UNIVERSITY OF CALIFORNIA

Los Angeles

Learning Bayesian Network Structures with Non-Decomposable Scores

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Yuh-Jie Chen

2016

ABSTRACT OF THE DISSERTATION

Learning Bayesian Network Structures with Non-Decomposable Scores

by

Yuh-Jie Chen

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2016

Professor Adnan Youssef Darwiche, Chair

Bayesian networks learned from data and background knowledge have been broadly used to reason under uncertainty, and to determine associations and dependencies between random variables in the data, in various fields such as artificial intelligence, machine learning, and bioinformatics. The problem of learning the structure of a Bayesian network is typically formulated as an optimization problem, by scoring each network structure with respect to data and background knowledge. Modern approaches to the structure learning problem assume the scores to be *decomposable*, so that the optimization problem can be decomposed into a number of smaller and easier subproblems that can be optimized independently. These approaches include those based on dynamic programming, heuristic search, and integer linear programming methods.

In this thesis, we break away from this tradition, and consider non-decomposable scores, that provide a richer expression for scoring the network structures. More specifically, we generalize the heuristic search approach for learning with decomposable scores, by using a more expressive search space, that accommodates non-decomposable scores. The search can be guided effectively by a heuristic function evaluated by existing structure learning approaches for decomposable scores. We show that our framework can learn an optimal Bayesian network structure with ancestral constraints and order-modular priors. Both are non-decomposable scores. Our framework can also

enumerate the k-best Bayesian network structures and the k-best Markov-equivalent Bayesian network structures, using decomposable scores, and is empirically orders of magnitude more efficient than the previous state of the art.

The dissertation of Yuh-Jie Chen is approved.

Richard E. Korf

Wei Wang

Qing Zhou

Adnan Youssef Darwiche, Committee Chair

University of California, Los Angeles

2016

To my parents ...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to express my sincere gratitude to my advisor, Adnan Darwiche, for his guidance and support in my journey through graduate school. I much appreciated the freedom he gave me to pursue the research topics that I am interested in, and the patience he showed during the process. I learned from Adnan how to choose a good research topic, and to present our ideas and work.

Second, I would like to thank Arthur Choi, with whom I worked closely on the research problems that center this thesis. Working with Arthur on a day-to-day basis is a very valuable experience, which made me grow as a researcher and an engineer.

Next, I would like to thank my committee members Richard Korf, Wei Wang, and Qing Zhou for their encouragement and invaluable comments and feedback. I would like to especially thank Rich, for the topics covered in his heuristic search class is an essential part of this thesis. I would also like to thank the researchers that we have consulted, for their generous comments and help: Mark Bartlett, Joseph Barker, Zhaoxing Bu, Mark Chavira, Yetian Chen, James Cussens, Cassio de Campos, Ru He, Mikko Koivisto, Brandon Malone, Ethan Schreiber, and Guy Van den Broeck.

Furthermore, I would like to thank Judea Pearl, Fan Chung Graham and Ronald Graham for advising me during my graduate studies; and Yung-Yu Chuang and Tsan-sheng Hsu for advising me during my undergraduate years. I would also like to thank the members in my labs, whom I have been very fortunate to know and learn from. I would like to especially thank Jason (Yujia) for our collaboration on one central research problem considered in this thesis.

Last but not least, I would like to thank my family and friends for their endless love and support.

VITA

2012-2016	Teaching Assistant, University of California, Los Angeles.
2010-2013	M.S. Computer Science, University of California, Los Angeles.
2010-2016	Research Assistant, University of California, Los Angeles.
2010	Software Engineering Intern, Google, Mountain View.
2009-2010	Ph.D. student, Computer Science and Engineering, University of California, San Diego.
2004-2009	B.S. Computer Science and Information Engineering, National Taiwan University, Taiwan.

PUBLICATIONS

Eunice Yuh-Jie Chen, Arthur Choi, and Adnan Darwiche. "Enumerating Bayesian Networks Using the MDL Score." Under review.

Eunice Yuh-Jie Chen, Yujia Shen, Arthur Choi, and Adnan Darwiche. "Learning Bayesian Networks with Ancestral Constraints." Under review.

Eunice Yuh-Jie Chen, Arthur Choi, and Adnan Darwiche. "Enumerating Equivalence Classes of Bayesian Networks using EC Graphs." In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 591-599, 2016.

Eunice Yuh-Jie Chen, Arthur Choi, and Adnan Darwiche. "Learning Bayesian Networks with Non-Decomposable Scores." In *the 4th International Workshop on Graph Structures for Knowledge Representation and Reasoning, Lecture Notes in Artificial Intelligence*, pages 50-71, 2015.

Eunice Yuh-Jie Chen, and Judea Pearl. "Random Bayesian Network with Bounded In-Degree." In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, pages 114-121, 2014.

Eunice Yuh-Jie Chen, and Judea Pearl. "A Simple Criterion for Controlling Selection Bias." In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, page 170-177, 2013.

CHAPTER 1

Introduction

In this chapter, we introduce Bayesian networks and the problem of Bayesian network structure learning, and provide an outline of the remainder of the thesis.

1.1 Probability Distributions and Bayesian Networks

The world is full of uncertainty. We may wake up to different weather, and on our way to work, we may experience different traffic conditions. As a consequence, representing and understanding uncertainty, such as with probability theory, has received much attention.

More specifically, a probability distribution over n random variables has an exponential number of possible outcomes, e.g., 10 binary variables have 2^{10} possible results, and it is therefore challenging to record the probability of each of these outcomes to represent the probability distribution. More importantly, a probability distribution, by itself, is often not what one is truly interested in. One wants, instead, to learn the relationship between each random variable, to understand these events, and in particular, how they interact. For example, consider four binary random variables: Fall (F), Rain (R), Sprinkler (S), and Wet Grass (W), and the probability distribution in Figure 1.1. The probability of whether there is rain and whether it is fall are dependent, so the events of rain and season are related, or *dependent*. Moreover, the probability of whether there is rain and whether the automatic sprinkler is on are independent, given whether it is fall, so the events of rain and sprinkler do not depend on each other, given

F	R	S	W	$\Pr(f, r, s, w)$	F	R	S	W	$\Pr(f, r, s, w)$
t	t	t	t	0.0570	f	t	t	t	0.0570
t	t	t	f	0.0030	f	t	t	f	0.0030
t	t	f	t	0.1120	f	t	f	t	0.0120
t	t	f	f	0.0280	f	t	f	f	0.0030
t	f	t	t	0.0105	f	f	t	t	0.3780
t	f	t	f	0.0045	f	f	t	f	0.1620
t	f	f	t	0.0035	f	f	f	t	0.0135
t	f	f	f	0.0315	f	f	f	f	0.1215

Figure 1.1: A probability distribution over random variables F, R, S, and W.

the season.

The dependence and independence relations in a probability distribution can be represented graphically, by using a directed acyclic graph (DAG), known as a *Bayesian network structure* [Pea88, Dar09, KF09, Mur12], which in turn gives an efficient representation of the probability distribution. In a Bayesian network structure, each node represents a random variable, and each edge expresses a direct dependence of the random variables. For example, the relations of the random variables in the probability distribution in Figure 1.1 can be represented by the Bayesian network structure in Figure 1.2, where whether there is rain and whether the sprinkler is on directly depend on the season, and whether the grass is wet directly depends on rain and the sprinkler.

1.2 Bayesian Network Structure Learning

In this section, we consider learning a Bayesian network structure, i.e., learning the dependence and independence relations of the random variables.

First, we consider learning a network structure from an *empirical* probability of the

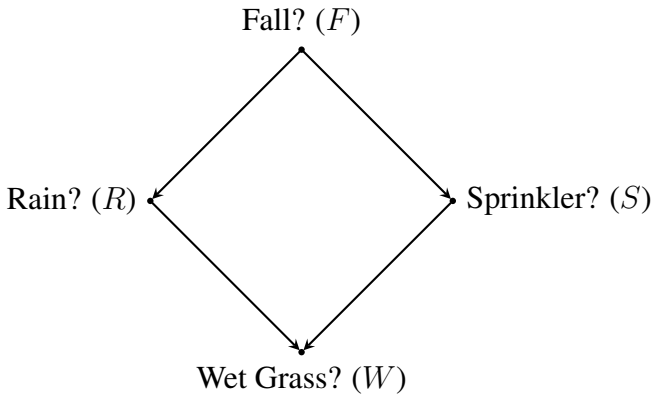


Figure 1.2: A Bayesian network structure representing the probability distribution in Figure 1.1.

F	R	S	W
t	f	t	t
t	f	t	t
t	f	f	t
f	t	f	f
t	f	t	t
t	f	t	f
f	f	t	t
t	t	f	f
f	t	f	f
t	f	t	t

Figure 1.3: A complete dataset over random variables F, R, S, and W, where for each data point, the outcomes of all four random variables F, R, S and W are observed.

random variables from a *complete* dataset. A dataset is complete if for every data point inside the dataset, the outcomes of all the random variables are observed. For example, Figure 1.3 shows a complete dataset. An empirical probability of a complete dataset is the ratio of the number of outcomes in the dataset. For example, in dataset in Figure 1.3, the empirical probability of rain is 0.3, since 3 of the 10 observations have rain; and the empirical probability that the sprinkler is on is 0.6, since 6 of the 10 observations have the sprinkler on.

We note that learning directly from the true underlying probability distribution of the random variables, which governs the relationship between these variables, is usually not possible, as one rarely has access to it. One, however, may obtain the empirical probability by observing the events. For instance, consider the example in Section 1.1. One cannot know that given rain and the sprinkler is on, the probability that the grass is wet is exactly 0.95. One can instead observe these three events, to obtain the empirical probability from the outcomes seen, such as in Figure 1.3. In general, when more observations are made, the empirical probability resembles more closely the true probability distribution; otherwise, the empirical probability may differ significantly from the true probability distribution, and as a result, a Bayesian network structure learned from the empirical probability does not express the true dependence and independence relations of the random variables [Bun91, CH92, Hec98].

When the number of observations in the dataset is small, one might want to improve the network structure learned by incorporating *background knowledge* of the structure, if available [Bun91, CH92, Hec98]. For instance, in the example in Section 1.1, if one knows for certain that rain directly depends on the season, then this knowledge may be used to ensure that the learned network has an edge from rain to fall. We therefore consider a more general framework of Bayesian network structure learning, where we use *both* the data and background knowledge in the learning.

The Bayesian network structure learning problem is typically formulated as an optimization problem, where each Bayesian network structure is associated with a *score*

with respect to the dataset and background knowledge, to reflect how well the network structure represents the empirical probability and the knowledge. A network structure with the best score, found through solving the optimization problem, is viewed as the best model for the relations between the random variables [Dar09, KF09, Mur12].

To make the optimization problem more manageable, a certain property, called *decomposability*, is usually assumed by the score. Score decomposability allows the optimization problem, which considers the entire network structure, to be simplified into a number of smaller and easier optimization problems over sub-structures, that can be optimized independently. These optimization problems can be solved exactly by algorithms based on dynamic programming, heuristic search, and integer linear programming, and scale to problems with dozens of variables [KS04, SM06, JSG10, Cus11, YMW11, YM13, FYM14]. A wide range of approximate algorithms have also been applied to optimize these problems, to scale up to thousands of variables [KF09, Mur12].

The score decomposability assumption, while significantly advancing structure learning in the past two decades, limits how well the structures capture the relations between the random variables, with respect to data and background knowledge. Hence, it is not surprising that there are scores that one would like to use, yet are non-decomposable. For instance, in the example in Section 1.1, if one knows that whether the grass is wet depends, directly or indirectly, on the season, then the learned Bayesian network structure should have a path from fall to wet grass. This background knowledge is non-decomposable, as we shall discuss in Chapter 6.

1.3 A New Search-Based Approach for Structure Learning with Non-Decomposable Scores

In this thesis, we propose a general search-based framework for learning Bayesian network structures with non-decomposable scores. Our framework generalizes the search-based approach for structure learning with decomposable scores from [YMW11,

YM13], by using a more expressive search space, called the *BN graph*, that accommodates non-decomposable scores.

The BN graph, compared to search spaces for learning with decomposable scores, has a much larger size. We show that the BN graph can be explored efficiently, to find an optimal Bayesian network structure, by using a tight heuristic function that guides a search nearly directly towards the optimal network structure, so that only a small portion of the search space is traversed. In particular, this heuristic function is evaluated by existing structure learning approaches for decomposable scores, such as [KS04, SM06, JSG10, Cus11, YMW11, YM13]. As a result, we expand the reach of these approaches to new learning tasks that are harder, and beyond those that they were originally designed for.

Using our new framework, we consider the problem of learning an optimal Bayesian network structure with two non-decomposable scores: those integrating ancestral constraints and order-modular priors. The ancestral constraints, as mentioned in Section 1.2, specify dependence, direct and indirect, between random variables. Order-modular priors provide a probability distribution of network structures, and is used extensively for sampling network structures, such as in Markov chain Monte Carlo (MCMC) methods for Bayesian model averaging [KF09, Mur12].

In addition, we also consider the problems of (1) enumerating the k-best Bayesian network structures, where instead of finding the network structure with the minimum score, we enumerate the k structures with the smallest scores, and (2) enumerating the k-best Markov-equivalent Bayesian network structures, i.e., network structures that encode the same set of dependence and independence constraints, and are therefore equally expressive in terms of representing probability distributions.

These problems can be of interest when the the dataset is small. In this case, as discussed in Section 1.2, the empirical probability differs from the true probability distribution, and the optimal Bayesian network structure may not represent the true dependence between random variables. Consequently, one might want to be aware of

other possible Bayesian network structures, by enumerating the k-best ones. We show empirically that our approach can be orders of magnitude faster than the state-of-the-art systems over real datasets, in enumerating the k-best network structures.

1.4 Overview

Below is an overview of each chapter.

In Chapter 2 we provide a formal definition of a Bayesian network, and review previous work on Bayesian network structure learning.

In Chapter 3 we propose our framework for learning optimal Bayesian network structures with non-decomposable scores, that generalizes the search-based approach for decomposable scores by using a more expressive search space called the *BN graph*. We discuss how to explore the BN graph efficiently with the A* search, by using a heuristic function evaluated by existing structure learning systems for decomposable scores. Moreover, we use the BN graph to enumerate the k-best Bayesian network structures using decomposable scores, and show empirically that our proposed approach is orders of magnitude more efficient than the existing state-of-the-art, which is based on dynamic programming.

In Chapter 4 we consider using the BN graph to learn an optimal Bayesian network structure with the order-modular priors. More specifically, the problem of evaluating order-modular prior by itself is hard. We show that the order-modular prior can be evaluated while performing the A* search, with a slight variation, on BN graph; and an optimal network structure can be learned accordingly.

In Chapter 5 we consider specializing the BN graph to a more compact search space, called the *EC graph*, over Markov-equivalent Bayesian network structures, to enumerate the k-best Markov-equivalent structures. The EC graph is further canonicalized into another search space, call the *EC tree*, so that each node in the search space is reached by exactly one path. Empirically, we show that our proposed approach is in practice

orders of magnitude more efficient than the existing state-of-the-art, which is based on dynamic programming.

In Chapter 6 we consider using the BN graph to learn an optimal Bayesian network structure with ancestral constraints. Moreover, we show that certain decomposable constraints can be inferred from the ancestral constraints, and can be exploited to tighten the heuristic function. Empirically, our approach is orders of magnitude more efficient than the state-of-the-art, which is based in integer linear programming.

In Chapter 7 we consider large random Bayesian network structures generated by the uniform order-modular prior, but where each node has relatively few parents. In other words, DAGs where the edges appear at random, each node has at most k parents, and k is much less than the number of nodes n. We analyze the asymptotic expected properties of these random Bayesian network structures, such as the expected number of parents a node in the network structure has, and the expected size of the Markov blanket.

Finally, we summarize the contribution of this thesis in Chapter 8.

CHAPTER 2

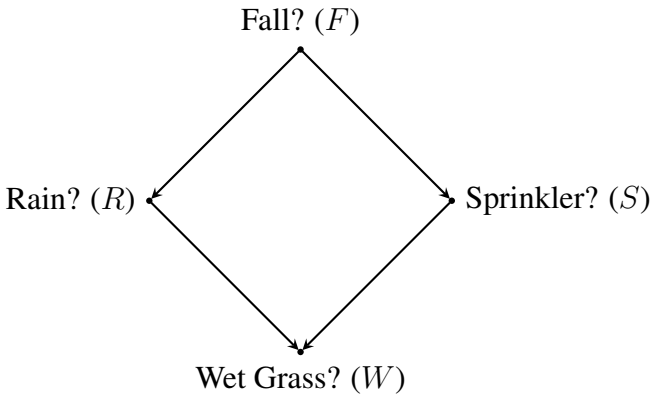
Bayesian Networks and Previous Work

In this chapter, we provide an introduction to Bayesian networks, and review the previous work on Bayesian network structure learning, where the main approaches are the *score-based* approach and the *constraint-based* approach.

2.1 Bayesian Networks

A Bayesian network consists of the *network structure*, i.e., a DAG, where each node represents a random variable, and the *parameterization*, i.e., conditional probability for each random variable given its parents in the DAG. For example, Figure 2.1 shows a Bayesian network that represents the probability distribution in Figure 1.1. As discussed in Section 1.1, the network structure expresses random variable dependence and independence in the probability distribution it represents. In Figure 2.1, the network structure suggests that whether the grass is wet directly depends on rain and the sprinkler; and by the parameterization, given say there is rain and the sprinkler is on, the probability that the grass is wet is 0.95. Moreover, the probability that the season is fall is $\Pr(F = \text{true}) = 0.25$; and the probability to rain in fall is $\Pr(R = \text{true}|F = \text{true}) = 0.8$; so the joint probability that the season is fall and it is raining is $\Pr(F = \text{true}, R = \text{true}) = \Pr(F = \text{true}) \Pr(R = \text{true}|F = \text{true}) = 0.25 \cdot 0.8$. Similarly, the joint probability for all four variables to be true is $\Pr(F = \text{true}, R = \text{true}, S = \text{true}, W = \text{true}) = \Pr(F = \text{true}) \Pr(R = \text{true}|F = \text{true}) \Pr(S = \text{true}|F = \text{true}) \Pr(W = \text{true}|R = \text{true}, S = \text{true}) = 0.25 \cdot 0.8 \cdot 0.3 \cdot 0.95$. More generally, the probability of a

full variable instantiation is obtained by multiplying a number of network parameters, one from each conditional probability table. The parameter included from each table must be compatible with the variable instantiation.



(a) Network structure.

F	$\Pr(f)$
t	0.25
f	0.75

F	R	$\Pr(r\|f)$
t	t	0.8
t	f	0.2
f	t	0.1
f	f	0.9

F	S	$\Pr(s\|f)$
t	t	0.3
t	f	0.7
f	t	0.8
f	f	0.2

R	S	W	$\Pr(w\|s,r)$
t	t	t	0.95
t	t	f	0.05
t	f	t	0.8
t	f	f	0.2
f	t	t	0.7
f	t	f	0.3
f	f	t	0.1
f	f	f	0.9

(b) Parameterization for node F, R, S and W, where t denotes true, and f denotes false.

Figure 2.1: A Bayesian network.

For the remainder of the thesis, we use upper case letters X to denote single variables and bold-face upper case letters $\mathbf{X}$ to denote sets of variables, and we will use lower case letters x to denote the instantiation of variable X and bold-face lower case letters $\mathbf{x}$ to denote the instantiation of sets of variables $\mathbf{X}$. Generally, we will use X to denote a variable in a Bayesian network and $\mathbf{U}$ to denote its parents, and refer to a node

and its parents $X\mathbf{U}$ as a *family*.

2.2 Scoring Bayesian Network Structures

In this section, we discuss how to score Bayesian network structures, which the structure learning problem exploits to formulate the learning problem as an optimization problem, i.e., finding a DAG with the minimum score.

Given a *complete* dataset $\mathcal{D}$, where the value of every random variable is observed, and background knowledge $\mathcal{P}$, the score of a Bayesian network structure G is denoted as $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$, such that the smaller the score, the better the Bayesian network structure represents the data $\mathcal{D}$ and background knowledge $\mathcal{P}$. A score is *decomposable* when the score of a Bayesian network structure G can be rewritten as the sum of the scores of the families in G:

$$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P}) = \sum_{X\mathbf{U}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}, \mathcal{P}), \tag{2.1}$$

As mentioned in Section 1.2, most existing literature on structure learning assumes score decomposability, as it significantly simplifies the learning problem, which we shall see later in this chapter. Below, we first consider scores that do not integrate background knowledge, then we consider some types of background knowledge that has been incorporated into such scores.

2.2.1 Scores with no Background Knowledge

In this section, we review two of the most commonly used scores, which are decomposable, that consider how well a Bayesian network structure represents the data $\mathcal{D}$ and the complexity of the network structure: the minimum description length (MDL) scores and the Bayesian Dirichlet likelihood equivalence uniform (BDeu) scores [KF09, Mur12].

The MDL score considers the likelihood of network structures G given data $\mathcal{D}$; and at the same time exploits the Occam's Razor principle, favoring structures G that

provide a simpler encoding of the data $\mathcal{D}$. Let N denote the dataset size, $\mathbf{X}^{\#}$ denote the number of instantiations for variables $\mathbf{X}$, and $\mathcal{D}\#(\mathbf{x})$ denote the number of cases in the dataset $\mathcal{D}$ that satisfies instantiation $\mathbf{x}$. The MDL score is defined as

$$\mathsf{MDL}(G \mid \mathcal{D}) = \mathsf{LL}(G \mid \mathcal{D}) - \big(\frac{\log_2 N}{2}\big)||G||, \tag{2.2}$$

where $\mathsf{LL}(G \mid \mathcal{D})$ is the log-likelihood of network structure G given data $\mathcal{D}$, defined as

$$\begin{aligned}\mathsf{LL}(G \mid \mathcal{D}) &= -\sum_{X\mathbf{U}} H(X|\mathbf{U}) \\ H(X|\mathbf{U}) &= -\sum_{x\mathbf{u}} \mathcal{D}\#(x\mathbf{u}) \log_2 \frac{\mathcal{D}\#(x\mathbf{u})}{\mathcal{D}\#(\mathbf{u})}.\end{aligned}$$

Here, $H(X|\mathbf{U})$ is the entropy of variable X given parents $\mathbf{U}$; and $||G||$ measures the complexity of the encoding, and is defined as

$$||G|| = \sum_{X\mathbf{U}} (X^{\#} - 1)\mathbf{U}^{\#}.$$

The BDeu score favors more probable structures G given data $\mathcal{D}$. That is,

$$\begin{aligned}\mathsf{BDeu}(G \mid \mathcal{D}) &\propto \log Pr(G \mid \mathcal{D}) \\ &\propto \log Pr(\mathcal{D} \mid G) Pr(G).\end{aligned} \tag{2.3}$$

The BDeu score further assumes a uniform prior on the structures G, and consequently,

$$\mathsf{BDeu}(G \mid \mathcal{D}) \propto \log Pr(\mathcal{D} \mid G). \tag{2.4}$$

The BDeu scores are defined as

$$\mathsf{BDeu}(G \mid \mathcal{D}) = \sum_{X\mathbf{U}} \log \sum_{\mathbf{u}} \frac{\Gamma(\alpha_{X|\mathbf{u}})}{\Gamma\left(\alpha_{X|\mathbf{u}} + \mathcal{D}\#\left(\mathbf{u}\right)\right)} \sum_{x} \frac{\Gamma\left(\alpha_{x|\mathbf{u}} + \mathcal{D}\#\left(x\mathbf{u}\right)\right)}{\Gamma(\alpha_{x|\mathbf{u}})}, \tag{2.5}$$

where $\alpha_{X|\mathbf{u}}$ is $\alpha/\mathbf{U}^{\#}$ and $\alpha_{x|\mathbf{u}}$ is $\alpha/X^{\#}\mathbf{U}^{\#}$, and α is the parameter *equivalent sample size*.

Finally, we note that to formulate the structure learning as a minimization problem, i.e., finding a best DAG with the minimum score, the BDeu and MDL scores need to be negated.

2.2.2 Scores with Background Knowledge

In this section, we review background knowledge on Bayesian network structures that have been combined with MDL and BDeu scores.

We first consider background knowledge that is decomposable, i.e., background knowledge that can be expressed over families. One commonly used form of background knowledge is the maximum number of parents constraint [CL68, CH92]. Here, each variable is assumed to have at most k parents. In this case, a family $X\mathbf{U}$ that violates the constraint, i.e., the number of parents $|\mathbf{U}|$ is greater than k, has a $\mathsf{score}(X\mathbf{U} \mid \mathcal{D}, \mathcal{P})$ of infinity.

Another popular type of background knowledge is the ordering constraint, i.e., a topological ordering of the variables that the learned network structure has to be compatible with [CH92], that is a topological ordering that is consistent with the DAG of a network structure. These constraints may be available when one knows the temporal order of variables. Ordering constraints can also be incorporated into scores by assigning families that violate the constraints an infinite score.

One other type of background knowledge that has been considered widely is the edge existence probability, where one specifies the probability that a certain variable is a parent of another variable [MGR95]. This background knowledge may be available when one understands the direct dependence between variables and can provide such probabilities. The edge existence knowledge provides a probability for each family $X\mathbf{U}$, and consequently can be used in the score of the family.

We now consider ancestral constraints, which are a type of non-decompsable background knowledge. These constraints allow one to specify that a variable depends, either directly or indirectly, on another variable,[1] e.g., in diagnosis, one may know that certain diseases are ancestors of certain symptoms [BT12, BT13].

The order-modular prior, a prior that considers a weighted count of the number of

[1]We note that when variable X is a parent of Y, it is also an ancestor of Y.

topological orderings a network structure is compatible with, is also non-decomposable. The order-modular prior greatly facilitates the process of sampling Bayesian network structures, where the network structures with better scores are more likely to be sampled [KF09, Mur12].

2.3 Approximate Score-Based Approaches to Structure Learning

In this section, we review approximate approaches for learning Bayesian network structures.

The problem of learning an optimal Bayesian network structure is NP-hard [CHM04]. As a result, various approximate algorithms have been developed to mitigate the complexity of the learning problem. Many early algorithms in this attempt use search-based algorithms over the space of all possible network structures, i.e., DAGs [LB93, CGH95, Hec98]. More specifically, the search first uses a random network structure as the initial network structure; then generates new network structures with local changes of the current network structure, i.e., edge addition, removal, or reversal; and stops when the search meets some stopping criterion, e.g., the new structures are all worse than the current structure.

The search space over all structures, i.e., DAGs, is super-exponential in the number of variables. As a result, following these early works, search-based algorithms over a variety of more compact search spaces have been proposed. Background knowledge is considered to constrain the search space, e.g., background knowledge on the number of parents and the existence of edges in the network, [FNP99, MW03, PIM08]. Markov-equivalent Bayesian networks, i.e., Bayesian networks that represent the same family of distributions, are also avoided in the search space [CM02, Chi02].

We note that in principle search-based algorithms over the space of all structures, and some of their more compact variations, can accommodate non-decomposable scores,

as the search process only requires the evaluation of $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$, and not its decomposability. However, historically, the scores have been assumed to be decomposable for computational efficiency. More specially, with decomposability, when a new network structure G is generated from the current structure G' by a local change, the score of the new structure, i.e., $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$, can be obtained from the score of the current structure, i.e., $\mathsf{score}(G' \mid \mathcal{D}, \mathcal{P})$, by only updating the scores over families that are involved in the local change, i.e., $\mathsf{score}(X\mathbf{U} \mid \mathcal{D}, \mathcal{P})$.

2.4 Exact Score-Based Approaches to Structure Learning

In this section, we review exact approaches for learning Bayesian network structures.

2.4.1 The K2 Algorithm

The K2 algorithm is one of the first exact algorithms to learn an optimal Bayesian network structure that minimizes the score of the structure, i.e., $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$ [Bun91, CH92]. The K2 algorithm exploits the assumption that an optimal Bayesian network structure G is consistent with a given topological ordering of the variables $\prec$.

By this assumption and score decomposability, an optimal DAG can be constructed by choosing the optimal set of parents for each variable X, from those variables that precede X in the ordering $\prec$. That is,

$$\min_{G \text{ consistent with } \prec} \mathsf{score}(G \mid \mathcal{D}, \mathcal{P}) = \min \sum_{X\mathbf{U}, \text{ where } \mathbf{U} \subset \mathbf{Y}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}, \mathcal{P}), \tag{2.6}$$

where $\mathbf{Y}$ is the set of the variables that precedes X in $\prec$.

We note that with non-decomposable scores, the minimization problem of the score of the structure cannot be decomposed as in Equation 2.6. We will revisit this key issue with a concrete example in Section 3.3.

2.4.2 Order-Graph-Based Approaches

An optimal Bayesian network structure G that minimizes the score of the structure, i.e., $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$, can be found by running the K2 algorithm on all $n!$ possible orderings $\prec$, and then taking a DAG with the smallest score.

This idea can be further improved by the observation that these $n!$ instances share many computational subproblems: finding the optimal set of parents for some variable X from the set $\mathbf{Y}$ of variables that precede X. One can aggregate these common subproblems, allowing one to solve at most $n \cdot 2^{n-1}$ unique subproblems. This technique underlies a number of modern approaches to score-based structure learning, including some based on dynamic programming [KS04, SM05, SM06], and related approaches based on heuristic search methods, such as A* search [YMW11, YM13]; for A* search, see [HNR68]. In the context of A* search, this aggregation of K2 sub-problems corresponds to a search space called the *order graph*.

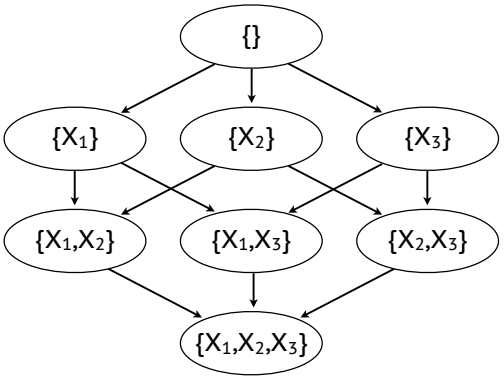


Figure 2.2: The order graph for variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

Figure 2.2 illustrates an order graph over variables $\mathbf{X} = \{X_1, X_2, X_3\}$, where each node represents a subset of these variables. There is a directed edge from set $\mathbf{Y}$ to set $\mathbf{Z}$ if and only if $\mathbf{Z}$ results from adding some variable X to $\mathbf{Y}$. We denote such an edge by $\mathbf{Y} \xrightarrow{X} \mathbf{Z}$. Hence, any path $\{\} \xrightarrow{X_1} \cdots \xrightarrow{X_n} \mathbf{X}$ from the root to the leaf corresponds to a unique ordering $\langle X_1, \ldots, X_n \rangle$ of the variables. We can associate each edge $\mathbf{Y} \xrightarrow{X} \mathbf{Z}$

with the local score: $\min_{\mathbf{U}\subseteq\mathbf{Y}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D})$, which corresponds to finding an optimal set of parents $\mathbf{U}$ of X from a candidate set of parents $\mathbf{Y}$. Hence, each edge of the order graph corresponds to one of the computational subproblems of the K2 algorithm. Moreover, any path from the root node of the order graph to the leaf corresponds to a full instance of the K2 algorithm, when invoked with the corresponding ordering. As a result, a path from the root node to the leaf gives the optimal DAG for the corresponding ordering, the cost of the path is the score of the optimal DAG, and the optimal score over all paths yields an optimal DAG over all orderings, based on Equation 2.1.

2.4.3 Integer-Linear-Programming-Based Approaches

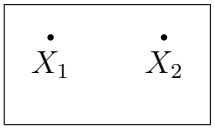
Bayesian network structure learning can also be formulated using integer linear programming (ILP), with Equation 2.1 as the linear objective function of an ILP. In particular, for each variable X and possible parent set $\mathbf{U}$, an ILP variable $I(X, \mathbf{U}) \in \{0, 1\}$ is introduced to represent the event that X has parents $\mathbf{U}$ when $I(X, \mathbf{U}) = 1$, and $I(X, \mathbf{U}) = 0$ otherwise. In addition, a set of constraints asserts that each variable X has a unique set of parents, $\sum_{\mathbf{U}} I(X, \mathbf{U}) = 1$. Another set of constraints ensures that all variables X and their parents $\mathbf{U}$ must yield an acyclic graph. One approach is to use cluster constraints [JSG10], where for each cluster $\mathbf{C} \subseteq \mathbf{X}$, at least one variable X in $\mathbf{C}$ has no parents in $\mathbf{C}$, $\sum_{X\in\mathbf{C}} \sum_{\mathbf{U}\cap\mathbf{C}=\emptyset} I(X, \mathbf{U}) \geq 1$. Finally, the objective function of the ILP is $\sum_{X\in\mathbf{X}} \sum_{\mathbf{U}\subseteq\mathbf{X}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}) \cdot I(X, \mathbf{U})$, which corresponds to Equation 2.1.

In principle, the ILP approach can accommodate non-decomposable scores, as long as the scores can be expressed using a linear cost function [OSM15]. In addition, we note that the A* search approach, discussed in Section 2.4.2, and the ILP approach are effective in different domains. More specifically, the ILP approach is more efficient when the parents of a node in the network are restricted to a relatively small size; otherwise, in the more general case without this restriction, the search approach is more effective [YM13, MKJ14].

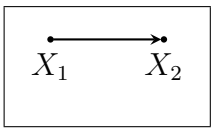
2.5 Constraint-Based Approaches to Structure Learning

In this section, we discuss constraint-based approaches to structure learning, which exploits independence relations between the variables in given data and background knowledge, to learn a family of network structures that is consistent with these dependence constraints.

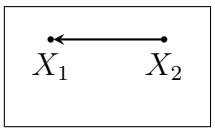
The constraint-based approach assumes *faithfulness*, i.e., there exists a DAG such that the data contains exactly the independencies in the DAG. With this assumption, various graphical rules can be used to infer the Bayesian network structures that satisfy the independencies in data and background knowledge [PV91, Mee95, SGS00]. For example, consider learning a network structure over variables X_1 and X_2, where the two variables are independent in the data. In this case, the only Bayesian network structure that satisfies the independency is DAG



where the two variables do not have dependency through directed edges. If, on the other hand, that X_1 and X_2 are dependent in the data, both DAG



and DAG



can potentially be learned, as the directed edges between X_1 and X_2 express an dependency.

2.6 Other Approaches to Structure Learning

In this section, we review structure learning of *Gaussian Bayesian networks*. Gaussian Bayesian networks are a particular family of Bayesian networks, where the random variables are continuous, and the parameterization over the families has a linear Gaus-

sian form. More specifically, in a Gaussian Bayesian network, the value of each variable X is defined as

$$X = \sum_{U \in \mathbf{U}} \beta_{X,U}\, U + \epsilon_X, \tag{2.7}$$

where $\beta_{X,U}$ is the coefficient captures the influence of U on X; and ϵ_X is independent Gaussian noise variable with mean 0 and variance σ_X, which captures uncertainty in the value of X given the values of its parents $\mathbf{U}$ [Nea04, KF09, Mur12].

For Gaussian Bayesian networks, the structure learning problem can be formulated as an estimation problem of unknown parameters $\beta_{X,U}$ and ϵ_X from data $\mathcal{D}$. The estimation problem considers how well the Bayesian network represents the data, and the complexity of the network structure. For the former, the smaller the mean squared error between the probability distribution induced by the Gaussian Bayesian networks using estimated parameters and the empirical probability from $\mathcal{D}$, the better the estimation of parameters $\beta_{X,U}$ and ϵ_X. For the latter, L_1-norm regularization, i.e., the number of edges, is used to represent model complexity, where models with fewer edges are preferred.[2] A good estimation of the parameters can be found approximately by coordinate descent algorithms [SM10, FZ13, AZ14].

[2] L_1-norm regularization is also known as *lasso* regularization [FHT01].

CHAPTER 3

A New Search Space for Learning Bayesian Network Structures

In this chapter, we consider learning optimal Bayesian networks, with *non-decomposable scores*. We generalize the order-graph approach [YM13], and propose a new search space, called the *Bayesian network graph (BN graph)*, to accommodate non-decomposable scores. Moreover, we discuss how the BN graph can be explored efficiently by A* search —in fact, we propose to use a tight heuristic function that can be evaluated by state-of-the-art learning systems for decomposable scores. Finally, we use the BN graph to enumerate the k-best Bayesian network structures, using decomposable scores, and show empirically that our proposed approach is orders of magnitude more efficient than the existing state-of-the-art. This chapter is based on [CCD15].

3.1 Introduction

Modern approaches for learning Bayesian network structures are typically formulated as a combinatorial optimization problem, where one wants to find the best network structure (i.e., best DAG) that has the lowest score, for some given scoring metric [Dar09, KF09, Mur12]. Typically, one seeks a Bayesian network that explains the data well, without overfitting the data, and ideally, also accommodating any background knowledge that may be available.

Some of the earliest procedures for learning Bayesian network structures used scoring metrics with a certain desirable property, called *score decomposability*. For ex-

ample, consider the K2 algorithm which exploited decomposable scores, in combination with an assumption on the topological ordering of the variables [CH92]. Under these assumptions, the structure learning problem itself decomposes into local subproblems, where we find the optimal set of parents for each variable, independently. Local search methods exploited decomposability in a similar way [CGH95]. Such methods navigated the space of Bayesian network structures, using operators on edges such as addition, deletion, and reversal. Score decomposability allowed these operators to be evaluated locally and efficiently. Indeed, almost all scoring metrics used for Bayesian network structure learning are decomposable. Such scores include the K2 score, [CH92], the BDeu score [Bun91], the BDe score [HGC95], and the MDL score [Bou93, LB94, Suz93], among many others.

Modern approaches to structure learning continue to exploit the decomposable nature of such scoring metrics. In particular, the past decade has seen significant developments in *optimal* Bayesian network structure learning. These recent advances were due in large part to dynamic programming (DP) algorithms, for finding optimal Bayesian network structures [KS04, SM05, SM06]. Subsequently, approaches have been proposed based on heuristic search, such as A* search [YMW11, YM13], as well as approaches based on integer linear programming (ILP), and their relaxations [JSG10, Cus11].

By exploiting the nature of decomposable scores, these advances have significantly increased the scalability of optimal Bayesian network structure learning. There is, however, a notable void in the structure learning landscape due to the relative lack of support for *non-decomposable scores*. This includes a general lack of support for non-decomposable priors, or more broadly, the ability to incorporate more expressive, but non-decomposable forms of prior knowledge (e.g., biases or constraints on ancestral relations). In this chapter, we take a step towards a more general framework for Bayesian network structure learning that targets this void.

The modern approaches for optimal structure learning, mentioned earlier, are based

on a search space called the order graph [KS04, YMW11]. The key property of the order graph is its size, which is only exponential in the number of variables of the Bayesian network that we want to learn. Our proposed framework however is based on navigating the significantly larger space of all network structures (i.e., all DAGs). Moreover, to facilitate the efficient navigation of this larger space, we employ state-of-the-art learning systems based on order graphs as a (nearly) omniscient oracle. In addition to defining this new search space, we show that it enables learning with non-decomposable scores and use it to learn optimal Bayesian networks under order-modular priors in Chapter 4, which are non-decomposable. We further demonstrate the utility of this new search space by showing how it lends itself to enumerating the k-best structures, resulting in an algorithm that can be three orders of magnitude more efficient than existing approaches based on dynamic programming (DP) and integer linear programming (ILP) [THR10, CBJ13].

This chapter is organized as follows. In Section 3.2, we propose our new search space for learning Bayesian networks. In Section 3.3, we show how our search space can be leveraged to find optimal Bayesian networks under a class of non-decomposable priors. In Section 3.4, we show how our search space can be further used to efficiently enumerate the k-best network structures. Proofs of theorems are provided in the Appendix.

3.2 A New Search Space: BN Graphs

In this section, we propose the new search space *BN graph*. Figure 3.1 illustrates a BN graph over variables $\mathbf{X} = \{X_1, X_2, X_3\}$. In the BN graph, nodes represent Bayesian network structures, i.e., DAGs, over different subsets of the variables $\mathbf{X}$. A directed edge $G_i \xrightarrow{X\mathbf{U}} G_j$ from a DAG G_i to a DAG G_j exists in the BN graph if and only G_j can be obtained from G_i by introducing variable X as a leaf node with parents $\mathbf{U}$. Hence, the BN graph, like the order graph, is a layered graph, but where each layer

adds one more leaf to an explicit, rather than just an implicit, DAG when we walk an edge to the next layer. Correspondingly, when we refer to a DAG G_i, we assume it is on the i-th layer, i.e., G_i has i nodes. The top 0-th layer contains the root of the BN graph, a DAG with no nodes, which we denote by G_0. The bottom n-th layer contains DAGs G_n over our n variables $\mathbf{X}$. Any path

$$G_0 \xrightarrow{X_1\mathbf{U}_1} \cdots \xrightarrow{X_n\mathbf{U}_n} G_n$$

from the root to a DAG G_n on the bottom layer, is a construction of the DAG G_n, where each edge $G_{i-1} \xrightarrow{X_i\mathbf{U}_i} G_i$ adds a new leaf X_i with parents $\mathbf{U}_i$. Moreover, each path corresponds to a unique ordering $\langle X_1, \ldots, X_n \rangle$ of the variables. Each edge $G_{i-1} \xrightarrow{X_i\mathbf{U}_i} G_i$ is associated with a cost, $\mathsf{score}(G_i \mid \mathcal{D}, \mathcal{P}) - \mathsf{score}(G_{i-1} \mid \mathcal{D}, \mathcal{P})$, and thus the cost of a path from the empty DAG G_0 to a DAG G_n gives us the score of the DAG, $\mathsf{score}(G_n \mid \mathcal{D}, \mathcal{P})$. We assume structure scores observe $\mathsf{score}(G_i \mid \mathcal{D}, \mathcal{P}) - \mathsf{score}(G_{i-1} \mid \mathcal{D}, \mathcal{P}) > 0$, such as the BDeu and MDL scores. Then we note that when the scores are decomposable, edge $G_{i-1} \xrightarrow{X_i\mathbf{U}_i} G_i$ has cost $\mathsf{score}(X_i\mathbf{U}_i \mid \mathcal{D}, \mathcal{P})$, corresponding to the cost associated with an edge in the order graph.

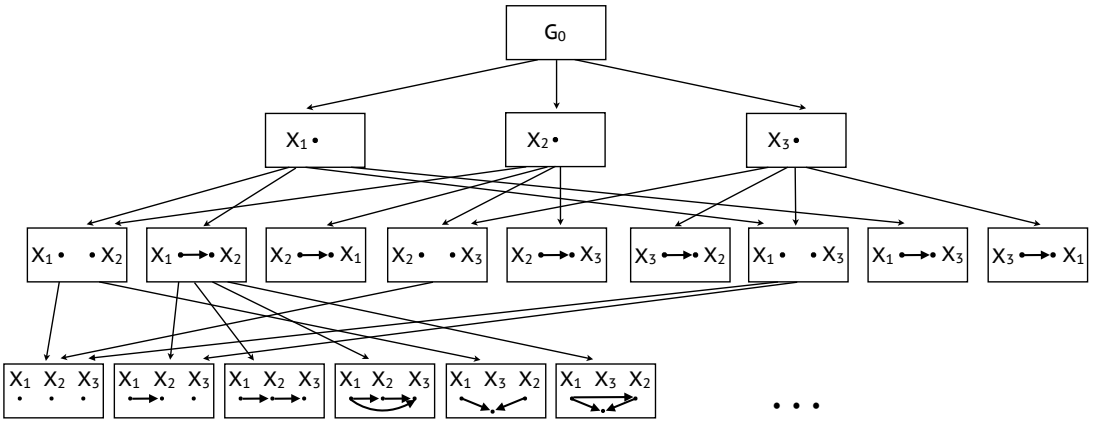


Figure 3.1: A Bayesian network graph (BN graph) for variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

For example, consider the BN graph of Figure 3.1 and the following path, i.e., sequence of DAGs:

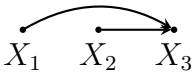
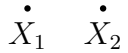
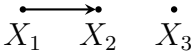
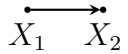
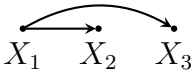
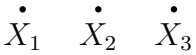
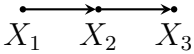
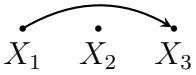
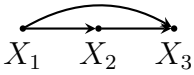
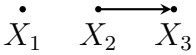
G_0	G_1	G_2	G_3
	X_1	$X_1 \rightarrow X_2$	$X_1 \rightarrow X_2 \quad X_3$

Starting with the empty DAG G_0, we add a leaf X_1 (with no parents), then a leaf X_2 (with parent X_1), then a leaf X_3 (with no parents), giving us a DAG G_3 over all 3 variables in $\mathbf{X}$.

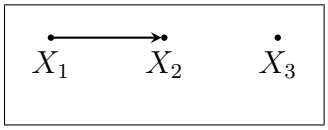
Finally, we note that both the order graph and the BN graph formulate the structure learning problem as a shortest path problem. The BN graph is however much larger than the order graph: an order graph has 2^n nodes, whereas the BN graph has $O(n! \cdot 2^{\binom{n}{2}})$ nodes. Despite this significant difference in search space size, we are still able to efficiently find shortest-paths in the BN graph, which we shall illustrate empirically.

3.3 Structure Learning with Non-Decomposable Scores

In this section, we consider the problem of learning an optimal Bayesian network structure using non-decomposable scores. As mentioned in Section 2.4.1, the K2 algorithm in general does not accommodate non-decomposable scores, since an optimal DAG cannot be obtained by choosing the optimal set of parents for each variable X, from those variables that precede X in the ordering $\prec$. To see this, suppose we are searching for an optimal DAG that is compatible with topological ordering $\langle X_1, X_2, X_3 \rangle$, using the following non-decomposable scores:

G	$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$	G	$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$
	1		25
	10		20
	15		25
	25		25
	25		25
	25		

The optimal DAG in this case is



Yet, we cannot learn this DAG by choosing the optimal set of parents for each variable X following ordering $\langle X_1, X_2, X_3 \rangle$. In particular, when considering X_2 alone, its best choice of parents from preceding variables is to have none; while in the optimal DAG, the best parent is X_1. Hence, the K2 algorithm and approaches based on the order graph, which can be viewed as a generalization of K2, does not accommodate non-decomposable scores.

The BN graph, on the other hand, naturally supports non-decomposable scores, as the BN graph contains all the possible DAGs as its leafs. Next, we discuss how to use the BN graph to find optimal DAGs using A* search. First, A* is an optimal search algorithm, which we describe next. Next, A* search is suited for search spaces that are graphs and not trees; in contrast, methods such as depth-first search would be inefficient in the BN graph, as it could revisit nodes exponentially many times.

3.3.1 Heuristic Functions for Non-Decomposable Scores

The A* search is a best-first search that uses an *evaluation* function f to guide the search process, where we expand first those nodes with the lowest f cost [HNR68, RN10]. The evaluation function for A* takes the form:

$$f(G) = g(G) + h(G), \tag{3.1}$$

where G is a given DAG, function g denotes the *path cost*, i.e., the cost of the path to reach G from G_0; and function h denote the *heuristic function*, which estimates the cost to reach an optimal goal, starting from G. More specifically, in the BN graph, the path cost $g(G)$ is $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$; heuristic function $h(G)$ estimates the score increase from G to any leaf G_n, i.e., from $\mathsf{score}(G \mid \mathcal{D}, \mathcal{P})$ to $\mathsf{score}(G_n \mid \mathcal{D}, \mathcal{P})$; and $f(G)$ represents a best estimate of the lowest path from G_0 to any leaf G_n that passes through G.

If the heuristic function h is *admissible*, i.e., it never over-estimates the lowest cost to reach a goal, then A* search is optimal [HNR68]. That is, the first goal node G_n that A* expands is one that has the lowest-cost path from the root G_0, and G_n is an optimal DAG. Moreover, when the heuristic function h is a more accurate estimate of the cost to any goal state, the search becomes more efficient. In Chapter 4 and 6, we will show how these admissible heuristics can be designed.

In a typical use of A* search, one tries to find a lowest-cost path from the root to a given goal node. In the BN graph, we don not know the goal node except that it lies on the last layer of the graph, i.e., a leaf node. Each such node represents a DAG over all variables. Moreover, we use A* search to find a leaf node that has a lowest-cost path from the root. Leaf nodes with such a property identify an optimal Bayesian network structure. Hence, we use A* search in a somewhat non-standard way. We shall revisit this distinction next, in the context of enumerate the k-best DAGs.

3.4 Enumerating the k-Best Structures

In this section, we consider another learning task, using the BN graph — enumerating the k-best Bayesian network structures. In some situations, learning a single optimal DAG is not sufficient — a single best DAG is subject to noise and other idiosyncrasies in the data. As such, a data analyst would want to be aware of other likely DAGs. Hence, a number of algorithms have been proposed to *enumerate* the k-most likely DAGs from a complete dataset, but with decomposable scores, using an augmented order graph and ILP [THR10, BC13, CBJ13, CT14, CCD15]. Such methods further facilitate approximate Bayesian model averaging [THR10, CT14]. We show that empirically, the BN graph can be *orders of magnitude* more efficient than these previous approaches on real-world datasets, on enumerating the k-best structures with decomposable scores.

Enumerating the k-best structures is straightforward when we perform the A* search on the BN graph. In particular, the k-th best DAG can be obtained by finding the goal node G_n that has the k-th lowest-cost path to the root G_0. We can thus enumerate the k-best DAGs by simply continuing the A* search, rather than stopping when we reach the first goal node [DFM12].[1] More specifically, an admissible heuristic function h for finding the first goal remains admissible for the k-th best goal, as h still lower bounds the cost to reach any goal. That is, it may just estimate the cost to reach a goal node that was already enumerated, and hence has a lower cost. We can thus continue to employ the same heuristic in A* search, to enumerate the remaining k-best goals.

We further note a distinction between the BN graph and the order graph. In the BN graph, DAGs are represented explicitly, whereas in an order graph, DAGs are implicit. In particular, each node $\mathbf{Y}$ in the order graph represents just a single optimal DAG over the variables $\mathbf{Y}$. Hence, the k-th best DAG may not be immediately recoverable. That is, we may not be able to obtain the k-th best DAG starting from an optimal sub-DAG

[1] We remark, however, that [DFM12] is more specifically concerned with the enumeration of the k-lowest-cost paths, rather than the k-lowest-cost goals.

— we can only guarantee that we obtain a single optimal DAG. While it is possible to augment the order graph, so that nodes in the order graph maintain additional information to recover the k-best DAGs, and find the k-best DAGs with dynamic programming, as in [THR10], this is not as effective as searching the BN graph, as we shall soon see.

Finally, we consider another approach to enumerating the k-best DAGs, based on integer linear programming (ILP) [CBJ13]. Basically, once an optimal DAG is obtained from an ILP, a new ILP can be obtained, whose optimal solution corresponds to the next-best DAG. More specifically, we assert additional constraints that exclude the optimal DAG that we found originally. This process can be repeated to enumerate the k-best DAGs.

3.4.1 Heuristic Functions for k-Best Structures

In this section, we consider heuristic functions to enumerate the k-best Bayesian networks. Consider the special but extreme case, where we have access to a *perfect* heuristic function $h(G)$, which could predict the optimal cost from G to any goal node G_n. That is,

$$h(G) = \min_{G_n : G \rightsquigarrow G_n} \mathsf{score}(G_n \mid \mathcal{D}, \mathcal{P}) - \mathsf{score}(G \mid \mathcal{D}, \mathcal{P}). \tag{3.2}$$

In this case, the A* search marches straight to the first optimal DAG, with appropriate tie-breaking that expands the deepest node first. Then the A* search enumerates other DAGs with the optimal score.[2] Once all the optimal DAGs are exhausted, $h(G)$ is no longer perfect, but still admissible for enumerating the remaining DAGs.

For decomposable scores, we do in fact have access to a perfect heuristic for finning the single best structure — any learning system for decomposable sores could be used as such, provided that it can accept a DAG G, and find an optimal DAG G_n that extends

[2]Typically, Bayesian network structures that have the same scores are Markov equivalent.

it. That is,

$$h(G) = \min_{G_n : G \rightsquigarrow G_n} \sum_{X\mathbf{U} \in G_n - G} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}, \mathcal{P}), \tag{3.3}$$

where we sum over families $X\mathbf{U}$ that appear in G_n but not in G.

Systems such as URLEARNING meet this criterion [YM13][3], which we use in our subsequent experiments. Such a system is treated as a black-box, and used to evaluate our heuristic function in the A* search, to potentially solve a learning problem that the black-box was not originally designed for.

We also remark that using such a black-box to evaluate a heuristic function, as described above, is also a departure from the standard practice of heuristic search. Conventionally, in heuristic search, one seeks heuristic functions that are cheap to evaluate, allowing more nodes to be evaluated, and hence more of the search space to be explored. Our black-box that finds an optimal extension of a DAG, in contrast, will be relatively expensive to evaluate. However, for the particular learning tasks that we consider, a strong heuristic can outweigh the expense to compute it, by more efficiently navigating the search space, i.e., by expanding fewer nodes to find a goal.

3.4.2 Implementation of the A* Search

Finally, we describe two further design decisions that are critical to the efficiency of A* search on the BN graph. First, if two DAGs G and G' are over the same set of variables, then they have the same heuristic value, i.e. $h(G) = h(G')$. Hence, we can cache the heuristic value $h(G)$ for a DAG G, and simply fetch this value for another DAG G', rather than re-invoking our black-box, when G' has the same set of variables as G. As a result, the heuristic function is invoked at most once for each subset $\mathbf{Y}$ of the variables $\mathbf{X}$. In addition, when we invoke our black-box on a DAG G, we can infer and then prime other entries of the cache. In particular, when our black-box returns an optimal completion G_n of a DAG G, then we know the optimal completion and heuristic values

[3] At `sites.google.com/site/bmmalone/files/urlearning`.

of any DAG in between G and G_n in the BN graph — their optimal completion is also G_n, from which we can infer the corresponding heuristic value.

Next, the branching factor of the BN graph is large, and hence we can quickly run out of memory if we expand each node and insert all of its children into the open list, i.e., the priority queue for the next node to expand by the A* search. We thus use partial-expansion A* in our implementation, i.e., when we expand a node G and generate its children, we insert child G' into the open list if and only if $f(G') \leq c$, where c is a given cut-off value. We can increase c and re-expand this node G, as many times as needed. In this thesis, we initialize c as $f(G)$, and increase its value by 1 with re-expansion. While we may spend extra work re-expanding nodes, this form of partial-expansion can save a significant amount of memory, without sacrificing the optimality of the A* search [YMI00, FGS12].

More specifically, similar to [YM13], before the A* search starts, for each X, we sort all of its possible parents in ascending order by $\mathsf{score}(X\mathbf{U} \mid \mathcal{D})$. When we generate a child DAG G' by appending variable X to DAG G, we scan the list sorted parents to find parents $\mathbf{U}$, from variables in G; once $f(G') > c$, since the parents are sorted, the generation stops. Finally, we note that this procedure generalizes the child generation approach of the A* search on order graph [YM13].

3.4.3 Experiments

We compare A* search, based on the BN graph, with two other recently proposed k-best structure learning algorithms: (1) KBEST,[4] which is based on an augmented order graph and dynamic programming [THR10], and (2) GOBNILP,[5] which is based on ILP [Cus11].

We use real-world datasets, from the UCI machine learning repository [BL13], and the National Long Term Care Survey (NLTCS). For learning, we assumed BDeu scores,

[4] At `www.cs.iastate.edu/~jtian/Software/UAI-10/KBest.htm`
[5] At `www.cs.york.ac.uk/aig/sw/gobnilp/`

benchmark			10-best				100-best				$1,000$-best			
			BN graph		KBEST		BN graph		KBEST		BN graph		KBEST	
name	n	N	t	m	t	m	t	m	t	m	t	m	t	m
wine	14	178	0.07	1	5.24	1	0.10	1	162.69	1	0.44	1	4,415.98	4
nlts	16	16,181	2.59	1	18.84	1	4.10	1	787.52	1	5.60	1	$\times_t$	
letter	17	20,000	8.54	1	42.16	1	13.30	1	1,849.29	2	18.86	1	$\times_t$	
voting	17	435	2.66	1	59.29	1	2.85	1	2,507.72	2	7.87	1	$\times_t$	
zoo	17	101	4.24	1	58.25	1	4.74	1	2,236.13	2	7.34	1	$\times_t$	
statlog	19	752	38.29	1	291.88	1	61.31	1	$\times_t$		64.12	1	$\times_t$	
hepatitis	20	126	39.79	2	675.34	2	76.34	2	$\times_t$		154.27	2	$\times_t$	
imports	22	205	288.29	8	2,646.41	8	383.32	8	$\times_t$		383.62	8	$\times_t$	
parkinsons	23	195	678.74	16	6,350.58	16	679.74	16	$\times_t$		1,297.75	16	$\times_t$	

Table 3.1: The BN graph and KBEST: A comparison of the time t (in seconds) and memory m (in GBs) used. An $\times_t$ corresponds to an out-of-time (7,200s), and an $\times_s$ corresponds to segmentation fault. n denotes the number of variables in the dataset, and N denotes the size of the dataset.

with an equivalent sample size of 1. Our experiments were run on a 2.67GHz Intel Xeon X5650 CPU, with access to 144GB RAM. We further pre-compute the BDeu scores, which are fed as input into each system evaluated. All timing results are averages over 10 runs. For each approach, we enumerate the 10-best, 100-best and $1,000$-best BNs, over a variety of real-world datasets. We impose a 7,200 second limit on running time. To analyze memory usage, we incrementally increased the amount of memory available to each system, from 1GB, 2GB, 4GB, 8GB, 16GB, and up to 64GB, and recorded the smallest limit that allowed each system to finish.

Table 3.1 summarizes our results for the A* search on the BN graph, and for the order-graph-based dynamic programming approach of KBEST. We omit the results for the ILP-based approach of GOBNILP, which ran out-of-memory (given 64GB) for all instances, except for the case of 10-best networks on the wine dataset, which took $2,707.13s$ and under 8GB of memory.[6]

[6]Note that GOBNILP is more effective in domains where, for example, we can constrain the number of parents that a node can have [YM13, MKJ14]. However, for our experiments here, we consider the

benchmark		10-best		100-best		$1,000$-best	
name	n	T_h	T_{A*}	T_h	T_{A*}	T_h	T_{A*}
wine	14	0.04	0.03	0.04	0.06	0.06	0.38
nltcs	16	2.58	0.01	4.08	0.02	5.36	0.24
letter	17	8.52	0.02	12.90	0.10	17.64	1.22
voting	17	2.62	0.04	2.66	0.19	5.39	2.48
zoo	17	4.22	0.02	4.72	0.02	7.16	0.18
statlog	19	38.22	0.07	61.14	0.17	63.75	0.37
hepatitis	20	39.61	0.18	75.91	0.43	151.60	2.67
imports	22	287.76	0.53	382.63	0.69	382.80	0.82
parkinsons	23	677.57	1.17	678.54	1.20	1,295.68	2.07

Table 3.2: The BN graph: The time T_h to compute the heuristic function and the time T_{A*} to traverse the BN graph with A* (in seconds).

We observe a few trends. First, the A* search on the BN graph can be over *three orders of magnitude* more efficient than KBEST, at enumerating the k-best DAGs. For example, when enumerating the 100-best and 1000-best DAGs on the wine dataset. Next, we observe that the A* search is consistently more efficient than KBEST at enumerating the k-best networks, scaling to larger networks, and to larger values of k. In fact, KBEST appears to scale super-linearly with k, but the A* search appears to scale *sub-linearly* with respect to k. These differences are due in part to: (1) the more exhaustive nature of dynamic programming, which needs to maintain all of the partial solutions that can potentially be completed to a k-th best solution, and (2) the more incremental nature of the A* search, i.e., the next best solutions are likely to be in the open list already.[7] Finally, we see that the memory usage of these two approaches is comparable, although the A* search appears to be more memory efficient as we increase the number k of networks that we enumerate.

To gain more insight about the computational nature and bottlenecks of the A*

more general case, where we do not assume such a constraint.

[7]In general, we expect heuristic search methods, such as A*, to be more efficient than more exhaustive methods such as dynamic programming, as heuristic search methods can more intelligently navigate the search space, when a good heuristic function is available.

benchmark		10-best			100-best			$1,000$-best		
name	n	gen.	exp.	re-exp.	gen.	exp.	re-exp.	gen.	exp.	re-exp.
wine	14	6,615	1,948	0	10,767	8,110	0	49,809	40,992	39,992
nltcs	16	244	211	0	2,196	2,160	0	20,181	20,078	38,156
letter	17	293	285	0	2,213	2,178	12,468	18,223	18,165	171,650
voting	17	4,030	1,884	0	20,932	13,836	13,736	183,010	118,779	353,337
zoo	17	493	205	0	1,404	1,165	0	24,400	10,855	9,855
statlog	19	745	411	0	9,772	3,589	3,489	30,299	26,943	25,963
hepatitis	20	6,177	2,165	0	19,152	15,897	0	172,244	111,568	110,568
imports	22	883	251	0	7,703	2,416	0	32,718	15,924	0
parkinsons	23	305	104	0	4,968	1,679	0	50,450	16,197	0

Table 3.3: The BN graph: (1) The number of generated nodes. (2) The number of expanded nodes. (3) The number of re-expanded nodes (in partial-expansion A*).

benchmark	n	10-best	100-best	$1,000$-best
wine	14	107	107	274
nltcs	16	182	441	628
letter	17	159	413	678
voting	17	318	798	1,678
zoo	17	234	384	1,067
statlog	19	156	491	794
hepatitis	20	273	4,653	13,269
imports	22	150	389	426
parkinsons	23	91	256	1,032

Table 3.4: The BN graph: The number of times the oracle is invoked to evaluate the heuristic function.

search on the BN graph, consider Table 3.2, which looks at how much time T_h that was spent on evaluating the heuristic function, versus the time T_{A*} that was spent in navigating the BN graph.[8] Table 3.3 further reports the number of nodes generated, i. e., the number of nodes inserted into the open list, and expanded by the A* search. First, we observe that the vast majority of the time spent in search is spent in evaluating the heuristic function. This is expected, as evaluating our black-box heuristic function

[8]We note that $t = T_h + T_{A*}$, with the total time t corresponding to those reported in Table 3.1.

is relatively expensive. Next, we observe that the number of nodes expanded is relatively small, which suggests that our black-box heuristic is indeed powerful enough to efficiently navigate the large search space of the BN graph. We also remark again that due to the caching of heuristic values, discussed in Section 3.4.2, the number of times that our black-box is invoked can be much smaller than the number of nodes generated. This is illustrated in Table 3.4.

CHAPTER 4

Learning with Order-Modular Priors

In this chapter, we consider order-modular priors, which are non-decomposable and whose evaluation is known to be $\#P$-hard. We show that the order-modular prior can be evaluated while performing A* search, with a slight variation on A* search. We also show how the framework developed in Chapter 3 can be used to find an optimal network structure under this prior. This chapter is based on [CCD15].

4.1 Introduction

In this chapter, we use the framework discussed in Chapter 3 to learn optimal Bayesian networks under order-modular priors.

The simplest form of order-modular priors is the *uniform* order-modular prior $Pr(G)$, which is proportional to the number of topological orderings consistent with a DAG G, i.e., the number of its linear extensions, which we denote by $\#G$. Hence,

$$\log Pr(G) = \log \#G - \log C, \tag{4.1}$$

where C is a normalizing constant. For example, DAG G

$$\boxed{X_1 \to X_2 \quad X_3}$$

is consistent with topological orderings $\langle X_1, X_2, X_3\rangle, \langle X_1, X_3, X_2\rangle$ and $\langle X_3, X_1, X_2\rangle$. Consequently, the order-modular prior $Pr(G)$ of G is $\log 3 - \log C$. More generally, order-modular priors can be viewed as a *weighted* count of linear extensions [KS04].

For the remained of this chapter, we assume uniform order-modular priors, and discuss the general order-modular priors in Appendix B.

Order-modular priors are notable, as they enable MCMC methods for approximate Bayesian model averaging [FK00]. They also enable some dynamic-programming-based methods for exact Bayesian model averaging, when there are a moderate number of network variables [KS04]. However, to our knowledge, only approximate approaches had been previously considered for this prior, when one wants a single optimal DAG. One significant difficulty is that counting the number of linear extensions of a graph is #P-complete [BW91]. Hence, it is intractable to even evaluate the score of a Bayesian network structure, using this prior. let alone to find an optimal Bayesian network structuring [KS04]. We shall revisit this issue.

We next develop a variation on A* search and a corresponding heuristic function for learning optimal network structures with order-modular priors. We then evaluate the resulting system on real-world dataset.

4.2 A Heuristic for Order-Modular Priors

Recall the discussion in scoring Bayesian network structures in Section 2.2. Given structure priors $Pr(G)$ as background knowledge $\mathcal{P}$, it can be incorporated into the scores as

$$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P}) = \mathsf{score}(G \mid \mathcal{D}) - \log Pr(G), \tag{4.2}$$

where scoring function $\mathsf{score}(G \mid \mathcal{D})$ evaluates $-\log Pr(\mathcal{D} \mid G)$. Moreover, $\mathsf{score}(G \mid \mathcal{D})$ is typically decomposable. That is,

$$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P}) = \sum_{X\mathbf{U}} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}) - \log Pr(G). \tag{4.3}$$

First, we note when scores in Equation 4.3 is used for the BN graph, the cost edge

$G_i \xrightarrow{X\mathbf{U}} G_j$ can be written as:

$$\mathsf{score}(X\mathbf{U} \mid \mathcal{D}) - \log \frac{Pr_j(G_j)}{Pr_i(G_i)},$$

where $Pr_0(G_0) = 1$. We then propose a simple heuristic function for learning an optimal DAG with a uniform order-modular prior. Let $G_i \rightsquigarrow G_n$ indicate that a DAG G_n is reachable from DAG G_i in the BN graph. We propose to use the heuristic function $h(G_i) = h_1(G_i) + h_2(G_i)$, which has two components. The first component is:

$$h_1(G_i) = \min_{G_n : G_i \rightsquigarrow G_n} \sum_{X\mathbf{U} \in G_n - G_i} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}), \tag{4.4}$$

where we sum over families $X\mathbf{U}$ that appear in G_n but not in G_i. This component corresponds to the shortest path to any goal, based on the decomposable part of the score in Equation 4.3, ignoring the prior, i.e., maximizing the marginal likelihood. The second component is:

$$h_2(G_i) = \min_{G_n : G_i \rightsquigarrow G_n} -\log \frac{Pr_n(G_n)}{Pr_i(G_i)}, \tag{4.5}$$

This component corresponds to the shortest path to the goal, based on the prior part of the score in Equation 4.3, but ignoring the data, i.e., maximizing the prior.

Theorem 1. *The heuristic function $h(G_i) = h_1(G_i) + h_2(G_i)$ of Equations 4.4 & 4.5 is admissible.*

To use this heuristic function, we must perform two independent optimization problems, for h_1 and h_2. The first is the familiar optimization of a decomposable score, as in Section 3.4.1 Equation 3.3, which can be evaluated with existing structure learning algorithm for decomposable scores. The second is an optimization of the prior, independently of the data. Next, we show how to both evaluate and optimize this component, for uniform order-modular priors.

4.2.1 Optimizing the Prior

Here, we describe how to solve the component h_2 for a uniform order-modular prior. Again, we want to identify the most likely goal node G_n reachable from G_i, i.e., the

DAG G_n with the largest linear extension count. Remember that DAG G_i has i nodes. Since adding any edge to the DAG constrains its possible linear extensions, then the DAG G_n with the largest linear extension count simply adds the remaining $n - i$ nodes independently to DAG G_i. If $\#G_i$ is the linear extension count of DAG G_i, then

$$\#G_n = \#G_i \cdot (i+1) \cdots n$$

is the linear extension count of DAG G_n.[1]

Next, we have that:

$$Pr_i(G_i) = \frac{1}{C_i} \cdot \#G_i \quad \text{and} \quad Pr_n(G_n) = \frac{1}{C_n} \cdot \#G_n$$

where C_k is a normalizing constant:

$$C_k = \sum_{G_k} \#G_k = \sum_{G_k} \sum_{\pi \sim G_k} 1 = \sum_{\pi} \sum_{\pi \sim G_k} 1 = \sum_{\pi} 2^{\binom{k}{2}} = k! \cdot 2^{\binom{k}{2}}$$

and where $\pi \sim G_k$ denotes compatibility with an ordering π and a DAG G_k. Thus,

$$\frac{Pr_n(G_n)}{Pr_i(G_i)} = \frac{C_i}{C_n} \frac{\#G_n}{\#G_i} = \frac{C_i}{C_n} \cdot (i+1) \cdots n = 2^{\binom{i}{2} - \binom{n}{2}}$$

Hence, $h_2(G_i) = [\binom{n}{2} - \binom{i}{2}] \cdot \log 2$. We note that for all DAGs G_i in the same layer, the heuristic function $h_2(G_i)$ evaluates to the same value, although this value differs for DAGs in different layers.

Note, that we also need to be able to compute the linear-extension counts $\#G_i$ themselves, which is itself a non-trivial problem (it is #P-complete). We discuss this next.

4.2.2 Counting Linear Extensions

In Section 4.1, we highlighted the relationship between uniform order-modular priors and counting linear extensions. We now show that the BN graph itself facilitates the

[1] For each linear extension π of G_i, there are $(i+1)$ places to insert the $(i+1)$-th node, then $(i+2)$ places to insert the next, and so on. Thus, there are $(i+1) \cdots n$ ways to extend a given ordering over i variables to n variables.

counting of linear extensions, for many DAGs at once. Subsequently, we shall show that this computation can be embedded in the A* search itself.

Recall that any path in the BN graph, from G_0 to G_i, corresponds to an ordering of variables $\langle X_1, \ldots, X_i \rangle$. In fact, this ordering is a linear extension of the DAG G_i. Hence, the linear extension count $\#G_i$ of a graph G_i is the number of paths from the root G_0 to G_i, in the BN graph. For example, consider the BN graph in Figure 3.1 and DAG:

$$\boxed{X_1 \rightarrow X_2 \quad X_3}$$

There are 3 distinct paths in the BN graph from the root G_0 to the DAG above, one path for each topological order that the DAG is consistent with. Next, observe that the number of linear extensions of a DAG G_i, or equivalently, the number of paths that reach G_i, is simply the sum of the linear extensions of the parents of G_i, in the BN graph. For example, our DAG above has 3 linear extensions, and 2 parents in the BN graph:

$$\boxed{X_1 \rightarrow X_2} \quad \boxed{X_1 \quad X_3}$$

the first with one linear extension, and the second with two. In this way, we can count the linear extensions of DAGs in a BN graph, from top-to-bottom, sharing computations across the different DAGs.[2]

Consider how A* navigates the BN graph the BN graph during search. If A* expands a node only when all of its parents are expanded, then as described above, we can count the number of linear extensions of a DAG, when it gets expanded.[3] Thus, we can evaluate its prior, and in turn, the function f. It so happens that, we can moderately weaken the heuristic function that we just described, so that A* will in fact expand a node only when all of its parents are expanded.

[2] A similar algorithm for counting linear extensions is described in [NK13]

[3] In particular, every time that we expand a node G, we can increment each of its children's linear extension counts by $\#G$. Once we have expanded every parent of a child, the child's linear extension count is correct.

Theorem 2. *Assuming a uniform order-modular prior, the heuristic function*

$$h(G_i) = h_1(G_i) + h'_2(G_i)$$

allows A to count the linear extensions of any DAG it expands, where* $h'_2(G_i) = -\sum_{k=i+1}^{n} \log k \leq h_2(G_i)$ *with components* h_1 *and* h_2 *coming from Equations 4.4 & 4.5.*

4.2.3 A* Search

Algorithm 1: A* search for learning an optimal BN with uniform a order-modular prior $Pr(G)$.

Data: A dataset $\mathcal{D}$ over variables $\mathbf{X}$.

Result: An optimal Bayesian network structure G maximizing

$$Pr(G \mid \mathcal{D}) \propto \textstyle\sum_{X\mathbf{U}} \mathsf{score}(X\mathbf{U}|\mathcal{D}) - \log Pr(G).$$

begin

- $H \leftarrow$ min-heap with only $(G_0, f(G_0), 1)$, where 1 is the number of linear extensions of G_0; and the heap is ordered by f
- **while** $H \neq \emptyset$ **do**
  - extract the minimum item (G, f, l) from H
  - **if** $V(G) = \mathbf{X}$ **then** return G
  - **foreach** G' obtained by adding a leaf to G **do**
    - **if** G' is not in H **then**
      - insert into H: $(G', \mathsf{score}(G'|\mathcal{D}) - \log l + h(G'), l)$
    - **else**
      - let (G', f', l') be in H, decrease f' by $\log \frac{l'+l}{l'}$, increase l' by l; and reheapify

Algorithm 1 provides pseudo-code for the A* search using a uniform order-modular prior. Note that this pseudo-code deviates slightly from the standard the A* search, as the linear extension counts $\#G$ are computed incrementally during the search.

Theorem 3. *Algorithm 1 learns an optimal Bayesian network with a uniform order-modular prior.*

Finally, we note that Algorithm 1 can be generalized for general order-modular priors, as discussed in Appendix B. We also note that Theorem 1 and the heuristic functions of Equation 4.4 & 4.5 were proposed for order-modular priors. In principle, the shortest-path formulation, and the heuristic function that we proposed, can support a much broader class of non-decomposable priors. However, one must be able to optimize the probability of a graph, as in the component h_2 of the heuristic function that we proposed, in Equation 4.5. If we had access to some oracle that can solve this component, then we would in principle have the pieces that are sufficient to perform A* search over the BN graph, using the corresponding prior.

4.3 Experiments

We evaluate our A* search approach to learning optimal Bayesian networks with real-world datasets, from the UCI machine learning repository [BL13], and the National Long Term Care Survey (NLTCS). For learning, we assumed BDeu scores, with an equivalent sample size of 1. Our experiments were run on a 2.67GHz Intel Xeon X5650 CPU, with access to 144GB RAM. We pre-compute the BDeu scores, which are fed as input into our system, and impose a 64GB second limit on running time. All timing results are averages over 10 runs.

In Table 4.1, we find that our approach can scale up to 17 variables on real-world datasets, i.e., the letter and voting datasets. We also note that with more data, and with more of the probability mass concentrated on fewer DAGs, traversing the BN graph with A* search appears to become more efficient. In particular, consider the time spent in A* search, i.e., T_{A*}, and the number of nodes generated, i.e., gen., in the datasets adult and wine, which both have 14 variables. Similarly, consider the datasetsletter and voting, which both have 17 variables. Moreover, consider dataset zoo, also over

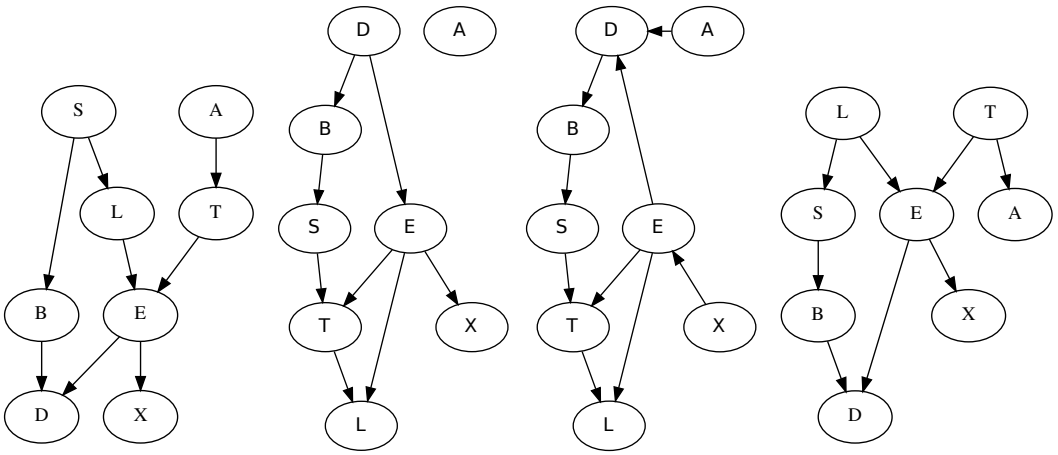
benchmark	n	N	T_h	T_{A*}	t	gen.	exp.	re-exp.
adult	14	30,162	1.03	0.26	1.29	106,832	12,620	33
wine	14	435	0.74	6.08	6.82	1559,900	244,694	57,259
nltcs	16	16,181	7.21	1.17	8.38	386,363	41,125	1
letter	17	20,000	29.42	3.79	32.20	360,899	37,034	16
voting	17	435	5.28	56.59	61.89	10540,132	1961,602	396,084
zoo	17	101	$\times_m$					

Table 4.1: The BN graph: The performance of A* search when learning with the uniform order-modular prior: (1) The time T_h to compute the heuristic function. (2) The time T_{A*} to traverse the BN graph with A* (in seconds) (3) The total time $t = T_h + T_{A*}$ spent in A* (4) The number of generated nodes. (5) The number of expanded nodes. (6) The number of re-expanded nodes (in partial-expansion A*). An $\times_m$ corresponds to an out-of-memory (64 GB).

17 variables, which was a very small dataset, containing only 101 instances. Here, A* search exhausted the 64GB of memory that it was allowed. We remark that, to our knowledge, ours is the first system for finding optimal DAGs using order-modular priors.[4]

In Figure 4.1, we consider a simple example, highlighting the effect that a uniform order-modular prior can have on the structure we learn. In Figure 4.1(a), we have the classical `asia` network, which we used to simulate datasets of different sizes. First, we simulated a small dataset of size 2^7 and learned two networks, one with a prior, Figure 4.1(b), and one without a prior, Figure 4.1(c). Ignoring node A, the two networks are Markov-equivalent. However, including node A, their linear extension counts are very different: 96 for network 4.1(b) but only 3 for network 4.1(c). This difference can explain why variable A is disconnected in 4.1(b), as a disconnected node non-trivially increases the linear extension count, and hence, the weight of the prior. In Figure 4.1(d), both cases, with and without the prior, learned precisely the same network when we

[4]There are systems available for (a) finding optimal DAGs using structure-modular priors, (b) for Bayesian model averaging using order-modular priors, and (c) for jointly optimizing over orders and DAGs, using order-modular priors. These tasks are all discussed in [KS04], which further states that finding optimal DAGs with order-modular priors is a more challenging problem (where we maximize over DAGs, but sum over orders).



(a) (b) (c) (d)

Figure 4.1: A network `asia` (a), and networks learned with dataset size 2^7 with a prior (b), without a prior (c), and a network learned with dataset size 2^{14} (d).

raised the size of the dataset to 2^{14}. This network is Markov-equivalent to the ground truth network that generated the data.

CHAPTER 5

A New Search Space for Learning Markov-Equivalent Network Structures

In this chapter, we consider *specializing* the BN graph to a more compact search space, called the *EC graph*, over Markov-equivalent Bayesian network structures. The EC graph is further canonized into another search space, called the *EC tree*, so that each node in the search space is reached by exactly one path. Empirically, we show that our proposed approach is in practice orders of magnitude more efficient than the existing state-of-the-art, which is based on an augmented order graph and dynamic programming [CT14]. This chapter is based on [CCD16].

5.1 Introduction

Learning the structure of a Bayesian network is a fundamental problem in machine learning and artificial intelligence. Historically, approximate methods, such as Markov Chain Monte Carlo (MCMC) and local search, were used for this task. In the past decade, there has been a surge in interest, in finding *optimal* Bayesian network structures, i.e., learning a single best directed acyclic graph (DAG) from a complete dataset; see, e.g., [KS04, SM05, SM06, JSG10, Cus11, YM13].

In some situations, learning a single optimal DAG is not sufficient—a single DAG is subject to noise and other idiosyncrasies in the data. As such, a data analyst would want to be aware of other likely DAGs. Hence, a number of algorithms have been proposed to *enumerate* the k-most likely DAGs from a complete dataset [THR10, CBJ13,

CT14, CCD15]. Such methods further facilitate approximate Bayesian model averaging [THR10, CT14].

There is a fundamental inefficiency in enumerating the k-most likely DAGs, namely that any given DAG may be Markov equivalent to many other DAGs, which are all equally expressive in terms of representing probability distributions. Thus, by enumerating DAGs, one may spend a significant amount of effort in enumerating redundant Bayesian networks. In this chapter, we consider instead the enumeration of their equivalence classes, with each equivalence class representing a potentially large (even exponential) number of DAGs, which we show can be the case in practice empirically.

In this chapter, we propose a new approach to enumerating equivalence classes that is in practice orders of magnitude more efficient than the existing state-of-the-art based on dynamic programming [CT14]. Our approach is based on the framework of navigating the BN graph, which we discussed in Chapter 3. We propose a specific instance of this framework, where we specialize the BN graph to a more compact search space over equivalence classes.

This chapter is organized as follows. This chapter is organized as follows. Section 5.2 reviews Markov equivalence and related concepts. In Section 5.3, we propose the EC graph, and then canonize the EC graph to the EC tree. Finally, we evaluate our approach empirically in Section 5.4.

5.2 Markov-Equivalent Bayesian Network Structures

In this section, we review related previous work on Markov-equivalent Bayesian network structures, i.e., DAGs. Two given DAGs are considered Markov-equivalent if and only if they encode the same conditional independencies, and consequently, represent the same class of probability distributions. For example, the following three DAGs are Markov-equivalent:

$X_1 \rightarrow X_2 \rightarrow X_3$	$X_1 \leftarrow X_2 \leftarrow X_3$	$X_1 \leftarrow X_2 \rightarrow X_3$

Markov equivalence can be characterized by a graphical criterion, based on the structure of a DAG. First, the *skeleton* of a DAG is the undirected graph found by ignoring the orientation of the edges. Second, a v-*structure* in a DAG is a set of three nodes X, Y, Z with edges $X \rightarrow Y \leftarrow Z$, but with no edge between X and Z. The following theorem characterizes Markov-equivalent DAGs.

Theorem 4. *[VP90] Two DAGs are Markov-equivalent if and only if they have the same skeleton and the same v-structures.*

A set of Markov-equivalent DAGs can be summarized by a *partially directed acyclic graph* (PDAG), which is a graph that contains both directed and undirected edges, but with no directed cycles [Chi02]. Given a PDAG P, we can induce a set of Markov-equivalent DAGs by directing the undirected edges of a PDAG, but as long as we introduce no directed cycles and no new v-structures. We use $\mathsf{class}(P)$ to denote this set of Markov-equivalent DAGs.

In an equivalence class of DAGs, each edge of their common skeleton can be classified as *compelled* or *reversible*. An edge connecting X and Y is compelled to a direction $X \rightarrow Y$ if *every* DAG in the equivalence class has the directed edge $X \rightarrow Y$. Otherwise, an edge is reversible, and there exists a DAG in the equivalence class with edge $X \rightarrow Y$, and another DAG with edge $Y \rightarrow X$. In a PDAG, if all compelled edges are directed in the appropriate orientation, and all reversible edges are left undirected, then we obtain a *completed PDAG* (CPDAG).[1] A CPDAG uniquely characterizes an equivalence class of DAGs [Chi02]. That is, there is a one-to-one correspondence between CPDAGs and equivalence classes. Further, a given CPDAG represents all DAGs, and only those DAGs, of a given equivalence class.

As an example, the CPDAG $X_1 - X_2 - X_3$ represents the Markov equivalence class for the three DAGs given in our example from the start of this section. We provide

[1] CPDAGs are also sometimes referred to as *essential graphs* or *maximally oriented graphs*.

another example below, of a CPDAG, in Figure 5.1(a); and its corresponding DAGs in Figure 5.1(b)-(d).

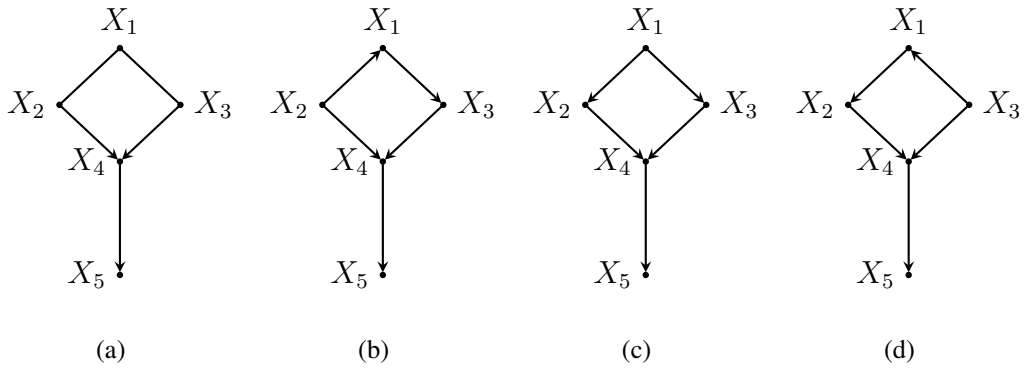


Figure 5.1: A CPDAG and the DAGs it represents. (a) CPDAG (b) - (d) DAGs.

Finally, we note that some scores of Bayesian network structures do not observe Markov equivalence, i.e., the scores may assign different values to structures encode the same conditional independencies, and consequently, represent the same class of probability distributions. For example, scores with uniform order-modular prior is not Markov-equivalent, as among structures that represents the same conditional independencies, the scores favors the ones with a larger number of linear extensions. In the rest of this chapter, we assume Markov-equivalent scores, which includes the MDL and BDeu scores.

5.3 A New Search Space: EC Trees

In this section, we propose a new search space for Bayesian network structures, but more specifically, for their equivalence classes. This is a more compact search space where each node now represents an equivalence class of DAGs, called the *EC graph*. We further propose a *canonization* of the EC graph, leading to the *EC tree*, which has desirable properties for heuristic search methods, such as the A* search, improving efficiency it terms of both time and memory.

5.3.1 EC Graphs

We now propose a new search space for Markov-equivalent Bayesian network structures, called the *EC graph*, where each node now represents an *equivalence class* of DAGs.

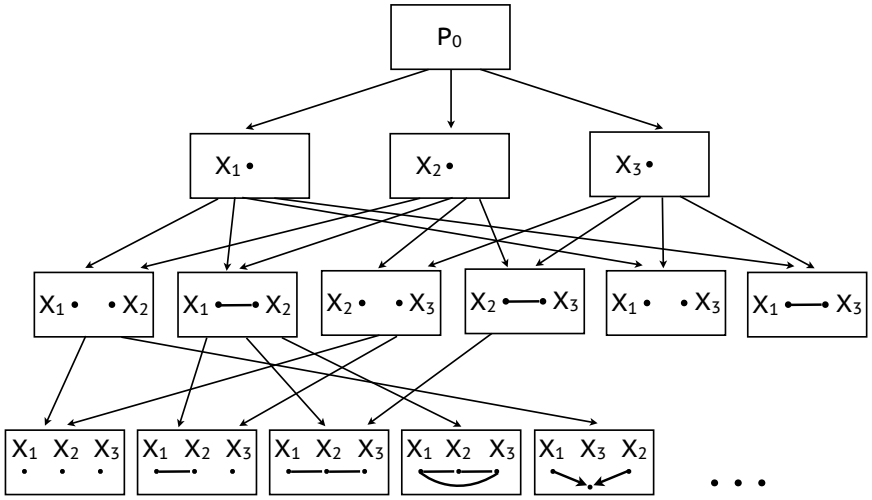


Figure 5.2: An EC graph for variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

In an EC graph, each node represents a CPDAG P, which denotes a set of Markov-equivalent DAGs G. Intuitively, we can obtain an EC graph by aggregating the Markov-equivalent DAGs of a BN graph into a single node, and labeling the resulting node with the corresponding CPDAG P.

Recall that in a BN graph, a directed edge $G_i \xrightarrow{X\mathbf{U}} G_j$ indicates that DAG G_j can be obtained from DAG G_i by adding to G_i a leaf node X with parents $\mathbf{U}$. In an EC graph, we have a corresponding directed edge $P_i \xrightarrow{X\mathbf{U}} P_j$. Here, the CPDAG P_i represents the equivalence class $\mathsf{class}(P_i)$, containing DAGs G_i. We can view the edge as adding a leaf node X with parents $\mathbf{U}$ to *each* of the DAGs $G_i \in \mathsf{class}(P_i)$. First, we observe that any of the resulting DAGs G_j must belong to the same equivalence class P_j.

Proposition 1. *Let P_i denote a CPDAG and let G_i denote a DAG in $\mathsf{class}(P_i)$. If we add a new leaf X_i with parents $\mathbf{U}_i$ to the DAGs G_i, the resulting DAGs G_j belong to*

the same equivalence class P_j.

Figure 5.2 illustrates an EC graph over variables $\mathbf{X} = \{X_1, X_2, X_3\}$. Consider, as an example, the CPDAG $X_1 - X_2$, which corresponds to the Markov-equivalent DAGs $X_1 \rightarrow X_2$ and $X_1 \leftarrow X_2$. Adding a new leaf X_3 with parent X_2, we obtain $X_1 \rightarrow X_2 \rightarrow X_3$ and $X_1 \leftarrow X_2 \rightarrow X_3$, with CPDAG $X_1 - X_2 - X_3$. We remark that there is a third DAG $X_1 \leftarrow X_2 \leftarrow X_3$ in this equivalence class, which we did not obtain here since X_3 is not a leaf. This DAG is obtained on a different path of the EC graph, where we add X_1 as a leaf to the CPDAG $X_2 - X_3$.

Proposition 2. *For a given CPDAG P, and any DAG $G \in \mathsf{class}(P)$, there exists a path that constructs G from root P_0 to node P in the EC graph.*

We remark that when we traverse an edge $P_i \xrightarrow{X\mathbf{U}} P_j$, we add a new leaf to the DAGs of $\mathsf{class}(P_i)$, yet the new edges in the resulting CPDAG P_j may be directed or undirected. Thus, to traverse the EC graph, we need to a way to orient the new edges from parents $\mathbf{U}$ to leaf X in CPDAG P_j. Previously, [Chi95] proposed a polytime algorithm that, given a DAG G, finds the corresponding CPDAG P. In principle, we could construct P_j from P_i by (1) picking some DAG $G_i \in \mathsf{class}(P_i)$ (2) adding a new leaf X with parents $\mathbf{U}$, and then (3) running Chickering's algorithm on the resulting DAG G_j, which is in $\mathsf{class}(P_j)$. We next observe that Chickering's algorithm can be run *incrementally*, by labeling only the new edges from $\mathbf{U}$ to X.

Proposition 3. *Consider a DAG G_i, its CPDAG P_i, and a DAG G_j obtained by adding a new leaf X to DAG G_i, with parents $\mathbf{U}$. The CPDAG P_j for G_j can be obtained locally from P_i, by applying Algorithm 2.*

Given a DAG G, Chickering's original algorithm traverses the nodes of a DAG G, in topological order, and labels the edges incoming a node as either compelled or reversible. Hence, running the same algorithm on a DAG G_j will first obtain the sub-CPDAG P_i. The edges incoming to the new leaf X can then be labeled by running an additional iteration of Chickering's algorithm, which is given by Algorithm 2.

Algorithm 2: LABELEDGES($P, X\,\mathbf{U}$)

Data: CPDAG P, new variable X with parents $\mathbf{U}$ in P.

Result: Label edges from X to $\mathbf{U}$ as compelled or reversible.

begin

label each edge between X and $\mathbf{U}$ as *unknown*

let G be any DAG in $\mathsf{class}(P)$

while there exists an *unknown* edge **do**

let $X' - X$ be the unknown edge with the greatest X' in a topological ordering of G

foreach $X'' \rightarrow X' \in P$ **do**

if $X'' \notin \mathbf{U}$ **then** label all unknown $Y - X$ incident to X as compelled to X

else label $X'' - X$ as compelled to X

if $\exists Z \in \mathbf{U}$ s.t. $Z \rightarrow X' \notin G$ **then** label all unknown $Y - X$ incident to X as compelled to X

else label all unknown $Y - X$ incident to X as reversible

As in the BN graph, each edge $P_{i-1} \xrightarrow{X_i \mathbf{U}_i} P_i$ is associated with a cost, $\mathsf{score}(G_i \mid \mathcal{D}) - \mathsf{score}(G_{i-2} \mid \mathcal{D})$, where DAG G_i is in $\mathsf{class}(P_i)$ and DAG G_{i-1} is in $\mathsf{class}(P_{i-1})$. Hence, the CPDAG P_n that has the shortest path from the root P_0 is an optimal equivalence class, whose DAGs have the lowest cost.

Finally, we note that since the EC graph can be viewed as a search space obtained aggregating the Markov-equivalent DAGs of a BN graph into a single node, a heuristic function is admissible in the EC graph if and only if it is admissible in the BN graph. As a result, heuristic functions design and caching used for the BN graph, as discussed in Section 3, and also be exploited by the EC graph. Similarly, partial-expansion A* search, discussed in Section 3, can also be incorporated.

5.3.2 EC Trees

For heuristic search methods such as the A* search, the structure of the search space has a significant impact on the efficiency of search. Consider the EC graph in particular. A CPDAG node P can be reached from the root P_0 via possibly many paths. Further, each path has the same cost, as Markov-equivalent DAGs have the same score. Thus, after we visit a node for the first time, we do not care about other paths that reach the same node.

This redundancy introduces significant memory and computational overheads to the A* search. Thus, we propose a *canonization* of the EC graph, that ensures that every CPDAG node P can be reached by a unique, canonical path. Here, each node has a single parent, and hence, we obtain a search tree.[2] Hence, we refer to the canonized space as the *EC tree*. Figure 5.3 depicts an EC tree over variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

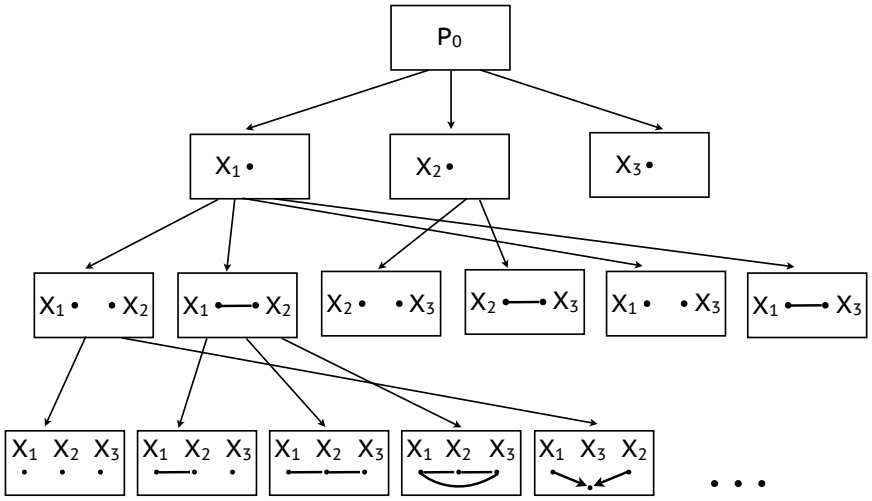


Figure 5.3: An EC tree for variables $\mathbf{X} = \{X_1, X_2, X_3\}$.

Consider any path $P_0 \xrightarrow{X_1\mathbf{U}_1} \cdots \xrightarrow{X_i\mathbf{U}_i} P_i$ from the root node P_0 to a node P_i, in the EC graph, which corresponds to an ordering of nodes, $\pi_i = \langle X_1, \ldots, X_i \rangle$. To define a canonical path from P_0 to P_i, it thus suffices to define a canonical ordering of

[2] In more technical terms, the savings that we obtain are: (1) duplicate detection is no longer needed, i.e., the closed list, and (2) fewer edges in the search space implies smaller a branching factor and fewer heuristic function evaluations.

the variables of P_i. In turn, to define an EC tree from an EC graph, it suffices to (1) associate each CPDAG node P with a canonical ordering π, and (2) show which edges of the EC graph remain in the EC tree, with respect to the canonical orderings.

First, let us define a canonical ordering for a given DAG G: let us use the largest topological ordering that is consistent with a given DAG G, but in reverse lexicographic order, where the right-most element in an order is the most significant.[3] We can construct such an order, from right-to-left, by iteratively removing the leaf with the largest index. For example, the DAG $X_1 \leftarrow X_2 \rightarrow X_3$ has two topological orderings: $\pi_a = \langle X_2, X_1, X_3 \rangle$ and $\pi_b = \langle X_2, X_3, X_1 \rangle$, where π_a is larger in reverse lexicographic order, i.e., we remove X_3 first, then X_1, and then X_2.

We now define a canonical ordering for a CPDAG P: let us use the largest canonical ordering of its Markov-equivalent DAGs $G \in \mathsf{class}(P)$. Consider the CPDAG $X_1 - X_2 - X_3$, and its Markov-equivalent DAGs:

$X_1 \rightarrow X_2 \rightarrow X_3$	$X_1 \leftarrow X_2 \leftarrow X_3$	$X_1 \leftarrow X_2 \rightarrow X_3$

with the canonical orderings: $\pi_a = \langle X_1, X_2, X_3 \rangle$, $\pi_b = \langle X_3, X_2, X_1 \rangle$, and $\pi_c = \langle X_2, X_1, X_3 \rangle$. Among these DAGs, ordering π_a is the largest, and is thus the canonical ordering of CPDAG $X_1 - X_2 - X_3$.

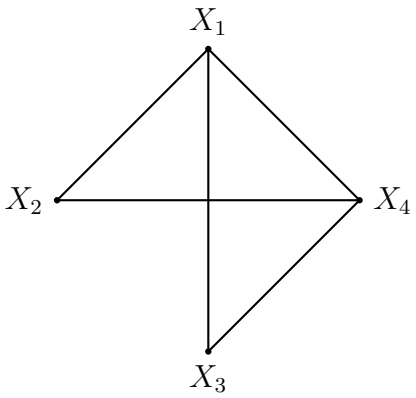
Given a CPDAG P, we can construct its canonical ordering π, again from right-to-left, by iteratively removing the largest leaf among the Markov-equivalent DAGs in $\mathsf{class}(P)$. As for obtaining the structure of the EC tree, this iterative process provides a local condition for determining whether an edge $P_{i-1} \xrightarrow{X_i \mathbf{U}_i} P_i$ belongs in the EC tree. That is, variable X_i must be the largest leaf among the Markov-equivalent DAGs in $\mathsf{class}(P_i)$. This is summarized by the following result.

Proposition 4. *Let π_{i-1} be the canonical ordering of CPDAG P_{i-1}, and let P_i be the CPDAG found by adding leaf X_i with parents $\mathbf{U}_i$ to the DAGs $G_{i-1} \in \mathsf{class}(P_{i-1})$.*

[3] Here, we assume comparisons are made based on the natural ordering of variables, i.e., by index.

In this case, $\pi_i = \langle \pi_{i-1}, X_i \rangle$ is the canonical ordering of P_i if and only if X_i has the largest index among all leaves in DAGs $G_i \in \mathsf{class}(P_i)$.

It remains to show how to identify, for a given CPDAG P, the largest leaf among the Markov-equivalent DAGs in $\mathsf{class}(P)$. Consider the following CPDAG:



We note that P cannot be obtained by appending X_4 as a leaf. If it could, then the resulting DAG would have a new v-structure $X_2 \to X_4 \leftarrow X_3$, since there is no edge connecting X_2 and X_3 (i.e., such a DAG would not belong in the equivalence class of P). As we cannot append X_4 as a leaf, it cannot be the last variable of any topological ordering of a DAG in $\mathsf{class}(P)$. However, there is a DAG in $\mathsf{class}(P)$ where X_3 is a leaf. The canonical ordering for P thus mentions X_3 last.

For a given CPDAG P_i, the following theorem allows us to enumerate all leaves among the DAGs in $\mathsf{class}(P_i)$, allowing us to easily test whether a node X appears as the largest leaf in some DAG of a given CPDAG P. Algorithm 3 further provides a polytime procedure for this test.

Theorem 5. *Consider a CPDAG P and variable X, with no compelled edges directed away from X. Let set $\mathbf{S}$ be the nodes adjacent to X through a reversible edge. In this case, there exists a DAG $G \in \mathsf{class}(P)$ where X is a leaf if and only if nodes $\mathbf{S}$ form a clique in P.*

Finally, we remark that the only difference between the EC tree and the EC graph is that each node in the EC tree can be reached through exactly one path, compared

Algorithm 3: VALIDEDGE($P_{i-1}, X_i\mathbf{U}_i$)

Data: CPDAG P_{i-1} and candidate family $X_i\mathbf{U}_i$

Result: **true** if $P_{i-1} \xrightarrow{X_i\mathbf{U}_i} P_i$ is valid, **false** otherwise

begin

let P_i be the CPDAG obtained by appending X_i to P_{i-1} (using Algorithm2)

foreach node X_k in P_i where $k > i$ **do**

if there exists compelled edge $Y \leftarrow X_k$ **then**

continue

let $\mathbf{S} = \{Y \mid Y - X_k$ is reversible in $P\}$

if variables in $\mathbf{S}$ form a clique **then**

return false

return true

to multiple paths in the EC graph. This distinction results in memory and computational savings for A* search, as discussed earlier. Otherwise, A* search in the EC tree proceeds in the same manner as in the EC graph. In particular, a heuristic function is admissible in the EC tree if and only if it is admissible in the EC graph.

5.4 Experiments

We compare our approach with the recently proposed algorithm for finding the k-best equivalence classes of Bayesian networks, called KBESTEC,[4] based on an augmented order graph and dynamic programming [CT14].

Our experiments were performed on a 2.67GHz Intel Xeon X5650 CPU. We use real-world datasets from the UCI ML Repository [BL13], and assume BDeu scores with an equivalent sample size of 1. We adapt the URLEARNING structure learning

[4]Open-source, available at `web.cs.iastate.edu/~jtian/Software/AAAI-14-yetian/KBestEC.htm`

benchmark			10-best				100-best				$1,000$-best			
			EC tree		KBESTEC		EC tree		KBESTEC		EC tree		KBESTEC	
name	n	N	t	m	t	m	t	m	t	m	t	m	t	m
wine	14	178	0.05	1	15.35	2	0.15	1	270.14	2	0.86	1	5,569.34	4
letter	17	20,000	18.11	1	120.26	2	48.31	1	2,559.38	2	81.72	1	$\times_t$	
voting	17	435	1.89	1	141.66	2	2.11	1	3,289.75	2	6.67	1	$\times_t$	
zoo	17	101	2.89	1	139.37	2	3.59	1	3,206.18	2	6.03	1	$\times_t$	
statlog	19	752	29.28	1	618.73	2	41.99	1	$\times_t$		43.89	1	$\times_t$	
hepatitis	20	126	36.33	1	1,328.27	2	63.37	1	$\times_t$		101.05	2	$\times_t$	
imports	22	205	174.84	4	$\times_s$		223.78	4	$\times_s$		224.11	4	$\times_s$	
parkinsons	23	195	897.81	8	$\times_t$		897.97	8	$\times_t$		898.68	8	$\times_t$	

Table 5.1: The EC tree and KBESTEC: A comparison of the Time t (in seconds) and memory m (in GBs) used. $\times_t$ denotes an out-of-time (7,200s), and $\times_s$ denotes a segmentation fault. n is the number of variables in the dataset, and N is the size of the dataset.

package of [YM13] and [FYM14],[5] We pre-compute the BDeu scores, which are fed as input into into each system evaluated. All timing results are averages over 10 runs.

For each approach, we enumerate the 10-best, 100-best and $1,000$-best CPDAGs, with a 7,200 second limit on running time. To analyze memory usage, we incrementally increased the amount of memory available to each system (from 1GB, 2GB, 4GB, to 8GB), and recorded the smallest limit that allowed each system to finish.

Table 5.1 summarizes our results for A* search on the EC tree, and for the order-graph-based dynamic programming approach of KBESTEC. First, we observe that on instances where both the A* search and KBESTEC are successful, A* search is consistently more efficient, both in terms of computation time and in memory usage. In terms of time, A* search can be *orders of magnitude* more efficient: for example, in the zoo dataset, the A* search is over 893 times faster than KBESTEC. We further observe that the A* search is capable of scaling to larger networks, and to larger values

[5]Open-source, available at `urlearning.org`.

of k. In fact, KBESTEC appears to scale super-linearly with k, but A* search appears to scale *sub-linearly* with respect to k. This trend can in part be attributed to the more exhaustive nature of dynamic programming, which maintains all partial solutions that can potentially be completed to a k-th best solution.

benchmark		10-best		100-best		$1,000$-best	
name	n	T_h	T_{A*}	T_h	T_{A*}	T_h	T_{A*}
adult	14	0.25	0.01	0.49	0.05	0.63	0.56
wine	14	0.03	0.02	0.05	0.10	0.09	0.77
nltcs	16	3.35	0.01	5.44	0.12	8.08	1.50
letter	17	18.07	0.04	47.74	0.57	75.54	6.18
msnbc	17	145.64	0.07	152.87	0.18	154.61	0.46
voting	17	1.88	0.01	1.92	0.19	4.16	2.51
zoo	17	2.88	0.01	3.55	0.04	5.83	0.20
statlog	19	29.27	0.01	41.94	0.05	43.49	0.40
hepatitis	20	36.24	0.09	62.79	0.58	96.82	4.23
imports	22	174.82	0.02	223.71	0.07	223.81	0.30
parkinsons	23	897.74	0.07	897.82	0.15	898.25	0.43

Table 5.2: The EC Tree: The Time T_h to compute the heuristic function and the time T_{A*} to traverse the EC tree with A* (in seconds).

To gain more insight about the computational nature of A* search on the EC tree, consider Tables 5.2 & 5.3, which includes 3 additional datasets, adult,nltcs, and msnbc[6]. In Table 5.2, we consider the amount of time T_h spent in evaluating the heuristic function, i.e., invoking our oracle, and the time T_{A*} spent traversing the EC tree in A* search.[7] Table 5.3 further reports the number of nodes generated, i.e., inserted into the open list, expanded, and re-expanded by partial-expansion in A* search. We also report the number of oracle invocations. First, observe that the A* search spends almost all of its time in evaluating the heuristic function, which we already know is relatively expensive. Next, observe that the the number of nodes generated is relatively low. This suggests that the oracle is powerful enough to efficiently navigate the search space of

[6]We were unable to generate score files for these datasets using KBESTEC

[7]We note that $t = T_h + T_{A*}$, with the total time t corresponding to those reported in Table 5.1.

benchmark		10-best				100-best				$1,000$-best			
name	n	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke
adult	14	243	220	630	67	2,045	1,865	12,355	265	15,072	13,574	163,462	465
wine	14	2,205	1,874	0	39	12,691	10,256	10,156	206	68,403	56,612	111,224	358
nltcs	16	252	249	2,151	287	1,407	1,389	27,069	689	11,431	11,193	356,755	1363
letter	17	377	377	2,936	324	2,705	2,704	59,915	1,562	16,057	15,979	659,076	3,373
msnbc	17	555	555	5,450	1,147	969	965	22,490	1,269	2,662	2,561	78,050	1,313
voting	17	1,617	1,419	0	147	15,971	11,613	23,026	413	114,498	106,378	421,512	1,402
zoo	17	192	151	0	166	1,978	1,029	929	377	9,330	8,812	7,812	864
statlog	19	393	369	0	212	3,379	3,065	2,965	444	26,698	24,063	46,126	685
hepatitis	20	2,431	2,281	0	667	17,875	15,852	15,752	2,191	121,022	99,956	197912	6423
imports	22	666	270	0	103	4181	2,034	0	244	21,544	14,064	0	259
parkinsons	23	836	524	0	214	4,222	2,616	0	237	26,390	17,134	0	318

Table 5.3: The EC tree: (1) The number of generated nodes. (2) The number of expanded nodes. (3) The number of re-expanded nodes (in partial-expansion A*). (4) The number of times the oracle is invoked to evaluate the heuristic function.

the EC tree. Further, we observe that the number of oracle invocations is also low, which is further minimized by caching and inferring heuristic values.

We further count the equivalent number of DAGs represented by the k-best CPDAGs, in Table 5.4. Previously, [GP01] observed that when the number of variables is small (not greater than 10), a CPDAG represents *on average* 3.7 DAGs. Here, we observe that for a moderate number of variables, a CPDAG may represent a much larger number of DAGs. That is, when we learn equivalence classes, the data may prefer CPDAGs with many reversible edges, deviating from the average case. For example, the larger datasets, i.e., larger N tend to have equivalence classes that contain many more DAGs.

When we compare the enumeration of the 10-best, 100-best and $1,000$-best equivalence classes, with the enumeration of an equivalent number of DAGs, using the system in Section 3, we can again see orders-of-magnitude improvements in efficiency; see Table 5.5. We observe similar gains by the EC tree, compared to the BN graph, in terms of nodes explored by A* search, as would be expected; see Table 5.6.

benchmark	n	N	10-best	100-best	$1,000$-best
adult	14	30,162	68	1,399	15,572
wine	14	178	60	448	4,142
nltcs	16	16,181	3,324	27,798	248,476
letter	17	20,000	884	15,796	569,429
msnbc	17	291,326	231,840	1,720,560	16,921,080
voting	17	435	30	413	3,671
zoo	17	101	52	377	5,464
statlog	19	752	44	444	4,403
hepatitis	20	126	89	892	8,919
imports	22	205	12	136	1,493
parkinsons	23	195	132	476	3,444

Table 5.4: Number of DAGs in the k-best equivalent classes.

benchmark			10-best EC				100-best EC				$1,000$-best EC			
			EC tree		BN graph		EC tree		BN graph		EC tree		BN graph	
name	n	N	t	m	t	m	t	m	t	m	t	m	t	m
adult	14	30162	0.26	1	0.47	1	0.54	1	1.49	1	1.19	1	11.87	1
wine	14	178	0.05	1	0.09	1	0.15	1	0.33	1	0.86	1	3.89	1
nltcs	16	16181	3.36	1	11.34	1	5.56	1	46.91	1	9.58	1	1,126.05	4
letter	17	20,000	18.11	1	20.68	1	48.31	1	84.07	1	81.72	1	4,666.29	4
msnbc	17	291,326	145.71	1	896.45	2	153.05	1	$\times_t$		155.07	1	$\times_t$	
voting	17	435	1.89	1	2.86	1	2.11	1	3.70	1	6.67	1	17.89	1
zoo	17	101	2.89	1	5.09	1	3.59	1	5.85	1	6.03	1	10.34	1
statlog	19	752	29.28	1	51.89	1	41.99	1	73.77	1	43.89	1	76.99	1
hepatitis	20	126	36.33	1	86.05	2	63.37	1	176.83	2	101.05	2	284.46	4
imports	22	205	174.84	4	455.65	8	223.78	4	603.68	8	224.11	4	604.14	8
parkinsons	23	195	897.81	8	779.90	16	897.97	8	1,034.50	16	898.68	8	1,450.46	16

Table 5.5: The EC Tree and the BN graph: A comparison of the Time t (in seconds) and memory m (in GBs) used. n is the number of variables in the dataset, and N is the size of the dataset. A $\times_t$ corresponds to an out-of-time (7,200s).

benchmark		10-best EC				100-best EC				1, 000-best EC			
name	n	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke	gen.	exp.	re-exp.	invoke
adult	14	1,672	1,352	3,852	173	30,619	27,015	179,312	634	245,687	215,753	2,602,353	1,050
wine	14	8,203	3,714	0	107	44,209	23,572	23,124	274	461,799	254,154	500,024	595
nltcs	16	56,719	53,572	452,232	633	326,813	314,528	6,021,330	1,372	2,727,808	2,605,978	82,512,570	2,180
letter	17	16,296	16,227	153,430	678	230,931	230,931	4,948,105	2,726	5,443,620	5,424,968	213,643,716	4,963
msnbc	17	1,288,695	1,288,339	10,564,990	2,695	$\times_t$				$\times_t$			
voting	17	5314	4,364	346	147	72,658	50,004	99,182	413	114,498	106378	421512	3965
zoo	17	1049	704	0	330	12,003	3,875	3,498	539	53,562	47,162	41,698	1695
statlog	19	1,915	1,558	0	212	19,653	12,847	12,403	628	153048	101570	194334	1029
hepatitis	20	18,726	15470	0	4,645	167,033	105,997	105,105	13,223	1,378,039	720,381	1422,924	31,854
imports	22	1023	295	0	150	8,041	2,724	0	404	41,007	21475	0	426
parkinsons	23	8,233	2,732	0	290	32,136	10,189	0	652	151,200	58,802	0	1273

Table 5.6: The BN graph, for enumerating the DAGs in k-best EC: (1) The number of generated nodes. (2) The number of expanded nodes. (3) The number of re-expanded nodes (in partial-expansion A*). (4) The number of times the oracle is invoked to evaluate the heuristic function.

CHAPTER 6

Learning with Ancestral Constraints

In this chapter, we consider ancestral constraints, which are non-decomposable. Ancestral constraints are important practically as they allow one to assert direct or indirect cause-and-effect relationships, or lack thereof, between random variables. Moreover, we show that certain decomposable constraints can be inferred from the ancestral constraints, and can be exploited to tighten the heuristic function. Empirically, our approach is orders of magnitude more efficient than the state-of-the-art, which is based in integer linear programming. This chapter is based in [CSC16].

6.1 Introduction

Bayesian networks learned from data are broadly used for classification, clustering, feature selection, and to determine associations and dependencies between random variables, in addition to discovering causes and effects; see, e.g., [Dar09, KF09, Mur12].

In this chapter, we consider the task of learning Bayesian networks optimally, subject to background knowledge in the form of *ancestral constraints*. Such constraints are important practically as they allow one to assert direct or indirect cause-and-effect relationships (or lack thereof) between random variables. Moreover, one would intuitively expect that their presence should improve the efficiency of the learning process as they reduce the size of the search space. However, nearly all mainstream approaches for optimal structure learning make a fundamental assumption, that the scoring func-

tion is decomposable (i.e., the prior and likelihood are decomposable). This in turn limits their ability to integrate ancestral constraints, which are non-decomposable. In particular, such approaches only support structure-modular constraints such as the presence or absence of edges, or order-modular constraints such as pairwise constraints on topological orderings; see, e.g., [KS04, PK13].

In this chapter, we show how to effectively incorporate non-decomposable constraints into the structure learning approach discussed in earlier chapters, which makes use of an oracle to evaluate the heuristic function in A* search. In particular, we consider learning with ancestral constraints, and inferring decomposable constraints from ancestral constraints to empower the oracle. In principle, structure learning approaches based on integer linear programming (ILP) can also incorporate ancestral constraints (as well as other non-decomposable constraints) [JSG10, BC15].[1] We empirically evaluate the proposed approach against those based on ILP, showing *orders of magnitude* improvements.

This chapter is organized the follows. In Section 6.2, we discuss scoring Bayesian network structure with non-decomposable constraints. In Section 6.3, we provide a review on ancestral constraints. In Section 6.4, we show how to learn optimal Bayesian network structures with ancestral constraints using our EC tree. In Section 6.5, we show how to project ancestral constraints, which are non-decomposable, to infer edge and ordering constraints, which are decomposable. Finally, we evaluate our approach empirically in Section 6.6.

[1] To our knowledge, however, the effectiveness of the ILP approach has not been previously evaluated, in terms of their efficacy in structure learning with ancestral constraints.

6.2 Scoring with Non-Decomposable Constraints

When the background knowledge $\mathcal{P}$ is given in the form of constraints, any Bayesian network structure that violates the constraints is given a score of infinity. That is,

$$\mathsf{score}(G \mid \mathcal{D}, \mathcal{P}) = \begin{cases} \mathsf{score}(G \mid \mathcal{D}), & \text{if } G \text{ satisfies the constraints in } \mathcal{P}. \\ \infty, & \text{otherwise.} \end{cases} \tag{6.1}$$

As discussed in Section 3.3.1, in principle, the BN graph can find an optimal Bayesian network structure, when non-decomposable scores are used, as long as an admissible heuristic function can be designed.

In this chapter, we assume the structure scores with respect to data, i.e., $\mathsf{score}(G \mid \mathcal{D})$, are the typical scores that observe Markov equivalence; BDeu and MDL scores fall into this category. Consequently, more compact search space that assumes Markov-equivalent scores, such as the EC tree discussed in Section 5, can be used to learn an optimal network structure.

Finally, we note that structure learning approaches based on ILP can enforce non-decomposable constraints, when they can be encoded as linear constraints [Cus08]. However, to our knowledge, these approaches have not been evaluated for learning structures with ancestral constraints. We provide such an empirical evaluation in Section 6.6.[2]

6.3 Ancestral Constraints

An ancestral constraint specifies a relation between two variables X and Y in a DAG G. If X is an ancestor of Y, then there is a directed path connecting X to Y in G. If X is not an ancestor of Y, then there is no such path. Ancestral constraints can be used

[2]We also make note of [BT12], which uses ancestral constraints, i.e., path constraints, for *constraint-based* learning methods, such as the PC algorithm. [BT13] further proposes a prior based on path beliefs, i.e., soft constraints, and evaluated using greedy local search.

to express background knowledge on the known causal relations between variables. In diagnosis, for example, we may know that certain diseases, i.e., causes, are ancestors of certain symptoms, i.e., effects.

When X is an ancestor of Y, we have a *positive* ancestral constraint, denoted $X \rightsquigarrow Y$. When X is not an ancestor of Y, we have a *negative* ancestral constraint, denoted $X \not\rightsquigarrow Y$. In this case, there is no directed path from X to Y, but there may still be a directed path from Y to X. Positive ancestral constraints are transitive, i.e., if $X \rightsquigarrow Y$ and $Y \rightsquigarrow Z$ then $X \rightsquigarrow Z$. Negative ancestral constraints are not transitive. For example, in the DAG

$$\boxed{X \rightarrow Z \qquad Y}$$

we have $X \not\rightsquigarrow Y$ and $Y \not\rightsquigarrow Z$ but not $X \not\rightsquigarrow Z$.

Ancestral constraints are *non-decomposable* in the following sense. We cannot generally check whether an ancestral constraint is satisfied or violated by *independently* checking the parents of each variable. To see this, assume the structure scores with respect to data, i.e., $\mathsf{score}(G \mid \mathcal{D})$, are decomposable, and suppose we are searching for an optimal DAG that is compatible with ordering $\langle X_1, X_2, X_3 \rangle$, using the following family scores:

X	$\mathbf{U}$	$\mathsf{score}(X\mathbf{U} \mid \mathcal{D})$
X_1	$\{\}$	1
X_2	$\{\}, \{X_1\}$	1, 2
X_3	$\{\}, \{X_1\}, \{X_2\}, \{X_1, X_2\}$	10, 10, 1, 10

The optimal DAG in this case is

$$\boxed{X_1 \qquad X_2 \rightarrow X_3}$$

If we assert the ancestral constraint $X_1 \rightsquigarrow X_3$, then the optimal DAG is

$$\boxed{X_1 \rightarrow X_2 \rightarrow X_3}$$

Yet, we cannot enforce this ancestral constraints using independent, local constraints on the parents that each variable can take. In particular, the choice of parents for variable X_2 and the choice of parents for variable X_3 will jointly determine whether X_1 is an ancestor of X_3. Hence, the K2 algorithm and approaches based on the order graph cannot enforce ancestral constraints.

These approaches, however, can enforce *decomposable constraints,* such as the presence or absence of an edge $U \rightarrow X$, or a limit on the size of a family $X\mathbf{U}$. Interestingly, one can infer, or *project*, some decomposable constraints from non-decomposable ones. We will discuss this technique extensively, showing how it can lead to significant results.

6.4 EC Trees and Ancestral Constraints

A DAG G satisfies a set of ancestral constraints $\mathcal{A}$, where both are over the same set of variables, if and only if the DAG G satisfies each constraint in $\mathcal{A}$. Moreover, a CPDAG P satisfies $\mathcal{A}$ iff there exists a DAG $G \in \mathsf{class}(P)$ that satisfies $\mathcal{A}$. For example, CPDAG $X_1 - X_2$ and CPDAG $X_2 \quad X_3$ both satisfy the constraint $X_1 \rightsquigarrow X_2$.

We enforce ancestral constraints by pruning a CPDAG node P from an EC tree when P does not satisfy the constraints $\mathcal{A}$. Satisfaction is checked as follows. First, consider an ancestral constraint $X_1 \not\rightsquigarrow X_2$ and a CPDAG P containing a directed path from X_1 to X_2. This CPDAG violates the given constraint, as every structure in $\mathsf{class}(P)$ contains a path from X_1 to X_2. Next, consider an ancestral constraint $X_1 \rightsquigarrow X_2$ and a CPDAG P with no partially directed paths from X_1 to X_2, i.e., paths whose directed edges are oriented towards X_2, such as $X_1 \rightarrow X_3 - X_2$. This CPDAG also violates the given constraint, as no structure in $\mathsf{class}(P)$ contains a path from X_1 to X_2. Given a CPDAG P, we first test for these two cases, which can be done efficiently. If these tests are inconclusive, we exhaustively enumerate the structures of $\mathsf{class}(P)$, to check if any satisfies the given constraints. If not, we can prune P and its descendants

from the EC tree. The soundness of this pruning step is due to the following.

Theorem 6. *In an EC tree, a CPDAG P satisfies ancestral constraints $\mathcal{A}$, both over the same set of variables $\mathbf{X}$, iff its descendants satisfy $\mathcal{A}$.*

6.5 Projecting Constraints

BN graphs and EC trees are expressive enough to support structure learning with non-decomposable constraints such as ancestral constraints. Order graphs, in contrast, do not directly support ancestral constraints, as we discussed previously. However, order graphs support edge constraints that enforce the presence or absence of edges, and pairwise constraints on topological orderings. In this section, we show how one can *project* ancestral constraints onto edge and ordering constraints. That is, we show how one can identify all edge and ordering constraints which are implied by a set of ancestral constraints. The projected constraints are later employed in our experiments to improve the efficiency of structure learning. In particular, recall that our approach to structure learning uses a heuristic function which is obtained from an order-graph system for optimal structure learning (the oracle). We *tighten* this heuristic by passing the projected edge and ordering constraints to the order-graph system, leading to an overall improved efficiency of our structure learning approach.

6.5.1 More on Ancestral Constraints

We first consider some properties of ancestral constraints, which we need for our projection algorithm. We consider constraints on DAGs over a set of nodes that we leave implicit in our notation. We consider both *positive* ancestral constraints $X \rightsquigarrow Y$ and *negative* ancestral constraints $X \not\rightsquigarrow Y$. We let $\mathcal{A}$ denote a set of ancestral constraints, which may include positive and negative constraints. We further let $\mathcal{G}(\mathcal{A})$

denote the set of DAGs G over the variables $\mathbf{X}$ that satisfy all ancestral constraints in the set $\mathcal{A}$.

Given a set of ancestral constraints $\mathcal{A}$, we say that $\mathcal{A}$ entails a positive ancestral constraint $X \rightsquigarrow Y$ iff $\mathcal{G}(\mathcal{A}) \subseteq \mathcal{G}(X \rightsquigarrow Y)$. Equivalently, we have that

$$\mathcal{G}(\mathcal{A}) = \mathcal{G}(\mathcal{A}) \cap \mathcal{G}(X \rightsquigarrow Y) = \mathcal{G}(\mathcal{A} \cup \{X \rightsquigarrow Y\}).$$

A set of ancestral constraints $\mathcal{A}$ can similarly entail negative ancestral constraints $X \not\rightsquigarrow Y$. If the set $\mathcal{A}$ entails another ancestral constraint not in $\mathcal{A}$, we can add that constraint to $\mathcal{A}$ without changing the set of DAGs $\mathcal{G}(\mathcal{A})$ that the constraints represent. Hence, we can obtain a maximal set of ancestral constraints $\mathcal{A}$ found by adding any ancestral constraint entailed by it.

The following lemma characterizes every positive ancestral constraint that can be entailed by a set $\mathcal{A}$.

Lemma 1. *Given a set of ancestral constraints $\mathcal{A}$, then $\mathcal{A}$ entails a positive ancestral constraint $X \rightsquigarrow Y$ iff either $X \rightsquigarrow Y$ is contained in $\mathcal{A}$ or there exists another variable Z such that $\mathcal{A}$ entails both $X \rightsquigarrow Z$ and $Z \rightsquigarrow Y$.*

The above Lemma is based on the transitivity of ancestral relations. Correspondingly, when we start with a set of positive ancestral constraints $\mathcal{A}$, the maximal set that one obtains from $\mathcal{A}$ is the *transitive closure* of $\mathcal{A}$.[3]

The next definition shall simplify the discussions to follow.

Definition 1. *For a given set of ancestral constraints $\mathcal{A}$ and a given variable X, let $\overline{X}$ and $\underline{X}$ denote the set of entailed ancestors and descendants, both including X:*

$$\overline{X} \stackrel{def}{=} \{Z \mid \mathcal{A} \textit{ entails } Z \rightsquigarrow X\} \cup \{X\}$$
$$\underline{X} \stackrel{def}{=} \{Z \mid \mathcal{A} \textit{ entails } X \rightsquigarrow Z\} \cup \{X\}.$$

[3]The transitive closure of a DAG G is another DAG G^+ where there is an edge $X \rightarrow Y$ in G^+ iff X is an ancestor of Y in G.

Further, for variables X and Y, let $\underline{X} \rightsquigarrow \overline{Y}$ and $\overline{X} \rightsquigarrow \underline{Y}$ denote the sets of ancestral relations whose elements would individually imply $X \rightsquigarrow Y$ and $X \not\rightsquigarrow Y$, respectively:

$$\underline{X} \rightsquigarrow \overline{Y} = \stackrel{def}{=} \{X' \rightsquigarrow Y' \mid X' \in \underline{X} \text{ and } Y' \in \overline{Y}\}$$
$$\overline{X} \not\rightsquigarrow \underline{Y} = \stackrel{def}{=} \{X' \not\rightsquigarrow Y' \mid X' \in \overline{X} \text{ and } Y' \in \underline{Y}\}.$$

The following lemma characterizes every negative ancestral constraint that can be entailed by a set $\mathcal{A}$.

Lemma 2. *Given a set of ancestral constraints $\mathcal{A}$, then $\mathcal{A}$ entails a negative ancestral constraint $X \not\rightsquigarrow Y$ iff $\mathcal{A}$ entails a constraint in $\underline{Y} \rightsquigarrow \overline{X}$ or $\overline{X} \not\rightsquigarrow \underline{Y}$.*

Intuitively, this theorem says that an ancestral relation $X \rightsquigarrow Y$ is excluded by a set of ancestral constraints $\mathcal{A}$ iff either (1) it would create a directed cycle through $\underline{Y} \rightsquigarrow \overline{X}$,[4] or (2) it would lead to violating a negative ancestral constraint $\overline{X} \not\rightsquigarrow \underline{Y}$.[5]

Lemmas 1 & 2 can be used to identify a maximal set of ancestral constraints $\mathcal{A}$, e.g., by iteratively searching for new constraints $X \rightsquigarrow Y$ and $X \not\rightsquigarrow Y$, by looking for existing constraints in $\mathcal{A}$ that entail them. As an example, consider DAGs over the variables X, Y and Z, and two initial ancestral constraints $Y \rightsquigarrow Z$ and $X \not\rightsquigarrow Z$. First, we can infer a new negative constraint $Z \not\rightsquigarrow Y$ from $Y \rightsquigarrow Z$ (otherwise, there would be a directed cycle). Next, suppose that $X \rightsquigarrow Y$; since $Y \rightsquigarrow Z$, this contradicts the existing constraint $X \not\rightsquigarrow Z$, hence $X \not\rightsquigarrow Y$. In this case, we can infer two new negative constraints (via Lemma 2), and no new positive constraints, leading to a maximal set of four ancestral constraints.

6.5.2 Edge Constraints

We first consider (decomposable) constraints on the presence of an edge, which we denote by $X \rightarrow Y$, or the absence of an edge, which we denote by $X \not\rightarrow Y$. We

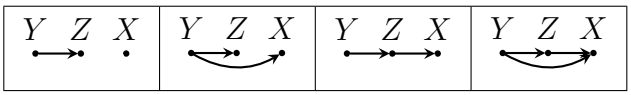
[4] If there is a sequence of constraints $Y \rightsquigarrow Y' \rightsquigarrow X' \rightsquigarrow X$, where $Y' \in \underline{Y}$ and $X' \in \overline{X}$, then another constraint $X \rightsquigarrow Y$ would create a directed cycle.

[5] If we have constraints $X' \rightsquigarrow X$ and $Y \rightsquigarrow Y'$ where $X' \not\rightsquigarrow Y'$, the additional constraint $X \rightsquigarrow Y$ would imply $X' \rightsquigarrow Y'$.

refer to edge presence constraints as *positive* constraints and edge absence constraints as *negative* constraints. We let $\mathcal{E}$ denote a set of edge constraints, which may include positive and negative constraints. We further let $\mathcal{G}(\mathcal{E})$ denote the set of DAGs G that satisfy all edge constraints $\mathcal{E}$. A set of ancestral constraints $\mathcal{A}$ entails a positive edge constraint $X \to Y$ iff $\mathcal{G}(\mathcal{A}) \subseteq \mathcal{G}(X \to Y)$, and it entails a negative edge constraint $X \not\to Y$ iff $\mathcal{G}(\mathcal{A}) \subseteq \mathcal{G}(X \not\to Y)$.

Our goal is to find a maximal set of edge constraints $\mathcal{E}$ that exclude as many DAGs as possible, without excluding any DAGs in $\mathcal{G}(\mathcal{A})$. It suffices to find: (1) all edges $X \to Y$ appearing in all DAGs $G \in \mathcal{G}(\mathcal{A})$ (all positive constraints), and (2) all edges $X \to Y$ that appear in none of the DAGs $G \in \mathcal{G}(\mathcal{A})$ (all negative constraints).

As an example, consider all DAGs over the variables X, Y and Z which satisfy the two ancestral constraints $X \not\to Z$ and $Y \rightsquigarrow Z$. Of the 25 DAGs over three variables, there are four DAGs satisfying these constraints:



First, we note that no DAG above contains the edge $X \to Z$, since this would immediately violate the constraint $X \not\to Z$. Next, no DAG above contains the edge $X \to Y$. Suppose this edge appeared; since $Y \rightsquigarrow Z$, we can infer $X \rightsquigarrow Z$, which contradicts the existing constraint $X \not\to Z$. Hence, we can infer the negative edge constraint $X \not\to Y$. Finally, no DAG above contains the edge $Z \to Y$, since this would lead to a directed cycle with the constraint $Y \rightsquigarrow Z$. The following theorem characterizes the cases where negative edge constraints can be inferred from a set of ancestral constraints.

Theorem 7. *Given a set of ancestral constraints $\mathcal{A}$, then $\mathcal{A}$ entails the negative edge constraint $X \not\to Y$ iff $\mathcal{A}$ entails a constraint in $\overline{X} \not\to \underline{Y}$.*

Intuitively, this theorem says that an edge $X \to Y$ is excluded by a set of ancestral constraints $\mathcal{A}$ iff it would lead to violating a negative ancestral constraint $X \not\to Y$. Note that this also excludes an edge $X \to Y$ that would create a directed cycle through $Y \rightsquigarrow X$ (since $Y \rightsquigarrow X$ already implies $X \not\to Y$, via Lemma 2).

In our earlier example, we note that all DAGs contained the edge $Y \rightarrow Z$. Suppose for contradiction that this edge was not required, and some DAG did not have this edge. The only other possible path from Y to Z must go through X. However, we have the constraint $X \not\rightsquigarrow Z$ and hence no path from Y to Z may go through X. Thus, to satisfy $Y \rightsquigarrow Z$, we must have the edge $Y \rightarrow Z$. The following theorem characterizes the cases where positive edge constraints can be inferred from a set of ancestral constraints.

Theorem 8. *Given a set of ancestral constraints $\mathcal{A}$, then $\mathcal{A}$ entails the positive edge constraint $X \rightarrow Y$ iff $\mathcal{A}$ entails $X \rightsquigarrow Y$ and for all $Z \notin \overline{X} \cup \underline{Y}$, the set $\mathcal{A}$ entails a constraint in either $\overline{X} \not\rightsquigarrow \underline{Z}$ or $\overline{Z} \not\rightsquigarrow \underline{Y}$.*

Intuitively, this theorem says that if X must be an ancestor of Y, then a direct path must connect X to Y (i.e., a positive edge constraint $X \rightarrow Y$) iff there is no other indirect path $X \rightsquigarrow Z \rightsquigarrow Y$ through any other node Z.

6.5.3 Topological Ordering Constraints

We next consider constraints on the topological orderings of a DAG. An ordering satisfies a constraint $X < Y$ iff X appears before Y in the ordering. Further, an ordering constraint $X < Y$ is *compatible* with a DAG G iff there exists a topological ordering of DAG G that satisfies the constraint $X < Y$. The negation of an ordering constraint $X < Y$ is the ordering constraint $Y < X$. A given ordering satisfies either $X < Y$ or $Y < X$, but not both at the same time. A DAG G may be compatible with both $X < Y$ and $Y < X$ through two different topological orderings. We let $\mathcal{O}$ denote a set of ordering constraints. We further let $\mathcal{G}(\mathcal{O})$ denote the set of DAGs G that are compatible with each ordering constraint in $\mathcal{O}$.

Our goal is to infer ordering constraints from ancestral constraints $\mathcal{A}$. In particular, we want a maximal set of ordering constraints $\mathcal{O}$ such that $\mathcal{G}(\mathcal{A}) \subseteq \mathcal{G}(\mathcal{O})$. This task is more subtle than the one for edge constraints.

For example, consider the set of ancestral constraints $\mathcal{A} = \{Z \not\rightsquigarrow Y, X \not\rightsquigarrow Z\}$.

We can infer the ordering constraint $Y < Z$ from the first constraint $Z \not\rightarrow Y$, and $Z < X$ from the second constraint $X \not\rightarrow Z$. Assuming both ordering constraints, we can infer the third ordering constraint $Y < X$, by transitivity. However, consider the following DAG G which satisfies $\mathcal{A}$: $\boxed{X \rightarrow Y \quad Z}$. This DAG is compatible with the constraint $Y < Z$ as well as the constraint $Z < X$, but it is not compatible with the constraint $Y < X$. Consider the three topological orderings of the DAG G: $\langle X, Y, Z\rangle, \langle X, Z, Y\rangle$ and $\langle Z, X, Y\rangle$. The first topological ordering satisfies the first constraint $Y < Z$, and the third ordering satisfies the second constraint $Z < X$. However, none of the orderings satisfy both ordering constraints at the same time. Hence, we cannot justify the inference of the third constraint $Y < X$. More precisely, we cannot infer both ordering constraints $Y < Z$ and $Z < X$ at the same time, as it would eliminate all topological orderings of the above DAG.

Consider another example over variables W, X, Y and Z with a (maximal) set of ancestral constraints: $\mathcal{A} = \{W \not\rightarrow Z, Y \not\rightarrow X\}$. The following DAG G satisfies $\mathcal{A}$: $\boxed{W \rightarrow X \quad Y \rightarrow Z}$. However, inferring the ordering constraints $Z < W$ and $X < Y$ from each ancestral constraint of $\mathcal{A}$ leads to a cycle in the above DAG: $W < X < Y < Z < W$.

Roughly, for a set of ancestral constraints $\mathcal{A}$ to imply a set of ordering constraints $\mathcal{O}$, only enough ordering constraints $X < Y$ are allowed to be inferred from ancestral constraints $Y \not\rightarrow X$, so long as the ordering constraints do not induce a cycle. This idea is formalized in the following result.

Theorem 9. *Let $\mathcal{A}$ be a maximal set of ancestral constraints, and let $\mathcal{O}$ be a closed set of ordering constraints. The set $\mathcal{O}$ is entailed by $\mathcal{A}$ if the constraints of $\mathcal{O}$ satisfy the following two conditions:*

1. *if $X < Y$ is in $\mathcal{O}$ then $Y \not\rightarrow X$ is in $\mathcal{A}$*
2. *if $X < Y$ and $Z < W$ are in $\mathcal{O}$, where X, Y, Z and W are distinct, then at least one of the following constraints are in $\mathcal{A}$:*

$$X \rightsquigarrow Y \qquad Z \rightsquigarrow W$$
$$X \rightsquigarrow Z \qquad Y \rightsquigarrow W$$
$$X \rightsquigarrow W \qquad Y \not\rightsquigarrow Z$$

Theorem 9 implies a greedy (heuristic) procedure for finding a set of ordering constraints $\mathcal{O}$ entailed by ancestral constraints $\mathcal{A}$: add a constraint $X < Y$ for any $Y \not\rightsquigarrow X \in \mathcal{A}$, as long as it introduces no cycles, such as $W < X < Y < Z < W$. In the Appendix, we show to find an *optimal* set of ordering constraints $\mathcal{O}$ by reduction to (partial) MaxSAT. This MaxSAT based approach, for projecting ancestral constraints onto ordering constraints, is evaluated in the next section.

6.6 Experiments

We now empirically evaluate the efficacy of our approach to learning with ancestral constraints. We start with several standard Bayesian network benchmarks: ALARM, ANDES, CHILD, CPCS54, and HEPAR2.[6] We simulated different structure learning problems from these networks, by (1) taking a random sub-network $\mathcal{N}$ of a given size,[7] (2) simulating a training dataset from $\mathcal{N}$ of varying sizes, and (3) simulating a set of ancestral constraints of a given size, by randomly selecting ordered pairs (X, Y), whose ground-truth ancestral relations in $\mathcal{N}$ were used as constraints. In our experiments, we varied the number of variables in the learning problem (n), the size of the training dataset (N), and the percentage of the $\frac{n(n-1)}{2}$ total ancestral relations that were given as constraints (p). We report results that were averaged over 50 different datasets: 5 datasets were simulated from each of 2 different sub-networks, which were taken from each of the 5 original networks mentioned above. Our experiments were run on a 2.67GHz Intel Xeon X5650 CPU. We assumed BDeu scores with an equivalent sample

[6] The networks used in our experiments are available for download at `http://www.bnlearn.com/bnrepository`

[7] We select random sets of nodes, and all of their ancestors, until we obtain a connected sub-network of a given size.

$n = 10$						
N	512		2048		8192	
p	EC TREE	GOBNILP	EC TREE	GOBNILP	EC TREE	GOBNILP
0	0.003 ± 0.001	7.808 ± 3.791	0.003 ± 0.001	112.982 ± 8.513	0.003 ± 0.001	19.702 ± 14.419
0.01	0.003 ± 0.001	9.608 ± 4.463	0.003 ± 0.001	15.408 ± 9.816	0.003 ± 0.001	23.579 ± 16.896
0.05	0.003 ± 0.002	11.558 ± 3.437	0.002 ± 0.001	14.541 ± 7.224	0.002 ± 0.001	19.853 ± 12.67
0.1	0.005 ± 0.008	10.742 ± 2.978	0.001 ± 0.001	11.601 ± 4.649	0.001 ± 4 E-4	13.873 ± 6.96
0.25	0.014 ± 0.034	4.035 ± 2.026	0.002 ± 0.004	3.426 ± 1.691	3 E-4 ± 2 E-4	3.367 ± 1.832
0.5	0.004 ± 0.009	0.866 ± 0.535	0.001 ± 0.001	0.705 ± 0.343	1 E-4 ± 4 E-5	0.718 ± 0.342
0.75	3 E-4 ± 4 E-4	0.314 ± 0.144	1 E-4 ± 8 E-5	0.746 ± 0.352	1 E-4 ± 4 E-5	0.304 ± 0.116
1	9 E-5 ± 9 E-6	0.205 ± 0.045	9 E-5 ± 8 E-6	0.310 ± 0.117	9 E-5 ± 3 E-5	0.207 ± 0.053
$n = 12$						
N	512		2048		8192	
p	EC TREE	GOBNILP	EC TREE	GOBNILP	EC TREE	GOBNILP
0	0.014 ± 0.007	70.852 ± 30.442	0.015 ± 0.006	98.280 ± 50.842	0.017 ± 0.006	144.214 ± 97.053
0.01	0.010 ± 0.003	73.392 ± 32.285	0.011 ± 0.003	99.460 ± 45.628	0.012 ± 0.002	145.750 ± 98.051
0.05	0.016 ± 0.031	60.157 ± 19.169	0.008 ± 0.006	75.397 ± 29.952	0.274 ± 1.857	95.107 ± 46.408
0.1	0.213 ± 1.282	52.016 ± 11.817	0.098 ± 0.623	53.291 ± 14.243	0.356 ± 2.560	59.415 ± 21.809
0.25	4.910 ± 22.067	22.468 ± 8.335	0.175 ± 0.564	20.880 ± 7.276	0.166 ± 0.618	19.677 ± 6.618
0.5	0.510 ± 2.207	6.106 ± 4.267	0.029 ± 0.083	6.101 ± 4.470	0.008 ± 0.023	5.847 ± 4.319
0.75	0.002 ± 0.004	2.661 ± 0.733	2 E-4 ± 0.001	2.618 ± 0.610	1 E-4 ± 9 E-5	2.570 ± 0.663
1	1 E-4 ± 2 E-5	2.291 ± 0.235	1 E-4 ± 2 E-5	2.302 ± 0.289	1 E-4 ± 2 E-5	2.272 ± 0.256
$n = 14$						
N	512		2048		8192	
p	EC TREE	GOBNILP	EC TREE	GOBNILP	EC TREE	GOBNILP
0	0.058 ± 0.039	625.081 ± 178.822	0.065 ± 0.012	839.460 ± 369.335	0.088 ± 0.024	1349.239 ± 986.919
0.01	0.049 ± 0.031	673.003 ± 196.424	0.064 ± 0.036	901.497 ± 426.888	0.077 ± 0.040	1356.628 ± 885.504
0.05	0.078 ± 0.090	243.681 ± 82.484	0.053 ± 0.052	287.449 ± 137.948	0.039 ± 0.024	411.219 ± 271.415
0.1	0.583 ± 2.098	176.500 ± 39.837	1.263 ± 5.843	198.176 ± 69.237	0.029 ± 0.078	218.935 ± 102.884
0.25	55.074 ± 268.997	126.312 ± 42.324	0.905 ± 4.288	112.800 ± 24.027	0.022 ± 0.129	107.435 ± 23.663
0.5	0.483 ± 1.300	73.236 ± 47.896	0.020 ± 0.056	67.291 ± 32.449	0.002 ± 0.008	62.602 ± 32.178
0.75	0.004 ± 0.009	44.074 ± 8.713	0.001 ± 0.001	42.948 ± 8.951	2 E-4 ± 0.001	41.207 ± 7.977
1	2 E-4 ± 6 E-5	39.484 ± 3.736	2 E-4 ± 7 E-5	39.662 ± 4.621	2 E-4 ± 5 E-5	37.775 ± 4.045

Table 6.1: Time t (in seconds) used by the EC tree and and GOBNILP to find optimal Bayesian networks. n is the number of variables, N is the size of the dataset, and p is the percentage of the ancestor constraints.

$n = 16$									
N	512			2048			8192		
p	t	s	Δ	t	s	Δ	t	s	Δ
0	0.486 ± 0.365	1	12.26	0.582 ± 0.256	1	7.26	0.733 ± 0.326	1	4.64
0.01	0.299 ± 0.196	1	12.56	0.409 ± 0.229	1	7.52	0.544 ± 0.321	1	5.18
0.05	1.709 ± 5.554	1	11.06	0.410 ± 0.695	1	6.42	0.189 ± 0.135	1	3.96
0.1	143.066 ± 784.804	0.98	9.63	4.800 ± 20.404	1	5.08	0.733 ± 4.389	1	2.90
0.25	279.377 ± 1076.302	0.92	6.48	6.428 ± 38.239	0.94	3.17	5.157 ± 30.363	1	0.94
0.5	114.624 ± 455.318	0.98	4.51	27.201 ± 133.496	1	1.80	0.028 ± 0.131	1	0.40
0.75	0.027 ± 0.079	1	2.16	0.001 ± 0.003	1	0.88	0.001 ± 0.001	1	0.20
1	2 E-4 ± 7 E-5	1	0.84	3 E-4 ± 1 E-4	1	0.60	3 E-4 ± 6 E-5	1	0.20
$n = 18$									
N	512			2048			8192		
p	t	s	Δ	t	s	Δ	t	s	Δ
0	2.250 ± 1.302	1	16.74	2.780 ± 2.067	1	8.32	3.111 ± 1.601	1	7.06
0.01	2.216 ± 2.145	1	16.58	3.458 ± 2.619	1	8.60	3.628 ± 3.190	1	7.38
0.05	41.146 ± 216.095	0.96	15.02	2.912 ± 7.939	0.98	6.96	2.116 ± 6.292	1	5.56
0.1	149.404 ± 311.223	0.94	12.72	73.029 ± 337.478	0.96	5.81	7.353 ± 34.795	1	3.78
0.25	251.735 ± 630.021	0.78	6.33	338.098 ± 870.234	0.94	3.79	30.897 ± 192.365	0.96	1.96
0.5	95.179 ± 381.632	0.98	5.49	13.917 ± 42.572	0.98	2.69	116.288 ± 797.095	0.98	1.24
0.75	9.068 ± 63.205	1	3.30	5.826 ± 40.530	1	1.66	0.721 ± 4.928	1	0.72
1	4 E-4 ± 1 E-4	1	0.72	4 E-4 ± 1 E-4	1	0.48	4 E-4 ± 1 E-4	1	0.26
$n = 20$									
N	512			2048			8192		
p	t	s	Δ	t	s	Δ	t	s	Δ
0	19.403 ± 24.539	1	23.44	20.615 ± 13.667	1	10.60	28.218 ± 20.990	1	7.22
0.01	30.378 ± 50.539	1	23.67	30.458 ± 41.297	1	10.53	34.340 ± 37.932	1	7.09
0.05	87.74 ± 250.173	0.96	18.44	39.252 ± 151.223	1	8.20	17.401 ± 56.342	1	5.00
0.1	492.588 ± 1131.446	0.82	14.67	185.823 ± 778.689	0.94	7.21	24.458 ± 113.15	0.98	3.94
0.25	507.019 ± 1223.195	0.58	6.17	572.676 ± 1553.382	0.88	4.46	153.811 ± 770.801	0.96	2.28
0.5	163.191 ± 378.992	0.88	6.36	46.425 ± 179.856	0.96	2.19	70.153 ± 282.016	1	1.07
0.75	1.471 ± 9.206	1	4.49	0.284 ± 1.685	1	1.36	0.375 ± 2.403	1	0.60
1	0.001 ± 3 E-4	1	2.02	0.001 ± 3 E-4	1	0.47	0.001 ± 3 E-4	1	0.18

Table 6.2: Time t (in seconds) used by the EC tree to find optimal Bayesian networks, with a 32G limit on memory, and a 2 hour limit on running time. n is the number of variables, N is the size of the dataset, p is the percentage of the ancestor constraints, s is the percentage of test cases that finishes, and Δ is the edge difference of the optimal networks learned and the true networks.

	N	512		2048		8192	
$n = 10$	p	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP
	0.01	0.003 ± 0.001 1	7.631 ± 3.502	0.003 ± 0.001 1	11.251 ± 5.688	0.733 ± 0.326 1	15.879 ± 11.497
	0.05	0.008 ± 0.011 1	9.465 ± 2.797	0.004 ± 0.003 1	11.780 ± 5.337	0.003 ± 0.001 1	15.448 ± 8.652
	0.10	0.112 ± 0.395 1	9.697 ± 2.736	0.007 ± 0.010 1	9.925 ± 3.288	0.004 ± 0.002 1	11.872 ± 5.221
	0.25	0.446 ± 0.824 1	3.456 ± 1.839	0.086 ± 0.394 1	2.952 ± 1.221	0.005 ± 0.006 1	2.862 ± 1.032
	0.50	0.699 ± 1.539 1	0.835 ± 0.408	0.053 ± 0.139 1	0.767 ± 0.312	0.009 ± 0.022 1	0.753 ± 0.300
	0.75	0.336 ± 0.747 1	0.402 ± 0.120	0.026 ± 0.056 1	0.398 ± 0.108	0.008 ± 0.014 1	0.399 ± 0.110
	1.00	0.222 ± 0.537 1	0.309 ± 0.023	0.017 ± 0.028 1	0.307 ± 0.023	0.005 ± 0.002 1	0.309 ± 0.019
	N	512		2048		8192	
$n = 12$	p	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP
	0.01	0.013 ± 0.007 1	63.532 ± 28.977	0.014 ± 0.005 1	83.588 ± 41.358	0.015 ± 0.004 1	128.234 ± 89.292
	0.05	0.064 ± 0.175 1	55.202 ± 17.445	0.031 ± 0.073 1	70.198 ± 32.189	1.175 ± 8.040 1	90.588 ± 46.001
	0.10	2.535 ± 11.087 1	50.334 ± 10.436	2.363 ± 10.371 1	52.798 ± 12.934	0.910 ± 5.989 1	57.663 ± 17.712
	0.25	70.186 ± 222.557 0.98	23.292 ± 7.55	4.571 ± 12.315 1	20.736 ± 6.283	1.626 ± 4.92 1	21.155 ± 6.862
	0.50	137.308 ± 367.122 1	7.741 ± 4.463	15.53 ± 73.340 1	7.801 ± 5.255	1.434 ± 4.844 1	7.360 ± 4.087
	0.75	21.858 ± 61.294 1	4.384 ± 0.650	1.731 ± 5.051 1	4.394 ± 0.746	0.496 ± 1.630 1	4.296 ± 0.607
	1.00	2.305 ± 3.932 1	4.095 ± 0.225	0.349 ± 0.695 1	4.067 ± 0.243	0.151 ± 0.308 1	4.024 ± 0.226
	N	512		2048		8192	
$n = 14$	p	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP	EC TREE (t/s)	GOBNILP
	0.01	0.013 ± 0.007 1	634.192 ± 179.048	0.123 ± 0.128 1	738.254 ± 281.88	0.120 ± 0.071 1	1295.898 ± 1012.983
	0.05	0.064 ± 0.175 1	228.565 ± 74.187	0.868 ± 2.921 1	276.678 ± 134.768	0.176 ± 0.138 1	404.350 ± 268.617
	0.10	2.535 ± 11.087 1	174.669 ± 37.257	34.979 ± 143.439 0.98	183.926 ± 40.567	0.603 ± 2.09 1	210.124 ± 87.371
	0.25	280.588 ± 853.874 0.84	137.670 ± 28.532	88.802 ± 293.489 1	126.794 ± 18.796	1.851 ± 7.157 1	126.242 ± 24.151
	0.50	609.175 ± 1562.083 0.88	90.915 ± 47.224	35.62 ± 108.154 1	85.575 ± 34.392	4.739 ± 16.845 1	83.808 ± 30.041
	0.75	258.797 ± 756.941 1	64.509 ± 7.748	6.489 ± 14.54 1	63.683 ± 7.925	2.275 ± 11.555 1	64.044 ± 7.804
	1.00	21.176 ± 34.265 1	61.438 ± 5.052	1.385 ± 1.918 1	60.55 ± 4.445	0.536 ± 0.727 1	61.062 ± 5.944

Table 6.3: Time and standard deviation $t \pm \sigma$ (in seconds) used by the EC tree and GOBNILP to find optimal structures, without any projected constraints, using a 32G limit on memory, and a 2 hour limit on time. n is the number of variables, N is the size of the dataset, p is the percentage of ancestral constraints, and s is the percentage of test cases that finishes.

size of 1. We further pre-computed the scores of candidate parent sets, which were fed as input into each system evaluated.

In our first set of experiments, we compared our approach with the prominent ILP-based system of GOBNILP,[8] where we encoded ancestral constraints using linear constraints (details are given in the supplementary Appendix); we further supplied the ILP with the constraints inferred from constraint projection (we shall revisit this subject later). We remark that while in principle, approaches such as ILP can be used to encode ancestral relations [Cus08], they have not previously been evaluated as a constraint used in score-based structure learning (to our knowledge). Table 6.1 summarizes our results. Our approach, which navigates the EC tree and which empowers our structure-learning oracle with projected constraints, is consistently orders-of-magnitude faster than GOBNILP, for all values of n, N and p that we varied. This difference increased as the number of variables n is increased. We remark that approaches based on heuristic search (such as ours) have been observed to scale better than ILP-based approaches when no limits are placed on the sizes of families (as was done here); see, e.g., [YM13, MKJ14].

Next, we evaluate (1) the effect that introducing ancestral constraints have on the efficiency of search, and (2) the scalability of our approach as the number of variables in the learning problem increases. In Table 6.2, we report results where we varied the number of variables $n \in \{16, 18, 20\}$, and where we set a 2 hour time limit and a 32GB memory limit. First, there appears to be an easy-hard-easy trend as we increase the proportion p of ancestral constraints to include in the learning problem. When this proportion p is small, then the learning problem is similar to the unconstrained problem, and our oracle can produce an accurate estimate. When this proportion p is large, then the problem is heavily constrained, and the (valid) search space is also significantly reduced. In contrast, the ILP approach more consistently became easier as more constraints were provided (from Table 6.1). As would be expected, the learning problem becomes more challenging with an increasing number of variables n, and with

[8] www.cs.york.ac.uk/aig/sw/gobnilp

smaller training set sizes N. We remark that our approach scales to $n = 20$ variables here, which is comparable to the scalability of modern score-based approaches reported in the literature (for BDeu scores); e.g., [YM13] reported results up to 26 variables (for the case of BDeu scores).

Table 6.2 also reports the average structural Hamming distance Δ between the learned network and the ground-truth network used to generate the data. We see that as the dataset size N and the proportion p of constraints available increases, the more accurate the learned model becomes.[9] We remark that a relatively small number of ancestral constraints (say 10%–25%) can have a similar impact on the quality of the observed network (relative to the ground-truth), as increasing the amount of data available from 512 to 2048, or from 2048 to 8192. This highlights the impact that background knowledge can have, in contrast to collecting more (potentially expensive) training data.

Next, we consider the effect of projecting ancestral constraints onto edge and ordering constraints. In Table 6.3, we performed structure learning in the EC tree and with ILP, but *without* projecting any of the constraints, in contrast to Table 6.1. When learning with the EC tree, the projection of constraints has a significant impact on the efficiency of learning (often several orders of magnitude). When learning with ILPs, there is some mild overhead with smaller numbers of variables ($n = 10, 12$), but with a larger number of variables ($n = 14$), there were consistent improvements when projected constraints are used.

[9]Note that Δ can be greater than 0 when $p = 1$, since there may be many DAGs that respect the same set of ancestral constraints. For example, the DAG $X \to Y \to Z$ expresses the same ancestral relations, after we add the edge $X \to Z$.

CHAPTER 7

Asymptotic Analysis on Structures Learned with Order-Modular Priors

In this chapter, we consider large random Bayesian network structures generated by the uniform order-modular prior, but where each node has relatively few parents. We analyze the asymptotic, expected properties of these random Bayesian network structures, such as the expected number of parents a node may have, and the expected size of the Markov blanket. This chapter is based in [CP14].

7.1 Introduction

In this chapter, we propose a particular model for the distribution of large random Bayesian networks, that is, DAGs where the edges appear at random, each node has at most k parents, and k is much less than the number of nodes n. Using this model and basic combinatorics tools, we can provide a set of characterizations of a random Bayesian network structure. We estimate the expected size of a Bayesian network. In addition, we estimate the expected size of the Markov blanket and a minimal d-separator, where both can render the variables inside a Bayesian network independent, and therefore simplify problems such as inference in Bayesian networks [Dar09, KF09, Mur12]. More specifically, we show that the expected size of a Bayesian network is $O\big(kn - k^2 \ln n\big)$, and the expected size of the Markov blanket is $O\big(k^2 - k^3 \ln n/n\big)$. We also show that the maximum size of a minimal d-separator is k; and that on the average, a revised version of the algorithm for finding a minimal d-separator by [AD96]

and [TPP98] has time complexity upper-bounded by $O\big(k^2(k^3+1)n - k^2 \ln n\big)$. We discuss further estimates in Section 7.4-7.6. In addition, we compare our estimates against Bayesian networks for analyzing gene expression data learned from the Gene Expression Omnibus database [EDL02, FLN00]. The results suggest that this model provides a useful characterization of the Bayesian networks.

The Bayesian network distribution of our model has been used as a prior for Bayesian network learning, known as the order-modular prior [FK00]. To the best of our knowledge, the mathematical properties of the model have yet to receive much attention. Most work on random graphs is about undirected graphs [Bol01, JLR00, New09]. In addition, a node in Bayesian networks usually has a limited number of parents [IC02]. The studies on random DAGs have not considered DAGs with this property [BE84, CRS03, DMS01, KN09, PT01].

This chapter is organized as follows. In Session 7.2, we propose our model for the distribution of large random Bayesian networks. In Section 7.3, we provide a mathematical background. In Section 7.4, we consider the basic properties of a random Bayesian network structure. In Section 7.5, we consider the moral graph and the Markov Blankets of a random Bayesian network structure. In Section 7.7, we consider isolated nodes in a random Bayesian network structure. Finally, we evaluate our random Bayesian network model against real-world Bayesian networks in Section 7.8.

7.2 Radnom Bayesian Network

We consider random DAGs generated by the following model.[1]

Definition 2. *(Random Bayesian Network)*
Given a set of nodes V, where $|V| = n$, and an integer k, let $G_{n,k}$ denote the random Bayesian network *for V where each node has at most k parents. Then let L denote a list of DAGs constructed as follows: Consider all possible orderings $(V, <)$ of the nodes in*

[1] We note that the uniform order-modular prior is considered for random graph generation by [BE84].

V. *For each ordering* $(V, <)$*, every node* v *has zero to* k *parents* v'*, where* $v' < v$*. Then* $P(G_{n,k} = G) = n_G/|L|$*, where* n_G *is the number of times that* G *appears in* L*.*

For example, consider $V = \{X, Y, Z\}$ and $k = 1$. In Definition 2, when $(V, <) = \langle X, Y, Z\rangle$ or $\langle X, Z, Y\rangle$, the DAGs constructed are

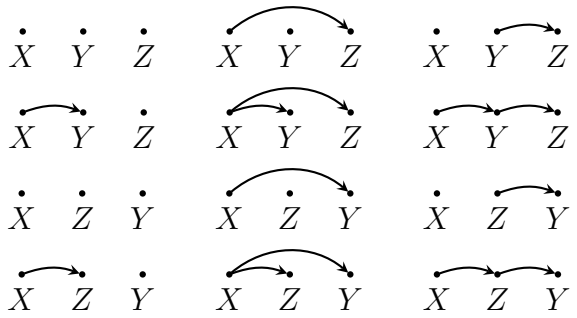


Figure 7.1: Given $V = \{X, Y, Z\}$ and $k = 1$, when $(V, <) = \langle X, Y, Z\rangle$ or $\langle X, Z, Y\rangle$, the DAGs constructed in Definition 2.

Note that there are six possible $(V, <)$. Let G be the edgeless DAG, that is, $X\ Y\ Z$. Since G appears once for each $(V, <)$, and each $(V, <)$ induces six possible DAGs, $P(G_{n,k} = G) = 1/6$. Then consider the DAG $G' : X \to Y \to Z$. Since G' only appears once when $(V, <) = \langle X, Y, Z\rangle$, $P(G_{n,k} = G') = 1/36$.

From the example above, it appears that $G_{n,k}$ is likely to contain isolated nodes. Section 7.7 shows that when n is large, $G_{n,k}$ contains no isolated nodes almost surely.

In addition, note that for a graph G in L, since there exists a $(V, <)$ such that the parents are always smaller than the children, G does not contain cycles, and is therefore a DAG. Moreover, since all possible $(V, <)$ are considered, L contains all the possible DAG configurations for $G_{n,k}$.

Now we discuss some fundamental properties of our random Bayesian network model. Given a $(V, <)$, consider distinct nodes v, v', pa, ch where $pa < v, v' < ch$. Recall that each node has at most k parents. Consequently if v is already a parent of ch, then v' is less likely to be a parent of ch. However since the constraint is only on the

number of parents, whether pa is a parent of v is independent of whether pa is a parent of v'. We state the properties formally as the following theorems.

Theorem 10. *(Children Dependence)*
Given a random Bayesian network $G_{n,k}$ and an ordering $(V, <)$, for distinct nodes v, v', ch where $v, v' < ch$, the event that whether ch is a child of v and the event that whether ch is a child of v' are dependent.

Theorem 11. *(Parent Independence)*
Given a random Bayesian network $G_{n,k}$ and an ordering $(V, <)$, for distinct nodes v, v', pa where $pa < v, v'$, the event that whether pa is a parent of v and the event that whether pa is a parent of v' are independent.

7.3 Mathematical Tools

Now we discuss the combinatorics tools that we use to analyze our random Bayesian network model.

First we show the absorption and extraction identities of the binomial coefficients. Let i be a non-negative integer. Then

$$(i+1)\binom{n}{i+1} = n\binom{n-1}{i} \tag{7.1}$$

$$(n-i)\binom{n}{i} = n\binom{n-1}{i}. \tag{7.2}$$

Then we consider some fundamental tools in probabilistic combinatorics. Let $I_1, \ldots, I_i$ be random variables and I_n be a non-negative random variable that depends on n. Then

$$E[I_1 + \ldots + I_i] = E[I_1] + \ldots + E[I_i]. \tag{7.3}$$

$$\text{If } \lim_{n\to\infty} E[I_n] \to 0, \text{ then } I_n \text{ is zero almost surely.} \tag{7.4}$$

Equation (7.3) is known as the linearity of expectation. Note that $I_1, \ldots, I_i$ may be dependent. Consequently (7.3) is a valuable tool for tackling problems involving dependent events. Equation (7.4) can be intuitively understood as follows: Since random

variable I_n is non-negative and its expected value approaches zero, it is almost always zero [AS08].

In addition, we use integral to approximate summation [GKO94]. Let $f(r)$ be a smooth function defined for all reals r in $[m, n]$, then

$$\sum_{i=m}^{n} f(i) \approx \int_{m}^{n} f(r)\, dr.$$

In the following, when integral is used to approximate $\sum_{i=m}^{n} f(r)$ to a real number r', we write $\sum_{i=m}^{n} f(r) \sim r'$. In addition, approximations made without explanation are based on the fact that $k \ll n$.

Moreover, let I_A denote the indicator random variable of event A, that is, $I_A = 1$ when A occurs, and $I_A = 0$ when A does not. Note that $E[I_A] = P(A)$.

7.4 Basic Properties

In this section, we study the probability that a given edge occurs, the expected Bayesian network size, and the expected number of parents for a node in our random Bayesian network model.

Recall that the definition of our model involves all possible $(V, <)$. Consider mapping each $(V, <)$ to integers between 1 and n. For example, for $(V, <) = \langle X, Y, Z \rangle$, map X, Y, Z to $1, 2, 3$. The mapping preserves the node ordering, and therefore can be used to represent $(V, <)$.

7.4.1 Edge Probability

In this subsection, we consider the probability that there exists edge $X \to Y$ for given X, Y in $G_{n,k}$. Let A denote the event that there exists $X \to Y$, and let O denote the event that in $(V, <)$, X, Y map to integers x, y. Note that the edge exists only if $x < y$.

Consequently

$$P(A) = \sum_{1 \le x < y \le n} P(A|O)P(O).$$

When $x < k$, Y does not have k nodes to serves as parents. Since $n \gg k$, we ignore these cases. Hence

$$P(A) \approx \sum_{k \le x < y \le n} P(A|O)P(O).$$

Clearly $P(O) = 1/\left(n(n-1)\right)$. Then note that $P(A|O)$ is the ratio of the number of DAGs where X is a parent of Y to the number of DAGs where X may or may not be a parent of Y when given $X < Y$. By Theorem 10 and 11, this ratio is the same as the ratio of possible parents of Y when X is a parent of Y to possible parents of Y when X may or may not be a parent of Y. When X is a parent, there are $y-2$ nodes, other than X, smaller than Y and may be parents of Y. Consequently there are $\sum_{i=0}^{k-1} \binom{y-2}{i}$ ways to choose parents of Y. When X may or may not be a parent, there are $\sum_{i=0}^{k} \binom{y-1}{i}$ ways to choose parents of Y. Hence

$$P(A|O) = \frac{\sum_{i=0}^{k-1} \binom{y-2}{i}}{\sum_{i=0}^{k} \binom{y-1}{i}}$$

A partial summation of the binomial coefficients such as the numerator and denominator in the equation above does not have a closed form [GKO94]. Therefore we now simplify this equation. Let $S_1 = \sum_{i=0}^{k-1} \binom{y-2}{i}$ and $S_2 = \sum_{i=0}^{k} \binom{y-1}{i}$. Then

$$\begin{aligned}
S_1 &= \frac{1}{y-1} \sum_{i=0}^{k-1} (i+1) \binom{y-1}{i+1} \big(\text{by (7.1)}\big) \\
&= \frac{1}{y-1} \sum_{i=0}^{k} i \binom{y}{i} \qquad (7.5) \\
S_1 &= \frac{1}{y-1} \sum_{i=0}^{k} (y-1-i) \binom{y-1}{i} - \binom{y-2}{k} \big(\text{by (7.2)}\big) \\
&= S_2 - S_1 - \binom{y-2}{k} \big(\text{by (7.5)}\big).
\end{aligned}$$

Consequently

$$P(A|O) = \frac{1}{2}\left(1 - \frac{\binom{y-2}{k}}{\sum_{i=0}^{k}\binom{y-1}{i}}\right).$$

Since when y is not small, $\binom{y-1}{k} + \binom{y-1}{k-1} = \binom{y}{k}$ are dominant in $\sum_{i=0}^{k}\binom{y-1}{i}$, we approximate $\sum_{i=0}^{k}\binom{y-1}{i}$ with $\binom{y}{k}$. Similarly, we approximate $y-1$ with y. As a result,

$$P(A|O) \approx \frac{k}{y} - \frac{k(k+1)}{2y^2}. \tag{7.6}$$

Then by using integral to approximate summation,

$$P(A) \sim \frac{k}{n} - \frac{k(k+3)\ln n}{2n^2}.$$

7.4.2 Expected Size and Number of Parents

Since $E[I_A]$ is the expected number of times $X \to Y$ exists for given X, Y, by (7.3), summing $E[I_A]$ over all possible X, Y gives the expected size of $G_{n,k}$:

$$\begin{aligned}\sum_{X,Y\in V} E[I_A] &= \sum_{X,Y\in V} P(A) \\ &\approx kn - \frac{k(k+3)\ln n}{2}.\end{aligned}$$

Then since each edge induces exactly one parent and there are n nodes, the expected number of parents a node has is approximately

$$k - \frac{k(k+3)\ln n}{2n}.$$

7.5 Moral Graph and Markov Blanket Analysis

In this section, we apply the analysis used in Section 7.4 to moral graphs and Markov blankets. We first provide a review on Markov blanket and the related d-separation concepts.

7.5.1 d-separation and Markov Blankets

In a Bayesian network, if two random variables X and Y are d-separated given another set of variables, known as the d-*separator*, $\mathbf{Z}$, then they are conditionally independent given variables $\mathbf{Z}$. More specifically, d-separation can be determined through graphical tests that involve checking if certain types of paths that involve X, Y and $\mathbf{Z}$ exist; see [Dar09, KF09, Mur12]. Given variable X, its Markov blanket is a particular d-separator $\mathbf{Z}$, that consists of all the parents, children, and children's parents of X, such that given $\mathbf{Z}$, variable X is d-separation from any other variable Y.

7.5.2 Moral Graph

In this subsection, we first consider the v-structure for given parents X, Y and child Z, that is, $X \to Z \leftarrow Y$ where there are no edges between X and Y. Then we consider the expected size increase of moralizing a Bayesian network.

Let B denote the event that there exists $X \to Z \leftarrow Y$ and there do not exist edges between X and Y in $G_{n,k}$. Let B_1 denote the event that there exists $X \to Z \leftarrow Y$, and let B_2 denote the event that there do not exist edges between X and Y. Let O denote the event that in $(V, <)$, X, Y, Z map to x, y, z. By Theorem 11, B_1 and B_2 are independent given O. Consequently

$$P(B) \approx \sum_{\substack{k \le x < y < z \le n \\ k \le y < x < z \le n}} P(B_1|O)P(B_2|O)P(O).$$

Consider when $k \le x < y < z \le n$. Note that edges between X and Y can only be $X \to Y$. Then similar to Subsection 7.4.1,

$$P(B_1|O) = \frac{\sum_{i=0}^{k-2} \binom{z-3}{i}}{\sum_{i=0}^{k} \binom{z-1}{i}} \approx \frac{k(k-1)}{4z^2}\left(3 - \frac{2(k+1)}{z}\right)$$
$$P(B_2|O) = 1 - \frac{\sum_{i=0}^{k-1} \binom{y-2}{i}}{\sum_{i=0}^{k} \binom{y-1}{i}} \approx 1 - \frac{k}{y} + \frac{k(k+1)}{2y^2}.$$

Please see the Appendix for details. When $k \le y < x < z \le n$, the case is essentially

identical. As a result,

$$P(B) \sim \frac{3k(k-1)}{4n^2} - \frac{k(k-1)(7k+1)\ln n}{2n^3}.$$

Each v-structure induces exactly one edge to be added in moralization. Consequently the expected number of edges added when moralizing $G_{n,k}$ is

$$\sum_{\substack{\{X,Y\}\subset V \\ Z\in V}} E[I_B] \approx \frac{3k(k-1)n}{8} - \frac{k(k-1)(7k+1)\ln n}{4}.$$

7.5.3 Markov Blanket

The Markov blanket of a node consists of its parents, spouses, and children. Recall Subsection 7.4.2. Note that since each edge induces a parent and a child, the expected number of parents is identical to the expected number of children. In addition, since each v-structure induces two spouses, following Subsection 7.5.2, the expected number of spouses is approximately

$$\frac{3k(k-1)}{4} - \frac{k(k-1)(7k+1)\ln n}{2n}.$$

Then the expected size of the Markov blanket is approximately

$$\frac{k(3k+5)}{4} - \frac{k(7k^2-5k+2)\ln n}{2n}.$$

7.6 d-Separator Analysis

In this section, we consider the problem of finding a minimal d-separator for given X, Y, that is, a d-separator such that none of its subsets d-separates X and Y. We study the maximum size of a minimal d-separator, and the time needed to find one.

7.6.1 Maximum Size

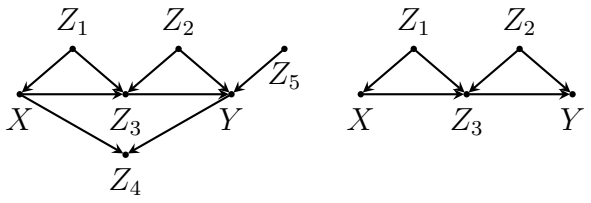
If there exists a d-separator for X and Y, at least one of the following two sets d-separates X and Y: the set consisting of the parents of X, and the set consisting of the

parents of Y [Pea88]. As a result, the maximum size of a minimal d-separator is k.

7.6.2 Complexity

We first review some results on this problem by [AD96] and [TPP98]. Let G be a DAG in $G_{n,k}$, and let G_A denote the subgraph induced by the ancestral set of $X \bigcup Y$. A minimal d-separator of X and Y only needs to d-separate them in G_A, and can be found by running two breadth-first searches on $(G_A)^m$, where m denotes the moral graph.

The results can be improved by exploiting the fact that a d-separator of X and Y only needs to d-separate them in G'_A, the subgraph of G_A consisting of paths between X and Y, as shown in Figure 7.2. Consequently a minimal d-separator can be found by (1) construct G'_A, in $O(|G_A|)$ time (2) run two breadth-first searches on $(G'_A)^m$, in $O\left(|(G'_A)^m|\right)$ time. Then since G'_A is a subgraph of G_A, the time complexity for finding a minimal d-separator $O(|G_A| + |(G'_A)^m|) = O(|G_A| + |E_m|)$, where E_m is the edges added in G'_A moralization.



(a) DAG G. (b) DAG G'_A.

Figure 7.2: A DAG G and its G'_A.

Below, we show that the expected time complexity for finding a minimal d-separator in $G_{n,k}$, that is, $O(|G_A| + |E_m|)$, is asymptotically upper bounded by $k^3(k-1)^2 n/20 + k^2 n/4 - k^2 \ln n$.

7.6.2.1 Expected Size of G_A

Let Z, Z' be two nodes in $(V, <)$ where $Z < Z'$, and let S denote the event that edge $Z \to Z'$ is in G_A. Then

$$E[|G_A|] = \sum_{\{Z,Z'\} \subset V} E[I_S].$$

Let A denote the event that Z, Z' are ancestors of X in G, and let B denote that for Y. Let O denote the event that in $(V, <)$, X, Y, Z, Z' map to x, y, z, z'. Note that an event may not hold for some $(V, <)$, such as A when $x < z < z' < y$. Then

$$\begin{aligned} E[I_S] < & \sum_{\substack{1 \le y < z < z' < x \le n \\ 1 \le z < y < z' < x \le n \\ 1 \le z < z' < x < y \le n \\ 1 \le z < z' < y < x \le n}} E[I_A|O]P(O) \\ + & \sum_{\substack{1 \le x < z < z' < y \le n \\ 1 \le z < x < z' < y \le n \\ 1 \le z < z' < x < y \le n \\ 1 \le z < z' < y < x \le n}} E[I_B|O]P(O). \end{aligned}$$

First note that $\sum E[I_A|O]P(O) = \sum P(A|O)P(O)$. Then since event A states there exists path $Z \to Z' \to \ldots \to X$, consider such a path for a $(V, <)$, as below:

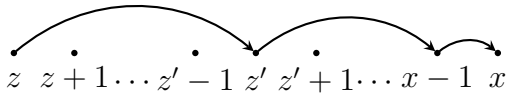


Figure 7.3: Path $Z \to Z' \to \ldots \to X$ for a $(V, <)$.

Note that the edges on the path appear independently. By (7.6), we approximate the probability that there exists $Z \to Z'$ with k/z', and that for $W_i \to X$ with k/x, where W_i are nodes between Z' and X. Then let p_i be the probability that there exist paths from Z' to W_i. Since the events that there exists $W_i \to X$ are dependent, and the events that there exist paths from Z' to W_i may also be dependent,

$$P(A|O) < \frac{k^2}{xz'}(p_1 + \ldots + p_{x-z'-1}) < \frac{k^2(x - z' - 1)}{xz'}.$$

As a result,

$$\sum_{\substack{1\leq y<z<z'<x\leq n\\ 1\leq z<y<z'<x\leq n\\ 1\leq z<z'<x<y\leq n\\ 1\leq z<z'<y<x\leq n}} E[I_A|O]P(O) < \frac{k^2}{4n} - \frac{k^2 \ln n}{n^2}.$$

The second term is essentially the same. Consequently

$$E[|G_A|] < \frac{k^2 n}{4} - k^2 \ln n.$$

7.6.2.2 Expected Size of E_m

Let Z, Z' be two nodes in $(V, <)$ where $Z < Z'$, and let S' denote the event that edge $Z - Z'$ is added when moralizing G'_A. Then

$$E[|E_m|] = \sum_{\{Z,Z'\}\subset V} E[I_{S'}].$$

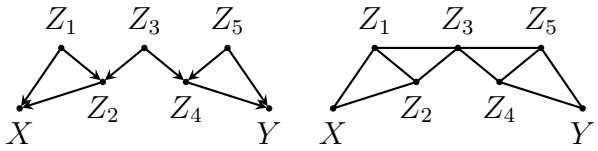
Edge $Z - Z'$ is added when there does not exist $Z \to Z'$, and there exist paths of the following types: (a) $X \to \ldots \to Z \to Z'' \leftarrow Z' \to \ldots \to Y$, where Z'' is an ancestor of Y (b) $X \leftarrow \ldots \leftarrow Z \to Z'' \leftarrow Z' \leftarrow \ldots \leftarrow Y$, where Z'' is an ancestor of X (c) $X \leftarrow \ldots \leftarrow Z \to Z'' \leftarrow Z' \to \ldots \to Y$, where Z'' is an ancestor of X or Y, or both.

To see this, first note that in (a) Z'' cannot be an ancestor of X or otherwise G contains a loop. Then consider path $X \leftarrow Z_1 \to Z_2 \leftarrow Z_3 \to Z_4 \leftarrow Z_5 \to Y$ in Figure 7.4(a). It appears to induce two edges $Z_1 - Z_3$ and $Z_3 - Z_5$ to be added, and yet is not any of the three types. Nevertheless, since in G'_A, nodes other than X and Y are ancestors of X or Y, there exist paths of the three types that induce $Z_1 - Z_3$ and $Z_3 - Z_5$, which in this case are $X \leftarrow Z_1 \to Z_2 \leftarrow Z_3 \to Z_4 \to Y$ and $X \leftarrow Z_2 \leftarrow Z_3 \to Z_4 \leftarrow Z_5 \to Y$.

By reasoning similar to that for $E[|G_A|]$,

$$E[|E_m|] < \frac{431k^3(k-1)^2 n}{9216}.$$

Please see the Appendix for details.



(a) G'_A. (b) $(G'_A)^m$.

Figure 7.4: DAG G'_A and its $(G'_A)^m$.

7.7 Isolated Nodes

In this section, we prove that $G_{n,k}$ contains no isolated nodes almost surely, that is, with probability 1.

Let A denote the event that a given node X is isolated, that is, in $(V, <)$, none of the nodes smaller than X is a parent of X, and none of the nodes greater than X is a child of X. Let A_1 denote the event that none of the nodes smaller than X is its parent, and let A_2 denote the event that none of the nodes greater than X is its child. Let O denote the event that in $(V, <)$, X maps to x. By Theorem 11, A_1 and A_2 are independent given O. Hence similar to Subsection 7.4.1,

$$P(A) \approx \sum_{x=k+1}^{n} P(A_1|O)P(A_2|O)P(O)$$
$$P(A_1|O) = \frac{1}{\sum_{i=0}^{k} \binom{x-1}{i}}.$$

Then by Theorem 11, whether a node greater than X is a child of X is independent of whether another such node is a child of X. By (7.6), we approximate the probability that there exists $X \to Y$ with k/y. Hence

$$P(A_2|O) \approx \prod_{i=x+1}^{n} 1 - \frac{k}{i}.$$

Then

$$P(A) < \frac{1}{n} \sum_{x=k+1}^{n} \frac{\prod_{i=x+1}^{n} 1 - \frac{k}{i}}{\binom{x}{k}} < \frac{1}{n} \sum_{x=k+1}^{n} \frac{k!}{(n-k+1)^k}$$
$$\sim \frac{k!}{(n-k+1)^k}.$$

As a result, the expected number of nodes that are isolated

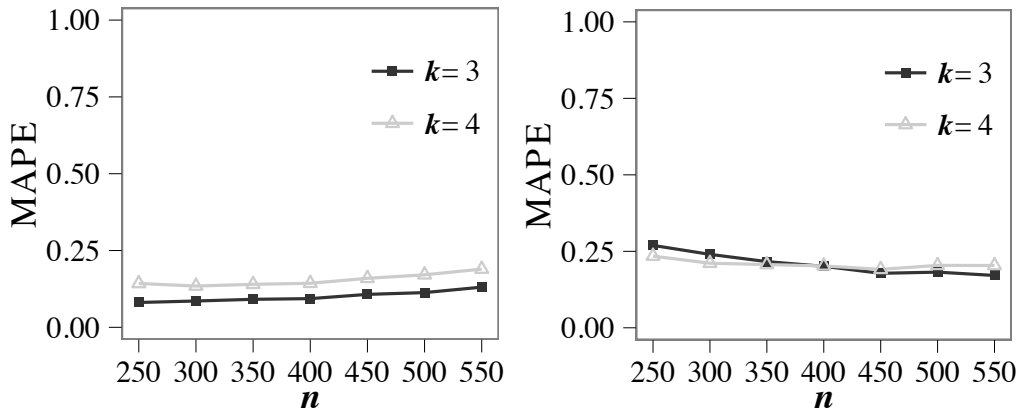
$$\sum_{X \in V} E[I_A] < \frac{k!n}{(n-k+1)^k}.$$

Since $k > 1$, this number approaches zero when $n \to \infty$. Hence by (7.4), $G_{n,k}$ contains no isolated nodes almost surely.

7.8 Experiments

In this section, we compare our estimates of the expected size and the expected size increase of moralization against Bayesian networks learned from real data.

We use Bayesian network learning software Banjo [Har05] and the Gene Expression Omnibus database [EDL02] to learn Bayesian networks for analyzing gene expression data [FLN00]. Banjo learns Bayesian network structures with the Bayesian Dirichlet scoring metric for a given maximum number of parents k. We learned about 30 Bayesian networks for $n = 250, 300, 350, 400, 450, 500, 550$ and $k = 3, 4$ respectively with an equivalent sample size that grows with k. Figure 7.5 shows the mean absolute percentage error (MAPE) of the estimates.



(a) The expected size. (b) The expected moralization size increase.

Figure 7.5: MAPE of the expected size and the expected size increase of moralization estimates.

The MAPE for the expected size is lower than 0.15 when $k = 3$, and lower than 0.2 when $k = 4$, though both slightly increase with n. The MAPE for the expected size increase for $n = 250$ is 0.27 when $k = 3$, and 0.23 when $k = 4$; and for $n = 550$ the MAPE drops to 0.17 when $k = 3$, and 0.20 when $k = 4$. The MAPEs suggest that our model provides a useful characterization of Bayesian networks with important applications. They also suggest that though our estimates are asymptotic ones, they may be applied to relatively small Bayesian networks.

CHAPTER 8

Conclusion

In this thesis, we proposed a general framework for learning Bayesian network structures from the given data and background knowledge that supports the use of non-decomposable scores. Our framework can be viewed as a generalization of the existing search-based approach to structure learning, but with decomposable scores, where we proposed a much more expressive search space, called the *BN graph*, that accommodates non-decomposable scores. We showed that the BN graph, in spite of its size, can be explored efficiently by the A* search, by using a highly accurate heuristic function that can be evaluated by existing approaches for structure learning with decomposable scores.

Using our framework, we developed the first system to learn an optimal Bayesian network structure with the order-modular prior, scaling up to 17 variables in real-world datasets. We also developed a system for learning an optimal network structure with ancestral constraints, that is empirically orders of magnitude faster than the state-of-the-art, integer-linear-programming based approach in real-world datasets. We also showed that ancestral constraints, though non-decomposable, imply decomposable constraints that can be used to improve the efficiency of both our framework and the integer linear programming approach.

In addition, we used our framework to enumerate the k-best Bayesian network structures, and the k-best Markov-equivalent Bayesian network structures, using decomposable scores. In both cases, our framework is empirically orders-of-magnitude more efficient in real-world dataset, compared to the state-of-the-art, dynamic-programming-

based approaches. More specifically, for enumerating the k-best Markov-equivalent Bayesian network structures, we specialized the BN graph into a more compact search space, and then further canonized the search space, so that the search space is of the form of a tree, where each search node is reached through exactly one path.

APPENDIX A

Proofs of Theorems in Chapter 4

Theorem 1.

$$\begin{aligned} h(G_i) &= \min_{G_n : G_i \rightsquigarrow G_n} \sum_{X\mathbf{U} \in G_n - G_i} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}) + \min_{G_n : G_i \rightsquigarrow G_n} -\log \frac{Pr_n(G_n)}{Pr_i(G_i)} \\ &\leq \min_{G_n : G_i \rightsquigarrow G_n} \Big(\sum_{X\mathbf{U} \in G_n - G_i} \mathsf{score}(X\mathbf{U} \mid \mathcal{D}) - \log \frac{Pr_n(G_n)}{Pr_i(G_i)} \Big) \\ &= \min_{G_n : G_i \rightsquigarrow G_n} g(G_n) - g(G_i) \end{aligned}$$

Since heuristic function h lower-bounds the true cost of to a goal node, it is admissible.

□

Below we consider the correctness of Algorithm 1.

Lemma 3. *In Algorithm 1:*

1. *all G_i that generate G_{i+1} are extracted from H before G_{i+1} is extracted;*
2. *when $(G_{i+1}, f_{i+1}, l_{i+1})$ is extracted from H,*

$$f_{i+1} = \mathsf{score}(G_{i+1} | \mathcal{D}) - \log \#G_{i+1} + h(G_{i+1}),$$

and l_{i+1} is the number of linear extensions of G_{i+1}, i.e., $\#G_{i+1}$.

where $h(G_i) = h_1(G_i) - \sum_{j=i+1}^{n} \log j$.

Proof. Consider a minimum item $(G_{i+1}, f_{i+1}, l_{i+1})$ extracted from H. Below we show by induction that (1) all G_i such that G_i generates G_{i+1} are extracted from H before

G_{i+1} (2) $f_{i+1} = \mathsf{score}(G_{i+1}|\mathcal{D}) - \log \#G_{i+1} + h(G_{i+1})$, and l_{i+1} is the number of linear extensions of G_{i+1}, which is also the number of paths from G_0 to G_{i+1}.

For $i = 0$, clearly (1) and (2) are true. Assume (1) and (2) are true for all (G_i, f_i, l_i). Then when $(G_{i+1}, f_{i+1}, l_{i+1})$ is extracted, l_{i+1} is the number of paths from G_0 to G_{i+1} that pass the G_i extracted from H before G_{i+1}. To see this, note that l is the number of path from G_0 to G_i. Moreover, since l_{i+1} is the number paths from G_0 to G_{i+1} that pass the G_i extracted from H before G_{i+1}, when $(G_{i+1}, f_{i+1}, l_{i+1})$ is in H,

$$f_{i+1} = \mathsf{score}(G_{i+1}|\mathcal{D}) - \log \sum_{G_i \prec G_{i+1}} N(G_0 \to \ldots \to G_i \to G_{i+1}) + h(G_{i+1}),$$

where $G_i \prec G_{i+1}$ denotes that G_i is extracted before G_{i+1}, and $N(G_0 \to \ldots \to G_i \to G_{i+1})$ denotes the number of paths $G_0 \to \ldots \to G_i \to G_{i+1}$. Note that f_{i+1} decreases as more G_i are extracted.

Consider when (G_i, f_i, l_i) and $(G_{i+1}, f_{i+1}, l_{i+1})$ are both in H. Below we show that all G_i that generates G_{i+1} are extracted from H before G_{i+1}. Consider

$$\begin{aligned} f_i &= \mathsf{score}(G_i|\mathcal{D}) \\ &\quad - \log \sum_{G_{i-1} \prec G_i} N(G_0 \to \ldots \to G_{i-1} \to G_i) + h_1(G_i) - \sum_{j=i+1}^{n} \log j \\ f_{i+1} &= \mathsf{score}(G_{i+1}|\mathcal{D}) \\ &\quad - \log \sum_{G_i' \prec G_{i+1}} N(G_0 \to \ldots \to G_i' \to G_{i+1}) + h_1(G_{i+1}) - \sum_{j=i+2}^{n} \log j \end{aligned}$$

We simply need to show $f_{i+1} > f_i$. Since h_1 is a consistent heuristic function for learning with score, $\mathsf{score}(G_{i+1}|\mathcal{D}) + h_1(G_{i+1}) \geq \mathsf{score}(G_i|\mathcal{D}) + h_1(G_i)$. Then we only need to show

$$\begin{aligned} &(i+1) \sum_{G_{i-1} \prec G_i} N(G_0 \to \ldots \to G_{i-1} \to G_i) \\ &> \sum_{G_i' \prec G_{i+1}} N(G_0 \to \ldots \to G_i' \to G_{i+1}) \end{aligned}$$

First, for any pair of DAGs G_i and G'_i that can generate a DAG G_{i+1}, there exists a unique DAG G_{i-1} that can generate both G_i and G'_i. For each G'_i on the right-hand side, there thus exists a corresponding (and unique) G_{i-1} on the left-hand side that can generate both G'_i and G_i. Further, since G'_i was expanded, G_{i-1} must also have been expanded (by induction). For each such G_{i-1}, if G'_i has a linear extension count of L, then G_{i-1} must have at least a linear extension count of L/i, and hence the corresponding $N(G_0 \to \ldots \to G_{i-1} \to G_i)$ is at least L/i. On the left-hand side, we the corresponding term is thus at least $(i+1) \cdot L/i > L$. Since this holds for each element of the summation on the right-hand side, the above inequality holds.

Since all G_i that generates G_{i+1} are extracted from H before G_{i+1}, as a result, $f_{i+1} = \mathsf{score}(G_{i+1}|\mathcal{D}) - \log \#G_{i+1} + h(G_{i+1})$, and l_{i+1} is the number of all paths from G_0 to G_{i+1}. □

of Theorem 3. To see the correctness of the algorithm, simply note that by Lemma 3, when $(G_{i+1}, f_{i+1}, l_{i+1})$ is extracted from H, i.e. the open list, $f_{i+1} = f(G_{i+1})$. □

of Theorem 2. By Lemma 3, Algorithm 1 can count the number of linear extensions. □

APPENDIX B

Weighted Linear Extensions and the A* Search

Now we consider general order-modular priors:

$$Pr(G) = \sum_{\pi \sim G} \prod_{X\mathbf{U}} \rho_X(X\mathbf{U}) \cdot \sigma_X(\pi_{:X}) - \log C, \tag{B.1}$$

where π is an ordering of variables; $\pi \sim G$ indicates that ordering π is compatible with DAG G; $\pi_{:X}$ represents the set of variables in the prefix of π ending with variable X; ρ_X and σ_X are non-negative scoring functions; and C is a normalizing constant.[1] The uniform order-modular prior arises when $\rho_X = 1$ and $\sigma_X = 1$ for all X. When the scoring functions ρ_X and σ_X are not equal to 1, $Pr(G)$ corresponds to a *weighted count* of linear extensions, where each count is weighted by $\prod_{X\mathbf{U}} \rho_X(X\mathbf{U})\sigma_X(\pi_{:X})$. Consequently, ideas for the uniform order-modular prior can be used for general order-modular priors.

The heuristic function h'_2 in Theorem 2 can be generalized by changing a lower bound of the linear extension count to a lower bound on the weighted linear extension count:

$$h'_2(G_i) = -\sum_{k=i+1}^{n} \log(\rho_{\max} \cdot \sigma_{\max} \cdot k) \tag{B.2}$$

where $\rho_{\max}$ is the maximum of $\rho_X(X\mathbf{U})$, and $\sigma_{\max}$ is the maximum of $\sigma_X(\pi_{:X})$. Algorithm 1 can be generalized for order-modular priors, which is given in Algorithm 4.

Theorem 12. *Algorithm 4 learns an optimal Bayesian network with an order-modular prior.*

[1] For our purposes, we can ignore the normalizing constant of the prior, since it is independent of the DAG G that we want to optimize; we make this assumption for the remainder of the section.

Algorithm 4: A* search for learning an optimal BN with order-modular priors $Pr(G)$.

Data: A dataset $\mathcal{D}$ over variables $\mathbf{X}$.

Result: An optimal Bayesian network structure G maximizing

$$Pr(G \mid \mathcal{D}) \propto \sum_{X\mathbf{U}} \mathsf{score}(X\mathbf{U}|\mathcal{D}) - \log Pr(G).$$

begin

- $H \leftarrow$ min-heap with only $(G_0, f(G_0), 1)$, where the heap is ordered by f
- **while** $H \neq \emptyset$ **do**
  - extract the minimum item (G, f, l) from H
  - **if** $V(G) = \mathbf{X}$ **then** return G
  - **foreach** G' obtained by adding a leaf X to G **do**
    - let w be $l \cdot \rho_X(X\mathbf{U}) \cdot \sigma_X(V(G))$
    - **if** G' is not in H **then**
      - insert into H: $(G', \mathsf{score}(G'|\mathcal{D}) - \log w + h(G'), w)$
    - **else**
      - let (G', f', l') be in H, decrease f' by $\log \frac{l'+w}{l'}$, increase l' by w; and
      - reheapify

Proof. Recall the proof of Algorithm 1, to show the correctness of Algorithm 4, we simply need to generalize Lemma 3 for order-modular prior, where l_{i+1} is now a weighted count of linear extensions. Recall the proof of Lemma 3, and let $N_{\rho\sigma}(G_0 \to \ldots \to G_{i-1} \to G_i)$ denote a weighted count of paths $G_0 \to \ldots \to G_{i-1} \to G_i$. Then we only need to show

$$\rho_{\max}\sigma_{\max}(i+1) \sum_{G_{i-1} \prec G_i} N_{\rho\sigma}(G_0 \to \ldots \to G_{i-1} \to G_i)$$
$$> \sum_{G'_i \prec G_{i+1}} N_{\rho\sigma}(G_0 \to \ldots \to G'_i \to G_{i+1})$$

To see this, first note that by the inductive hypothesis, all the G'_{i-1} that generate G'_i are extracted. Moreover, since G_i and G'_i both generate G_{i+1}, among all G'_{i-1}, there exists at least one G^*_{i-1} that generates G_i. Then note that when the weighted count of

linear extensions of G'_i is L, the weighted count of linear extensions of G^*_{i-1} is at least $L/(\rho_{\max}\sigma_{\max}i)$. $\square$

APPENDIX C

Proofs of Theorems in Chapter 5

Proof of Theorem 5

First recall the following theorem from [VP92, Mee95].

Theorem 13. *Given a PDAG where all the v-structures are oriented, then the CPDAG can be obtained by repeatedly applying the following orientation rules:*

R_1 *Orient $b - c$ into $b \to c$ whenever there is an arrow $a \to b$ such that a and c are nonadjacent.*

R_2 *Orient $a - b$ into $a \to b$ whenever there is chain $a \to c \to b$.*

R_3 *Orient $a - b$ into $a \to b$ whenever there are two chains $a - c \to b$ and $a - d \to b$ such that c and d are nonadjacent.*

R_4 *Orient $a - b$ into $a \to b$ whenever there are two chains $a - c \to d$ and $c \to d \to b$ such that c and b are nonadjacent and a and d are nonadjacent.*

Proof of Theorem 5. (If:) First, note that there are no compelled edges directed away from X (which would make X a non-leaf). Next, note that we can compel each edge $S - X$ towards X one-by-one, each time checking if Theorem 13 compels another edge in $S' - X$ towards S' instead. If this never happens, then all edges $S - X$ can be oriented towards X (which makes X a possible leaf).

We start by orienting one edge $S - X$ towards S. Consider each of the rules in Theorem 13:

R_1 cannot be used to orient the edge from $X \rightarrow S$. First, if there were some other compelled edge $Y \rightarrow X$ where $Y \notin \mathbf{S}$ and Y is not adjacent to S, then the edge $S - X$ should already have been compelled (i.e., P was not a CPDAG). Otherwise, we only orient edges from $S \rightarrow X$ and all $S \in \mathbf{S}$ are adjacent.

R_2 cannot be used to orient the edge from $X \rightarrow S$, since there is no chain from X to S (which would imply X was a non-leaf).

R_3 cannot be used to orient the edge from $X \rightarrow S$, since all potential neighbors c and d via an unoriented edge must be adjancent.

R_4 cannot be used to orient the edge from $X \rightarrow S$, since all potential neighbors c and b via an unoriented edge must be adjancent.

Since no rule compels us to orient an edge away from X, we can orient the edges one-by-one to make X a leaf.

(Only if:) We show that if $\mathbf{S}$ is not a clique, then X is not a leaf. If $\mathbf{S}$ is not a clique, then there is a pair S and S' in $\mathbf{S}$ that are non-adjacent. If we orient $S - X$ as $S \leftarrow X$, then X is not a leaf. If we orient $S - X$ as $S \rightarrow X$, then by Theorem 13, Rule R_1, we must orient $S' - X$ as $S' \leftarrow X$. Hence, X is not a leaf. □

Proofs of Propositions

Proof of Proposition 1. Follows from the definition of equivalence classes. □

Proof of Proposition 2. This path can be constructed by following any path to G in the BN graph, and then identifying the corresponding CPDAGs in the EC graph. □

Proof of Proposition 3. Follows from Theorem 10 in [Chi95]. □

Proof of Proposition 4. Follows from the fact that the canonical ordering is the ordering with the largest reverse lexicographic order. □

APPENDIX D

Proofs of Theorems in Chapter 6

D.1 ILP with Ancestral Constraints

Here, we review an encoding of ancestral relations in ILP. This is an adaptation (and extension) of the MaxSAT encoding described in [Cus08].

First, we have transitivity of positive ancestral constraints:

$$\neg A(X,Y) + \neg A(Y,Z) + A(X,Z) \geq 1$$

For ancestor and family variables: we let edge/ancestor variables $E(X,Y,Z)$ be 0-1 variables that denote $X \rightsquigarrow Y \rightarrow Z$. We have

$$\neg E(X,Y,Z) + A(X,Z) \geq 1$$
$$\neg A(X,Z) + \sum_{Y} E(X,Y,Z) \geq 1$$

and

$$\neg E(X,Y,Z) + A(X,Y) + I(Z,\mathbf{U}) \geq 1$$
$$\neg A(X,Y) + \neg I(Z,\mathbf{U}) + E(X,Y,Z) \geq 1$$

where $Y \in \mathbf{U}$.

D.2 Inferring Orderings via MAXSAT

We can encode Theorem 9 as a (partial) MaxSAT problem to find an optimal set $\mathcal{O}$, where the number of ordering constraints added is maximized. First, we have the hard constraints:

1. **transitivity of ordering constraints**: for all $X < Y$ and $Y < Z$ in $\mathcal{O}$:

$$(X < Y) \wedge (Y < Z) \Rightarrow (X < Z)$$

2. **ordering constraint only if negative ancestral constraint:** for all $X < Y$ in $\mathcal{O}$:

$$(X < Y) \Rightarrow (Y \not\rightsquigarrow X)$$

3. **acyclicity:** for all $X < Y$ and $Z < W$ in $\mathcal{O}$:

$$\begin{aligned}&(X < Y) \wedge (Z < W)\\ &\Rightarrow (X \rightsquigarrow Y) \vee (X \rightsquigarrow Z) \vee (X \rightsquigarrow W) \vee (Z \rightsquigarrow W) \vee (Y \rightsquigarrow W) \vee (Y \not\rightsquigarrow Z)\end{aligned}$$

Second, we have the soft constraints:

4. **candidate ordering constraint:** for all $X \not\rightsquigarrow Y$ in $\mathcal{A}$

$$(X \not\rightsquigarrow Y) \Rightarrow (Y < X)$$

The optimal (partial) MaxSAT solution maximizes the number of soft constraints satisfied. The above soft constraint is satisfied when $X \not\rightsquigarrow Y \notin \mathcal{A}$ (the number of such clauses satisfied in this manner is fixed by the input set $\mathcal{A}$). The above soft constraint is otherwise satisfied when $X \not\rightsquigarrow Y \in \mathcal{A}$ and $Y < X \in \mathcal{O}$. Since $Y < X \in \mathcal{O}$ only if $X \not\rightsquigarrow Y \in \mathcal{A}$ (via the corresponding hard constraint), the partial MaxSAT solution maximizes the number of ordering constraints $Y < X$ in $\mathcal{O}$.

In our experiments, we used the EVASOLVER partial MaxSAT solver, available at `http://www.maxsat.udl.cat/14/solvers/eva500a__`.

D.3 Proof of Theorem 6

Lemma 4. *In a EC tree, if a CPDAG with variables X, Y does not satisfy $X \rightsquigarrow Y$, its descendants do not satisfy $X \rightsquigarrow Y$ either.*

Proof. Assume the contrary, i.e., consider CPDAG P_i that does not satisfy $X \not\rightsquigarrow Y$, and $P_i \to \ldots \to P_{k-1} \xrightarrow{X_k \mathbf{U}_k} P_k \to \ldots \to P_j$, where P_k is the first CPDAG from P_i to P_j that satisfies $X \rightsquigarrow Y$, i.e., there exists a DAG G_k in $\mathsf{class}(P_k)$ with path $X \to \ldots \to Z \to X_k \to W \to \ldots \to Y$. By the EC tree edge generation rules, G_k also contains edge $Z \to W$. Then consider DAG G_{k-1} in $\mathsf{class}(P_{k-1})$, constructed by removing X_k from G_k. Since G_{k-1} contains $X \to \ldots \to Z \to W \to \ldots \to Y$, P_k is not the first CPDAG with a DAG satisfying $X \rightsquigarrow Y$. □

Lemma 5. *In a EC tree, if a CPDAG with variables X, Y satisfies $X \rightsquigarrow Y$, its descendants also satisfy $X \rightsquigarrow Y$.*

Proof. Follows directly from the fact that variables are added as leaves to all the DAGs in a CPDAG. □

Theorem 6 follows directly from the two lemmas.

D.4 Proofs for Section 6.5.1

Proof of Lemma 1. This lemma is simply based on the transitivity of positive ancestral relations. □

Before we prove Lemma 2, we first consider a few lemmas.

Lemma 6. *If a constraint in $\underline{Y} \rightsquigarrow \overline{X}$ is entailed by $\mathcal{A}$, then $Y \rightsquigarrow X$ is entailed by $\mathcal{A}$. If a constraint in $\overline{X} \not\rightsquigarrow \underline{Y}$ is entailed by $\mathcal{A}$, then $X \not\rightsquigarrow Y$ is entailed by $\mathcal{A}$.*

Proof of Lemma 6. If $Y' \in \underline{Y}$ and $X' \in \overline{X}$, then $Y \rightsquigarrow Y'$ and $X' \rightsquigarrow X$. Hence, if $Y' \rightsquigarrow X'$ then $Y \rightsquigarrow X$.

If $X' \in \overline{X}$ and $Y' \in \underline{Y}$, then $X' \rightsquigarrow X$ and $Y \rightsquigarrow Y'$. Hence, if $X' \not\rightsquigarrow Y'$ then $X \not\rightsquigarrow Y$. □

Lemma 7. *If $X \rightsquigarrow Y$ is entailed by $\mathcal{A}$, then $Y \not\rightarrow X$ is entailed by $\mathcal{A}$. Conversely, if $Y \not\rightarrow X$ is not entailed by $\mathcal{A}$, then $X \rightsquigarrow Y$ is not entailed by $\mathcal{A}$.*

Proof of Lemma 7. Suppose for contradiction that $Y \not\rightarrow X$ is not entailed by $\mathcal{A}$, and there exists a DAG $G \in \mathcal{G}(\mathcal{A})$ where $Y \rightsquigarrow X$. Since $X \rightsquigarrow Y$ is entailed by $\mathcal{A}$, then DAG G must have a directed cycle, which is a contradiction. □

Proof of Lemma 2. First, if $\mathcal{A}$ entails a constraint in $\underline{Y} \rightsquigarrow \overline{X}$, then $\mathcal{A}$ entails $Y \rightsquigarrow X$, by Lemma 6. Next, if $\mathcal{A}$ entails $Y \rightsquigarrow X$ then $\mathcal{A}$ entails $X \not\rightarrow Y$, by Lemma 7. Finally, $\mathcal{A}$ entails a constraint in $\overline{X} \not\rightarrow \underline{Y}$, by Definition 1.

It thus suffices to show that $\mathcal{A}$ entails $X \not\rightarrow Y$ iff $\mathcal{A}$ entails a constraint in $\overline{X} \not\rightarrow \underline{Y}$. The only if direction follows by Definition 1, and the if direction follows by Lemma 6. □

D.5 Proofs for Section 6.5.2

Lemma 8. *For a given set of ancestral constraints $\mathcal{A}$, if there is a DAG $G \in \mathcal{G}(\mathcal{A})$ satisfying the ancestral constraint $X \rightsquigarrow Y$, then $\mathcal{G}(\mathcal{A})$ also contains a DAG with the additional edge $X \rightarrow Y$.*

Proof of Lemma 8. Let G be a DAG in $\mathcal{G}(\mathcal{A})$ where there is a path from X to Y. Let G' be the DAG where we add the edge $X \rightarrow Y$ to G. First, G' satisfies any positive ancestral constraint in $\mathcal{A}$, since any such constraint satisfied by G remains satisfied in G'. Second, G' satisfies any negative ancestral constraint in $\mathcal{A}$, since adding an edge $X \rightarrow Y$ introduces no new ancestral relations that were not already present in G, since G already satisfies $X \rightsquigarrow Y$. Thus, G' must also be in $\mathcal{G}(\mathcal{A})$ □

Proof of Theorem 7. ($\Leftarrow$) If $\overline{X} \not\rightarrow \underline{Y}$, then by Lemma 6, $X \not\rightarrow Y$, which in turn implies $X \not\rightarrow Y$.

($\Rightarrow$) Suppose for contradiction that no constraint in $\overline{X} \not\rightsquigarrow \underline{Y}$ is entailed by $\mathcal{A}$. In this case, there exists a DAG $G \in \mathcal{G}(\mathcal{A})$ where $X \rightsquigarrow Y$. By Lemma 8, there is also a DAG $G' \in \mathcal{G}(\mathcal{A})$ with the edge $X \to Y$, which is a contradiction. Hence, $\mathcal{A}$ entails a constraint in $\overline{X} \not\rightsquigarrow \underline{Y}$. $\square$

Lemma 9. *Given a set of ancestral constraints $\mathcal{A}$, if $\mathcal{G}(\mathcal{A})$ contains a DAG satisfying $X \rightsquigarrow Z$, and a DAG satisfying $Z \rightsquigarrow Y$, but it does not contain a DAG satisfying both $X \rightsquigarrow Z$ and $Z \rightsquigarrow Y$, then $\mathcal{A}$ entails $X \not\rightsquigarrow Y$.*

Proof. First, if $\mathcal{A}$ entails $Y \rightsquigarrow X$, then by Lemma 7, $\mathcal{A}$ entails $X \not\rightsquigarrow Y$. Assume instead that $\mathcal{A}$ does not entail $Y \rightsquigarrow X$.

Let $\mathcal{A}_+ = \mathcal{A} \cup \{X \rightsquigarrow Z\}$. Let $\overline{X}_+$ and $\underline{X}_+$ denote the entailed ancestors and entailed descendents of X with respect to $\mathcal{A}_+$. Note that:

$$\overline{X}_+ = \overline{X} \qquad \text{and} \qquad \overline{Z}_+ = \overline{Z} \cup \overline{X}.$$

Further, since $\mathcal{A}$ does not entail $Y \rightsquigarrow X$, adding $X \rightsquigarrow Z$ does not introduce new entailed descendants of Y, so:

$$\underline{Y}_+ = \underline{Y}.$$

First, $\mathcal{A}$ does not entail $Z \not\rightsquigarrow Y$ since $\mathcal{G}(\mathcal{A})$ contains a DAG satisfying $Z \rightsquigarrow Y$. Further, $\mathcal{A}$ does not entail any constraint in $\overline{Z} \not\rightsquigarrow \underline{Y}$, by Lemma 2.

Second, $\mathcal{A}_+$ entails $Z \not\rightsquigarrow Y$ since $\mathcal{G}(\mathcal{A})$ does not contain a DAG satisfying both $X \rightsquigarrow Z$ and $Z \rightsquigarrow Y$. Further, $\mathcal{A}_+$ entails a constraint in $\overline{Z}_+ \not\rightsquigarrow \underline{Y}_+$, by Lemma 2.

Let $Z' \not\rightsquigarrow Y'$ denote one of the entailed constraints in $\overline{Z}_+ \not\rightsquigarrow \underline{Y}_+$, where $Z' \in \overline{Z}_+$ and $Y' \in \underline{Y}_+$. The constraint $Z' \not\rightsquigarrow Y'$ could not have been in $\overline{Z} \not\rightsquigarrow \underline{Y}$; otherwise $Z \not\rightsquigarrow Y$ would be entailed by $\mathcal{A}$, by Lemma 6. Hence, the constraint $Z' \not\rightsquigarrow Y'$ must have a $Z' \in \overline{Z}_+$ that is in $\overline{X}$ but not in $\overline{Z}$ (and otherwise $Y' \in \underline{Y}_+ = \underline{Y}$). That is, $Z' \not\rightsquigarrow Y'$ is a constraint in $\overline{X} \not\rightsquigarrow \underline{Y}$. Hence, $\mathcal{A}$ entails $X \not\rightsquigarrow Y$, again by Lemma 6. $\square$

Proof of Theorem 8. ($\Leftarrow$) First, for any $Z \in \overline{X} \cup \underline{Y}$, there is no path from X to Y through Z, since by Definition 1 either $Z \rightsquigarrow X$ or $Y \rightsquigarrow Z$, and thus by Lemma 7

either $X \not\twoheadrightarrow Z$ or $Z \not\twoheadrightarrow Y$. Second, for any $Z \notin \overline{X} \cup \underline{Y}$, there is no path from X to Y through Z, since either $\overline{X} \not\twoheadrightarrow \underline{Z}$ or $\overline{Z} \not\twoheadrightarrow \underline{Y}$, and thus by Lemma 6 either $X \not\twoheadrightarrow Z$ or $Z \not\twoheadrightarrow Y$. Since there is no path from X to Y through any other variable Z, for $\mathcal{A}$ to entail $X \rightsquigarrow Y$, then $\mathcal{A}$ must also entail the edge $X \rightarrow Y$.

($\Rightarrow$) Assume for contradiction that $X \rightsquigarrow Y$ is not entailed by $\mathcal{A}$. In this case there exists a DAG in $\mathcal{G}(\mathcal{A})$ where $X \not\twoheadrightarrow Y$, and hence $X \not\rightarrow Y$, which is a contradiction.

Assume instead that $X \rightsquigarrow Y$ is entailed by $\mathcal{A}$, but there exists a variable $Z \notin \overline{X} \cup \underline{Y}$ such that $\mathcal{A}$ entails no constraint in $\overline{X} \not\twoheadrightarrow \underline{Z}$ and no constraint in $\overline{Z} \not\twoheadrightarrow \underline{Y}$. Hence, there is a DAG in $\mathcal{G}(\mathcal{A})$ satisfying $X \rightsquigarrow Z$, and a DAG satisfying $Z \rightsquigarrow Y$. By Lemma 9, there must also be a DAG satisfying $X \rightsquigarrow Y$ (if there was not, then $X \not\twoheadrightarrow Y$, which would be a contradiction). Thus, there must also be a DAG in $\mathcal{G}(\mathcal{A})$ without the edge $X \rightarrow Y$ (since removing the edge does not violate any positive ancestral constraints), which is a contradiction. □

D.6 Proofs for Section 6.5.3

We assume here that sets of ordering constraints $\mathcal{O}$ are closed, i.e., if $X < Y$ and $Y < Z$ are in $\mathcal{O}$, then $X < Z$ is in $\mathcal{O}$. Given an arbitrary set of ordering constraints, we can compute its closure efficiently using standard algorithms.

We say that a set of ancestral constraints $\mathcal{A}$ is maximal if any constraint that is entailed by $\mathcal{A}$ is also an element of $\mathcal{A}$. We can obtain a maximal set of ancestral constaints, using Lemmas 1 & 2, which allows us to enumerate all constrained entailed by $\mathcal{A}$. For the rest of this section, we assume that all sets of ancestral constraints $\mathcal{A}$ are maximal.

Note that a DAG G is a (consistent) set of ordering constraints $\mathcal{O}$, and vice versa. Both are also partially ordered sets (posets). Further, a set of ordering constraints $\mathcal{O}$ is consistent (i.e., it corresponds to a non-empty set of orderings) iff the directed graph induced by $\mathcal{O}$ is acyclic.

Hence, a DAG G is compatible with a set of ordering constraints $\mathcal{O}$ iff the union of their edges yield a graph that is acyclic. We define another type of graph below, which is acyclic iff a DAG G is compatiable with a set $\mathcal{O}$.

Definition 3. *For a given DAG $G \in \mathcal{G}(\mathcal{A})$ and a set of ordering constraints $\mathcal{O}$, let $G^{\mathcal{O}}$ denote the extension of G with respect to $\mathcal{O}$, which is a directed graph found by taking DAG G and adding dashed directed edges $X \dashrightarrow Y$ iff (1) $X < Y \in \mathcal{O}$ and (2) $X \not\rightarrow Y$ in G.*

Let $\mathcal{G}(\mathcal{A}, \mathcal{O}) = \{G^{\mathcal{O}} \mid G \in \mathcal{G}(\mathcal{A})\}$ denote the set of all extensions, with respect to $\mathcal{A}$ and $\mathcal{O}$.

Lemma 10. *$\mathcal{A}$ entails $\mathcal{O}$ iff all graphs $G^{\mathcal{O}} \in \mathcal{G}(\mathcal{A}, \mathcal{O})$ do not contain directed cycles.*

Hence, in Theorem 9, it suffices to consider the extensions $G^{\mathcal{O}}$ of $\mathcal{G}(\mathcal{A}, \mathcal{O})$. In our proof of Theorem 9, we make use of the following lemmas.

Lemma 11. *For a DAG $G \in \mathcal{G}(\mathcal{A})$, if the extension $G^{\mathcal{O}}$ contains an edge $X \dashrightarrow Y$, then $X \rightsquigarrow Y \notin \mathcal{A}$.*

Proof. By Defintion 3, we add an edge $X \dashrightarrow Y$ in $G^{\mathcal{O}}$ iff $X \not\rightarrow Y$ in G. Since G is a DAG in $\mathcal{G}(\mathcal{A})$ without a path from X to Y, then $X \rightsquigarrow Y$ is not entailed by $\mathcal{A}$. □

Lemma 12. *Let $G^{\mathcal{O}}$ be an extension in $\mathcal{G}(\mathcal{A}, \mathcal{O})$ with a path $X \rightarrow Y \rightarrow Z$. There exists another extension $H^{\mathcal{O}}$ in $\mathcal{G}(\mathcal{A}, \mathcal{O})$ which also has the edge $X \rightarrow Z$.*

Proof. The extension $G^{\mathcal{O}}$ has a corresponding DAG $G \in \mathcal{G}(\mathcal{A})$. Since $X \rightsquigarrow Z$ in G, by Lemma 8, there is another DAG $H \in \mathcal{G}(\mathcal{A})$ which also includes the edge $X \rightarrow Z$. Hence, the extension $H^{\mathcal{O}}$ is also in $\mathcal{G}(\mathcal{A}, \mathcal{O})$. □

Lemma 13. *Let $G^{\mathcal{O}}$ be an extension in $\mathcal{G}(\mathcal{A}, \mathcal{O})$ with a path $X \dashrightarrow Y \dashrightarrow Z$. There exists another extension $H^{\mathcal{O}}$ in $\mathcal{G}(\mathcal{A}, \mathcal{O})$ which also has an edge $X \rightarrow Z$ or $X \dashrightarrow Z$.*

Proof. If $X \dashrightarrow Y$ and $Y \dashrightarrow Z$ are edges in $G^{\mathcal{O}}$, then $X < Y$ and $Y < Z$ are constraints in $\mathcal{O}$. Since $\mathcal{O}$ is closed, $X < Z$ is also in $\mathcal{O}$. Hence, edge $X \dashrightarrow Z$ is also in $G^{\mathcal{O}}$, unless there is a path $X \rightsquigarrow Z$. By Lemma 12, then there must be an extension $H^{\mathcal{O}}$ with the edge $X \to Z$. □

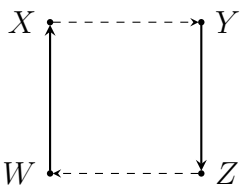
To prove Theorem 9, we prove the contrapositive. That is, if there is an extension $G^{\mathcal{O}} \in \mathcal{G}(\mathcal{A}, \mathcal{O})$ with a directed cycle, then it violates one of the conditions of Theorem 9. We prove this by induction on the length of the directed cycle in $G^{\mathcal{O}}$. In such cases, we may refer to a cycle of length m as an m-cycle.

Proof of Theorem 9. First, by construction, we know that there are no cycles of purely solid edges (since G is acyclic), and no cycles of purely dashed edges (since $\mathcal{O}$ is consistent). We can thus assume, without loss of generality, that the "first" edge of our cycle is a dashed edge $X \dashrightarrow Y$, since we can pick any edge to be the "first" edge.

We first consider cycles of length $m = 2$. Given the above assumptions, it suffices to consider the cycle $X \dashrightarrow Y \to X$, which violates Condition 1 (if $X < Y$ is in $\mathcal{O}$ then $Y \not\to X$ is in $\mathcal{A}$).

For cycles of length $m = 3$: either we have two dashed edges followed by one solid edge, or one dashed edge followed by two solid edges. We can reduce a 3-cycle to a 2-cycle, using either Lemma 12 or 13, which shows that Condition 1 is violated.

For cycles of length $m = 4$: if the cycle has consecutive solid edges or consecutive dashed edges, we can use Lemmas 12 and 13 to reduce a 4-cycle to a 2-cycle or 3-cycle, which are handled above. Otherwise, we must have a cycle



First, we have dashed edges $X \dashrightarrow Y$ and $Z \dashrightarrow W$, so by Lemma 11, constraints

$X \rightsquigarrow Y$ and $Z \rightsquigarrow W$ are not in $\mathcal{A}$. Second, constraints $X \rightsquigarrow Z$ and $Y \rightsquigarrow W$ are not in $\mathcal{A}$, otherwise they would have created a shorter cycle. Finally, we have solid edges $W \rightarrow X$ and $Y \rightarrow Z$, so the constraints $X \rightsquigarrow W$ and $Y \not\rightsquigarrow Z$ are not in $\mathcal{A}$. Hence, Condition 2 is violated.

Given the cases of cycle lengths of $m-1$ or fewer, we now consider cycles of length m. First, it suffices to consider cycles where dashed and solid edges alternate. Otherwise, we can use Lemma 12 or 13 to produce a shorter cycle. Second, we can assume that there is no directed path of solid edges connecting two non-adjacent nodes in the cycle; we can again obtain a shorter cycle in this case. Third, we note that any odd-length cycle must have two consective edges of the same type. Hence, it suffices to consider only cycles of even length:

$$X \dashrightarrow Y \rightarrow Z \dashrightarrow W \rightarrow \cdots \dashrightarrow V \rightarrow X$$

(we assume without loss of generality that the "first" edge is dashed). First, we have dashed edges $X \dashrightarrow Y$ and $Z \dashrightarrow W$, so by Lemma 11, constraints $X \rightsquigarrow Y$ and $Z \rightsquigarrow W$ are not in $\mathcal{A}$. Second, constraints $X \rightsquigarrow Z$ and $Y \rightsquigarrow W$ and $X \rightsquigarrow W$ are not in $\mathcal{A}$, otherwise they would have created a shorter cycle. Finally, we have a solid edge $Y \rightarrow Z$ so the constraint $Y \not\rightsquigarrow Z$ is not in $\mathcal{A}$. Hence, Condition 2 is violated. □

APPENDIX E

Proofs of Theorems in Chapter 7

Moral Graph

Consider $P(B_1|O)$. Let $S_1 = \sum_{i=0}^{k-2} \binom{z-3}{i}$ and $S_2 = \sum_{i=0}^{k} \binom{z-1}{i}$. Let i be a positive integer. Note that

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}. \tag{E.1}$$

Then similar to Subsection 7.4.1,

$$\begin{aligned}
S_1 &= \frac{1}{z-2} \sum_{i=0}^{k-1} i \binom{z-2}{i} \quad \big(\text{by (7.1)}\big), \qquad (E.2)\\
S_1 &= \frac{1}{z-2} \sum_{i=0}^{k-2} (z-2-i) \binom{z-2}{i} \quad \big(\text{by (7.2)}\big)\\
&= \frac{1}{z-1} \sum_{i=0}^{k-2} (z-1-i) \binom{z-1}{i} - S_1 + \binom{z-3}{k-2} \big(\text{by (7.1, 7.2, E.2)}\big)\\
&= S_2 - \binom{z}{k} - \frac{1}{z-1} \sum_{i=0}^{k-2} i \binom{z-1}{i} - S_1 + \binom{z-3}{k-2} \big(\text{by (E.1)}\big).
\end{aligned}$$

Let $S_3 = \sum_{i=0}^{k-2} i \binom{z-1}{i}$. Then

$$\begin{aligned}
S_3 &= \sum_{i=0}^{k-3} (z-1-i) \binom{z-1}{i} \quad \big(\text{by (7.1) (7.2)}\big)\\
&= -S_3 + (z-1) \left(S_2 - \binom{z}{k} - \binom{z-1}{k-2} + \binom{z-2}{k-3} \right) \big(\text{by (7.1, E.1)}\big).
\end{aligned}$$

Consequently

$$\frac{S_1}{S_2} = \frac{1}{4} \left(1 + \frac{-\binom{z}{k} + \binom{z-1}{k-2} - \binom{z-2}{k-3} + 2\binom{z-3}{k-2}}{\sum_{i=0}^{k} \binom{z-1}{i}} \right)$$

and again we approximate $\sum_{i=0}^{k} \binom{z-1}{i}$ by $\binom{z}{k}$. Then

$$P(B_1|O) \approx \frac{k(k-1)}{4z^2}\left(3-\frac{2(k+1)}{z}\right).$$

Expected Size of E_m

Let A denote the event that in G'_A there does not exist $Z \to Z'$ and there exists a path of Type (a). Similarly, let B, C denote that for Type (b), (c) paths. Then

$$\begin{aligned} E[I_{S'}] \leq & \sum_{x<z<z'<z''<y} P(A|O)P(O) \\ & + \sum_{y<z<z'<z''<x} P(B|O)P(O) \\ & + \sum_{\substack{z<x<z'<z''<y \\ z<z'<x<z''<y \\ z<z'<z''<x<y \\ z<y<z'<z''<x \\ z<z'<y<z''<x \\ z<z'<z''<y<x}} P(C|O)P(O). \end{aligned}$$

Consider the first term. Recall Section 7.5. We approximate the probability that there exists $W \to Y \leftarrow W'$ with $3k(k-1)/(4y^2)$, and approximate the probability that v-structure $Z \to Z'' \leftarrow Z'$ exists with $3k(k-1)/(4z''^2)\,(1-k/z') = 3k(k-1)(z'-k)/(4z'z''^2)$. Then similar to Subsection 7.6.2.1,

$$P(A|O) < \frac{9k^3(k-1)^2(z'-k)(z-x-1)(y-z'-1)(y-z''-1)}{16y^2zz'z''^2}$$

and

$$\sum_{x<z<z'<z''<y} P(A|O)P(O) < \frac{5k^3(k-1)^2}{3072n^2}.$$

The case for the second term is the same as the first. Then consider the third term $\sum P(C|O)P(O)$. First note that the case for $\sum_{z<x<z'<z''<y} P(C|O)P(O)$ is essentially identical to $\sum_{x<z<z'<z''<y} P(A|O)P(O)$. Then when $z < z' < x < z'' < y$,

$$P(C|O) < \frac{9k^3(k-1)^2(z'-k)(x-z-1)(y-z'-1)(y-z''-1)}{16xy^2z'z''^2}$$

and

$$\sum_{z<z'<x<z''<y} P(C|O)P(O) < \frac{49k^3(k-1)^2}{18432n^2}.$$

Then consider when $z < z' < z'' < x < y$. Recall that Z'' is an ancestor of X or Y, or both. First consider when Z'' is an ancestor of X:

$$P(C|O) < \frac{9k^3(k-1)^2(z'-k)(x-z-1)(x-z''-1)(y-z'-1)}{16x^2yz'z''^2}.$$

Then consider when Z'' is an ancestor of Y:

$$P(C|O) < \frac{9k^3(k-1)^2(z'-k)(x-z-1)(y-z'-1)(y-z''-1)}{16xy^2z'z''^2}.$$

Consequently

$$\sum_{z<z'<z''<x<y} P(C|O)P(O) < \frac{251k^3(k-1)^2}{6144n^2}.$$

The remaining cases are the same as the above.

REFERENCES

[AD96] S. Acid and L. M. De Campos. "An algorithm for finding minimum d-separating sets in belief networks." In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pp. 3–10, 1996.

[AS08] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley, third edition, 2008.

[AZ14] B. Aragam and Q. Zhou. "Concave penalized estimation of sparse Gaussian Bayesian networks." *Journal of Machine Learning Research*, **16**:2273?2328, 2014.

[BL13] K. Bache and M. Lichman. "UCI Machine Learning Repository.", 2013.

[BE84] A. B. Barak and P. Erdös. "On the maximal number of strongly independent vertices in a random acyclic directed graph." *SIAM Journal on Algebraic Discrete Methods*, **5**(4):508–514, 1984.

[BC13] M. Bartlett and J. Cussens. "Advances in Bayesian Network Learning using Integer Programming." In *Proceedings of the Twenty-Nineth Conference on Uncertainty in Artificial Intelligence*, 2013.

[BC15] M. Bartlett and J. Cussens. "Integer Linear Programming for the Bayesian network structure learning problem." *Artificial Intelligence*, 2015.

[Bol01] B. Bollobás. *Random graphs*. Cambridge University Press, second edition, 2001.

[BT12] G. Borboudakis and I. Tsamardinos. "Incorporating causal prior knowledge as path-constraints in Bayesian networks and maximal ancestral graphs." In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, 2012.

[BT13] G. Borboudakis and I. Tsamardinos. "Scoring and searching over Bayesian networks with causal and associative priors." In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013.

[Bou93] R. R. Bouckaert. "Probalistic Network Construction Using the Minimum Description Length Principle." In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pp. 41–48, 1993.

[BW91] G. Brightwell and P. Winkler. "Counting linear extensions." *Order*, **8**(3):225–242, 1991.

[Bun91] W. Buntine. "Theory refinement on Bayesian networks." In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 52–60, 1991.

[CCD15] E. Y.-J. Chen, A. Choi, and A. Darwiche. "Learning Optimal Bayesian Networks with DAG Graphs." In *Proceedings of the 4th IJCAI Workshop on Graph Structures for Knowledge Representation and Reasoning*, 2015.

[CCD16] E. Y.-J. Chen, A. Choi, and A. Darwiche. "Enumerating equivalence classes of Bayesian networks using EC graphs." In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Statistics*, 2016.

[CP14] E. Y.-J. Chen and J. Pearl. "Random Bayesian network with bounded indegree." In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, 2014. To appear.

[CSC16] E. Y.-J. Chen, Y. Shen, A. Choi, and A. Darwiche. "Learning Bayesian networks with ancestral constraints." 2016. Submitted to Neural Information Processing Systems.

[CT14] Y. Chen and J. Tian. "Finding the k-Best Equivalence Classes of Bayesian Network Structures for Model Averaging." In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2014.

[Chi95] D. M. Chickering. "A transformational characterization of equivalent Bayesian network structures." In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 87–98, 1995.

[Chi02] D. M. Chickering. "Learning equivalence classes of Bayesian network structures." *Journal of Machine Learning Research*, **2**:445–498, 2002.

[CGH95] D. M. Chickering, D. Geiger, and D. Heckerman. "Learning Bayesian networks: Search methods and experimental results." In *Proceedings of Fifth Conference on Artificial Intelligence and Statistics*, pp. 112–128, 1995.

[CHM04] D. M. Chickering, D. Heckerman, and C. Meek. "Large-sample learning of Bayesian networks is NP-hard." *Journal of Machine Learning Research*, **5**:1287–1330, 2004.

[CM02] D. M. Chickering and C. Meek. "Finding optimal Bayesian networks." In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 94–102, 2002.

[CL68] C. K. Chow and C. N. Liu. "Approximating discrete probability distributions with dependence trees." *IEEE Transactions on Information Theory*, **14**(3):462–467, 1968.

[CRS03] R. Cohen, A. F. Rozenfeld, N. Schwartz, D. Ben-Avraham, and S. Havlin. "Directed and non-directed scale-free networks." In *Statistical Mechanics of Complex Networks*, pp. 23–45. Springer, 2003.

[CH92] G. F. Cooper and E. Herskovits. "A Bayesian method for the induction of probabilistic networks from data." *Machine Learning*, **9**(4):309–347, 1992.

[Cus08] J. Cussens. "Bayesian network learning by compiling to weighted MAX-SAT." In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pp. 105–112, 2008.

[Cus11] J. Cussens. "Bayesian network learning with cutting planes." In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 153–160, 2011.

[CBJ13] J. Cussens, M. Bartlett, E. M. Jones, and N. A. Sheehan. "Maximum likelihood pedigree reconstruction using integer linear programming." *Genetic epidemiology*, **37**(1):69–83, 2013.

[Dar09] A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[DFM12] R. Dechter, N. Flerova, and R. Marinescu. "Search algorithms for m best solutions for graphical models." In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, 2012.

[DMS01] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin. "Giant strongly connected component of directed networks." *Physical Review E*, **64**(2):025101, 2001.

[EDL02] R. Edgar, M. Domrachev, and A. E. Lash. "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository." *Nucleic acids research*, **30**(1):207–210, 2002.

[FYM14] X. Fan, C. Yuan, and B. Malone. "Tightening bounds for Bayesian network structure learning." In *In Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2014.

[FGS12] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaeffer, and R. Holte. "Partial-Expansion A* with Selective Node Generation." In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, 2012.

[FHT01] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, 2001.

[FK00] N. Friedman and D. Koller. "Being Bayesian about network structure." In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 201–210, 2000.

[FLN00] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. "Using Bayesian networks to analyze expression data." *Journal of computational biology*, **7**(3-4):601–620, 2000.

[FNP99] N. Friedman, I. Nachman, and D. Peér. "Learning bayesian network structure from massive datasets: the sparse candidate algorithm." In *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence*, pp. 206–215, 1999.

[FZ13] F. Fu and Q. Zhou. "Learning sparse causal Gaussian networks with experimental intervention: regularization and coordinate descent." *Journal of the American Statistical Association*, **108**(501):288–300, 2013.

[GP01] S. B. Gillispie and M. D. Perlman. "Enumerating markov equivalence classes of acyclic digraph dels." In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 171–177, 2001.

[GKO94] R. L. Graham, D. E. Knuth, and Patashnik O. *Concrete mathematics: a foundation for computer science*. Addison-Wesley, second edition, 1994.

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *IEEE Trans. Systems Science and Cybernetics*, **4**(2):100–107, 1968.

[Har05] A. Hartemink. "Banjo (Bayesian Network Inference with Java Objects)." 2005.

[Hec98] D. Heckerman. *A tutorial on learning with Bayesian networks*. Springer, 1998.

[HGC95] D. Heckerman, D. Geiger, and D. M. Chickering. "Learning Bayesian networks: The combination of knowledge and statistical data." *Machine Learning*, **20**(3):197–243, 1995.

[IC02] J. S. Ide and F. G. Cozman. "Random generation of Bayesian networks." In *Advances in Artificial Intelligence*, pp. 366–376. Springer, 2002.

[JSG10] T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. "Learning Bayesian network structure using LP relaxations." In *Proceedings of the Thirteen International Conference on Artificial Intelligence and Statistics*, pp. 358–365, 2010.

[JLR00] S. Janson, T. Luczak, and A. Rucinski. *Random graphs*. Cambridge University Press, 2000.

[KN09] B. Karrer and M. E. J. Newman. "Random graph models for directed acyclic networks." *Physical Review E*, **80**(4):046110, 2009.

[KS04] M. Koivisto and K. Sood. "Exact Bayesian structure discovery in Bayesian networks." *The Journal of Machine Learning Research*, **5**:549–573, 2004.

[KF09] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[LB93] W. Lam and F. Bacchus. "Using causal information and local measures to learn Bayesian networks." In *Proceedings of the Ninth international conference on Uncertainty in Artificial Intelligence*, pp. 243–250, 1993.

[LB94] W. Lam and F. Bacchus. "Learning Bayesian Belief Networks: An Approach Based on the MDL Principle." *Computational Intelligence*, **10**:269–294, 1994.

[MGR95] D. Madigan, J. Gavrin, and A. E. Raftery. "Eliciting prior information to enhance the predictive performance of Bayesian graphical models." *Communications in Statistics-Theory and Methods*, **24**(9):2271–2292, 1995.

[MKJ14] B. Malone, K. Kangas, M. Järvisalo, M. Koivisto, and P. Myllymäki. "Predicting the hardness of learning Bayesian networks." In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2014.

[Mee95] C. Meek. "Causal inference and causal explanation with background knowledge." In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, 1995.

[MW03] A. Moore and W.-K. Wong. "Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning." In *Proceedings of the 20th International Conference on Machine Learning*, volume 3, pp. 552–559, 2003.

[Mur12] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[Nea04] R. E. Neapolitan. *Learning Bayesian networks*. Pearson Prentice Hall, 2004.

[New09] M. E. J. Newman. *Networks: an introduction*. Oxford University Press, 2009.

[NK13] T. M. Niinimäki and M. Koivisto. "Annealed Importance Sampling for Structure Learning in Bayesian Networks." In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[OSM15] C. J. Oates, J. Q. Smith, S. Mukherjee, and J. Cussens. "Exact estimation of multiple directed acyclic graphs." *Statistics and Computing*, pp. 1–15, 2015.

[PK13] P. Parviainen and M. Koivisto. "Finding Optimal Bayesian Networks Using Precedence Constraints." *Journal of Machine Learning Research*, **14**(1):1387–1415, 2013.

[Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[PV91] J. Pearl and T. S. Verma. "A theory of inferred causation." In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pp. 441–452, 1991.

[PIM08] E. Perrier, S. Imoto, and S. Miyano. "Finding Optimal Bayesian Network Given a Super-Structure." *Journal of Machine Learning Research*, **9**(10), 2008.

[PT01] B. Pittel and R. Tungol. "A phase transition phenomenon in a random directed acyclic graph." *Random Structures & Algorithms*, **18**(2):164–184, 2001.

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010.

[SM10] A. Shojaie and G. Michailidis. "Penalized likelihood methods for estimation of sparse high-dimensional directed acyclic graphs." *Biometrika*, **97**(3):519–538, 2010.

[SM06] T. Silander and P. Myllymäki. "A Simple approach for finding the globally optimal Bayesian network structure." In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 445–452, 2006.

[SM05] A. P. Singh and A. W. Moore. "Finding optimal Bayesian networks by dynamic programming." Technical report, CMU-CALD-050106, 2005.

[SGS00] P. Spirtes, C. N. Glymour, and R. Scheines. *Causation, prediction, and search*. MIT press, 2000.

[Suz93] J. Suzuki. "A Construction of Bayesian Networks from Databases Based on an MDL Principle." In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 266–273, 1993.

[THR10] J. Tian, R. He, and L. Ram. "Bayesian model averaging using the k-best Bayesian network structures." In *Proceedings of the Twenty-Six Conference on Uncertainty in Artificial Intelligence*, pp. 589–597, 2010.

[TPP98] J. Tian, J. Pearl, and A. Paz. "Finding minimal d-separators." Technical report, UCLA Cognitive Systems Laboratory, 1998.

[VP90] T. Verma and J. Pearl. "Equivalence and synthesis of causal models." In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 220–227, 1990.

[VP92] T. Verma and J. Pearl. "An algorithm for deciding if a set of observed independencies has a causal explanation." In *Proceedings of the Eighth international Conference on Uncertainty in Artificial Intelligence*, pp. 323–330, 1992.

[YMI00] T. Yoshizumi, T. Miura, and T. Ishida. "A* with Partial Expansion for Large Branching Factor Problems." In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2000.

[YM13] C. Yuan and B. Malone. "Learning Optimal Bayesian Networks: A Shortest Path Perspective." *Journal of Artificial Intelligence Research*, **48**:23–65, 2013.

[YMW11] C. Yuan, B. Malone, and X. Wu. "Learning optimal Bayesian networks using A* search." In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 2186–2191, 2011.