{n} \binom{n}{i} 2^i = 3^n. \tag{A.1}$$

Upper Bounds for Binomial Coefficients

Consider the central binomial coefficient $\binom{2n}{n}$ for $n \geq 1$. We have

$$\begin{aligned}
\binom{2n}{n} &= \frac{2n(2n-1)\cdots 2\cdot 1}{n(n-1)\cdots 2\cdot 1\cdot n(n-1)\cdots 2\cdot 1} \\
&= 2^n \frac{(2n-1)(2n-3)\cdots 3\cdot 1}{n(n-1)\cdots 2\cdot 1} \\
&= 2^{2n} \frac{(2n-1)(2n-3)\cdots 3\cdot 1}{2n(2n-2)\cdots 4\cdot 2} \\
&\leq 2^{2n-1}.
\end{aligned} \tag{A.2}$$

The binary entropy function is defined as

$$H(p) = -\big(p \log p + (1-p)\log(1-p)\big),$$

if $0 < p < 1$ and $H(p) = 0$ if $p = 0$ or $p = 1$; see Figure A.1. It holds (see, for example, Cormen et al. [20, p.1097–1098]) that

$$\binom{n}{pn} \leq 2^{H(p)n}.$$

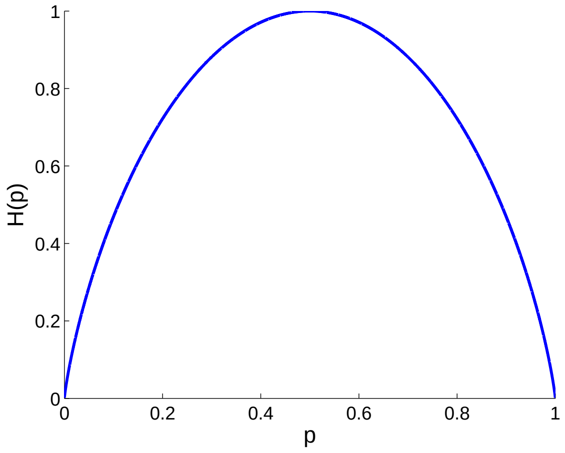


Figure A.1: The binary entropy function.

For $p \leq 1/2$ actually a stronger result holds. It is well-known (for a proof, we refer the reader to Flum and Grohe [34, p. 427]) that

$$\sum_{i=0}^{pn} \binom{n}{i} \leq 2^{H(p)n}. \tag{A.3}$$

A-2008-1 I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)

A-2008-2 J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).

A-2008-3 N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)

A-2008-4 J. Korhonen: IP Mobility in Wireless Operator Networks. (Ph.D. Thesis)

A-2008-5 J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)

A-2009-1 K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)

A-2009-2 T. Silander: The Most Probable Bayesian Network and Beyond. (Ph.D. Thesis)

A-2009-3 K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)

A-2009-4 P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)

A-2009-5 J. Suomela: Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs. 106+96 pp. (Ph.D. Thesis)

A-2009-6 U. Köster: A Probabilistic Approach to the Primary Visual Cortex. 168 pp. (Ph.D. Thesis)

A-2009-7 P. Nurmi: Identifying Meaningful Places. 83 pp. (Ph.D. Thesis)

A-2009-8 J. Makkonen: Semantic Classes in Topic Detection and Tracking. 155 pp. (Ph.D. Thesis)

A-2009-9 P. Rastas: Computational Techniques for Haplotype Inference and for Local Alignment Significance. 64+50 pp. (Ph.D. Thesis)

A-2009-10 T. Mononen: Computing the Stochastic Complexity of Simple Probabilistic Graphical Models. 60+46 pp. (Ph.D. Thesis)

A-2009-11 P. Kontkanen: Computationally Effcient Methods for MDL-Optimal Density Estimation and Data Clustering. 75+64 pp. (Ph.D. Thesis)

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)

A-2010-6 L. Daniel: Cross-layer Assisted TCP Algorithms for Vertical Handoff. 84+72 pp. (Ph.D. Thesis)

A-2011-1 A. Tripathi: Data Fusion and Matching by Maximizing Statistical Dependencies. 89+109 pp. (Ph.D. Thesis)

A-2011-2 E. Junttila: Patterns in Permuted Binary Matrices. 155 pp. (Ph.D. Thesis)

A-2011-3 P. Hintsanen: Simulation and Graph Mining Tools for Improving Gene Mapping Efficiency. 136 pp. (Ph.D. Thesis)

A-2011-4 M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. 104+90 pp. (Ph.D. Thesis)